SUBLINEAR GEOMETRIC ALGORITHMS AND GEOMETRIC LOWER BOUNDS

DING LIU

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

MAY 2005

Abstract

This thesis consists of two parts unified under the common theme that both of them are concerned with the complexity of geometric problems.

The first part of this thesis initiates a study of sublinear algorithms for geometric problems in two and three dimensions when no preprocessing is allowed. The problems we consider include intersection detection of convex polygons and polyhedra, point location in two-dimensional Voronoi diagrams and triangulations, ray shooting towards convex polyhedra, and nearest neighbor type problems. Our (randomized) algorithms read only a small fraction of the input. Unlike their predecessors, our algorithms never err and randomization only affects the running time but not the correctness of the output. For each problem considered (with input size n), we achieve expected running time of $O(\sqrt{n})$, which we show to be optimal. We also approximate, for any fixed $\varepsilon > 0$, the volume of a n-vertex convex polytope and the length of the shortest path between two points on its boundary, to within a relative factor of $1 + \varepsilon$ and in expected time $O(\sqrt{n})$.

In the second part of this thesis, we prove strong lower bounds for geometric problems in standard models of computation. We prove several near-optimal lower bounds for intersection searching problems in two and three dimensions in the pointer machine model. Our lower bounds resolve a couple of long-standing open problems in computational geometry. We show that: (1) The two-dimensional generalization of fractional cascading is impossible; (2) There exists a n-vertex convex polytope that admits of no boundary dominant Dobkin-Kirkpatrick hierarchy, for any n large enough. We also prove a near-optimal deterministic query time lower bound for Approximate Nearest Neighbor Searching – a basic problem in computational geometry with a variety of applications – in the cell probe model. This lower bound holds in a very special space (the Hamming Cube) and for a very loose approximation factor.

Acknowledgments

First and foremost, I am forever indebted to my advisor Bernard Chazelle, whose generosity, enthusiasm, broad and deep knowledge of mathematics and theoretical computer science, and ability to see the mathematical essence of a complicated problem has constantly amazed me. Bernard introduced me to the fascinating area of theoretical computer science. He has taught me how to practice and enjoy research: how to understand a problem from a fundamental level, how to improve the solutions relentlessly, and how to present the key ideas clearly. Being an excellent speaker himself, Bernard also taught me how to give talks, which I found extremely useful. I cannot thank him enough for his enormous guidance and encouragement in my research, as well as the financial support he has provided over the years.

I am grateful to Sanjeev Arora and Moses Charikar for serving on my dissertation committee as readers and for their careful reading of this thesis. I am grateful to David Dobkin and Robert Sedgewick for serving on my dissertation committee as non-readers.

I am also grateful to Andy Yao for his encouragement during my early years at Princeton. One of my research papers originated from attending his research seminar in Spring 2001. I thank Andy for introducing to me the problem studied in that paper. Besides, Andy is one of the best teachers I have seen. I was also impressed by his enthusiasm for attacking fundamental open problems.

Many results presented in this thesis are based on collaborations. I am grateful to my co-authors Bernard Chazelle and Avner Magen. In particular, most results in the first part of this thesis are taken from joint work with Bernard and Avner, and results in Chapter 4 are taken from joint work with Bernard.

It is impossible to overstate the influence of the theory group of Princeton Computer Science on my research. I enjoyed attending theory courses in our department as well as our weekly theory lunch seminars. I thank all theory faculty members who shaped my broad perspective on theoretical computer science by teaching these courses and organizing these seminars. I also thank my fellow graduate students in the theory group who influenced my research directly or indirectly. At the risk of leaving out many of them, I acknowledge Nir Ailon, Amit Chakrabarti, Seshadhri Comandur, Subhash Khot, Manoj Prabhakaran, Yaoyun Shi, and Shengyu Zhang.

Thanks also go to Melissa Lawson who helped me on many important things. For example, she wrote a letter for me on behalf of the department to consulate officers of the United States Embassy in Beijing. This letter (along with another letter from Bernard) helped my wife getting her visa to the United States to accompany me during my final year of PhD.

I would like to take this opportunity to thank all my friends who made my life at Princeton so enjoyable. Amongst them, I am especially thankful to Ming Zhang, Xinming Ou, Li Shang, and Yulei Luo for their help and support.

Finally, my infinite thanks to my parents and my wife Yuan for their love and support throughout the years.

Ding Liu

I gratefully acknowledge my advisor Bernard Chazelle's NSF Grant CCR-96-23768, NSF grant CCR-998817, NSF grant 0306283, ARO Grant DAAH04-96-1-0181, and the George Van Ness Lothrop Fellowship of Princeton University Graduate School for supporting my PhD research at Princeton University.

Contents

List of Figures

Chapter 1

Introduction

This thesis is concerned with the complexity of geometric problems in terms of both upper and lower bounds. It has two independent parts: sublinear geometric algorithms and geometric lower bounds.

The first part of this thesis initiates a study of sublinear algorithms for geometric problems in two and three dimensions when no preprocessing is allowed. The problems we consider include intersection detection of convex polygons and polyhedra, point location in two-dimensional Voronoi diagrams and triangulations, ray shooting towards convex polyhedra, and nearest neighbor type problems. For each of these problems with input size n, our new algorithm achieves expected running time of $O(\sqrt{n})$, which we show to be optimal. Given a n-vertex convex polytope, we can also approximate its volume, or the length of the shortest path between two points on its boundary, to within arbitrary fixed precision in expected time $O(\sqrt{n})$. These new results are presented in Chapters 2 and 3.

In the second part of this thesis, we prove strong lower bounds for several geometric problems. In Chapter 4, we prove near-optimal lower bounds for intersection searching problems in two and three dimensions. As a result, these lower bounds resolve some long-standing open problems in computational geometry. In Chapter 5, we prove a near-optimal lower bound on the query time of any deterministic algorithms for Approximate Nearest Neighbor Searching, a fundamental problem in computational geometry.

Before describing our specific new results, we begin with a brief overview of sublinear algorithms (Section 1.1) and geometric lower bounds (Section 1.2).

1.1 Sublinear Algorithms

With the current information explosion, massive data sets are now pervasive. The most familiar example is today's Internet: the number of webpages has already exceeded a billion and is still growing fast. In telecommunication industries, big phone companies collect huge amount of call data everyday. Massive data sets also appear in scientific simulation and computation, electronic commerce, and business transactions. Fundamentally new algorithms are needed to organize, process, and analyze this vast amount of information. This is because traditional algorithmic techniques may be of little use for massive data sets. For example, a linear-time algorithm, which is considered as the ultimate efficiency goal for many years, may take several days to finish on massive data sets. In some other situations, the amount of data is so enormous that it is impossible (or impractical) to store the whole input. There is thus a need to develop algorithms that use sublinear amount of time or space resources – the so-called sublinear algorithms. In Section 1.1.1, we give a brief overview of the different types of sublinear algorithms developed in Theoretical Computer Science.

1.1.1 Overview of Sublinear Algorithms

Property Testing

One class of sublinear algorithms that has been extensively studied in the last ten years is property testing, a relaxation of the standard decision problem. In a decision problem, we must determine whether an input object has a property $\mathcal{P}$ or not. Unfortunately, this could be very hard. For example, it is NP-complete to decide whether an arbitrary graph is 3-colorable or not [89]. In property testing, we only need to give correct answers (with high probability) when the input has $\mathcal{P}$ or when it is far from doing so. Surprisingly, this relaxation brings in sublinear-time property testers for a long list of combinatorial, algebraic, and geometric problems.

Let's focus on the "graph 3-colorability" problem mentioned earlier. We say that a n-vertex graph G is ε-far from being 3-colorable for some $\varepsilon > 0$, if after deleting any subset of εn^2 edges of G the remaining graph is still not 3-colorable. In property testing, we only need to give an algorithm that accepts every 3-colorable graph and rejects every graph that is ε-far from being 3-colorable with probability at least $2/3$. Such an algorithm is called an ε-tester for 3-colorability.[1] Consider the following simple algorithm. It extracts uniformly at random a subset T of vertices of the graph G. Let G_T be the induced subgraph of G on T. It then tests 3-colorability of G_T using the naive exponential time algorithm, and accepts G if and only if G_T is 3-colorable. If G is 3-colorable then G_T is also 3-colorable. On the other hand, Goldreich et al. [90] proved that if G is ε-far from being 3-colorable, then with probability at least $1/2$, G_T is not 3-colorable for a random

[1] Since we require the algorithm to accept every 3-colorable graph, it is called an one-sided error tester. There are other property testers that may also err with small probability when the object has the property. They are called two-sided error testers. For one-sided error tester, the choice of success probability $2/3$ is arbitrary, and any fi xed constant $c > 0$ can be used. We can repeat the algorithm enough times to achieve the desired success probability.

T of size $O(1/\varepsilon^3)$. The size of T was later improved to $O(1/\varepsilon^2)$ by Alon and Krivelevich [11]. This illustrates the main advantage of property testing, that is, a property tester runs in time sublinear on, or even independent of, the size of the input.

There is a huge literature on property testing. We recommend the surveys by Ron [125] and Fischer [82] to readers who want to pursue this topic further.

Streaming Algorithms

In the past few years, the data streaming model was extensively studied by the Theoretical Computer Science community. In this model, the input comes as a continuous stream and the algorithm must process each input element as it passes. Once the algorithm has seen an element, it is gone forever.[2] Moreover, the algorithm does not have enough space to archive the whole input. It thus must maintain a succinct "synopsis" of the data seen so far. Typically, a data streaming algorithm uses poly-logarithmic amount of space and spends poly-logarithmic processing time for each element of the input.

Let's focus on a specific problem: estimating the zero-th frequency moment. Given a sequence S of n elements from the domain $[m] = \{1, \ldots, m\}$, the zero-th frequency moment of S is the number of distinct elements in S. We denote it by $F_0 = F_0(S)$. It is obvious that we can compute F_0 using $O(m)$ space, but what happens if m is very large and we cannot afford that much space? Surprisingly, it turns out that, as long as we allow a small probability of error, it is possible to approximate F_0 to within arbitrary precision using only $O(\log m)$ bits [12, 19]. The time needed to process each element is also roughly $O(\log m)$. Here is the intuition for the algorithm of [12]. It first picks a random (linear) hash function $f : [m] \to [0, 1]$. It then computes $f(x_i)$ for each element x_i of the sequence, and maintains $f_0 = \min_{1 \le i \le n} f(x_i)$. In the end, it returns $1/f_0$ as an estimation

[2]In some variation of the data streaming model, we allow the algorithm to have a small number of passes over the input data.

of F_0. Since there are F_0 distinct elements and the hash function is chosen at random, we can think of the images of these F_0 elements as uniformly distributed in the interval $[0, 1]$. Their expected minimum is then around[3] $1/F_0$. In [12], this intuitive argument is made precise and f is chosen from a pairwise independent family of hash functions. The theorem there is that for every $c > 2$, there exists a streaming algorithm that uses $O(\log m)$ bits and computes a number Y such that $1/c \leq Y/F_0 \leq c$ with probability at least $1-2/c$. Recently, this result was improved by Bar-Yossef et al. [19] in the following way: For any $\varepsilon, \delta > 0$, there exists a streaming algorithm that approximates F_0 to within a relative error of $1+\varepsilon$ (from both sides) with probability at least $1-\delta$. The algorithm uses $\tilde{O}(\log m + 1/\varepsilon^2)$ space and processes each element in worst case $\tilde{O}(1/\varepsilon^2 \cdot \log m)$ time.[4] Alternatively, the algorithm processes each element in amortized $\tilde{O}(\log m + \log(1/\varepsilon))$ time. For simplicity, we also ignored the dependence on δ for both the space and time bounds, which is a multiplicative factor of $\log(1/\delta)$.

We avoid discussing more results in the data streaming model here, and refer interested readers to the survey by Muthukrishnan [122].

Sublinear Time Approximation Algorithms

In the last few years, sublinear time approximation algorithms have been developed for a wide range of problems, including in particular, fast matrix approximation and multiplication [1, 71, 87], approximating the edit distance between two strings [23], approximating the entropy of a discrete distribution [22], approximating the minimum spanning tree weight [50, 60], and approximate clustering [61, 98, 99, 116]. We don't try to survey all these results here. Instead, we give a flavor of sublinear approximation algorithms

[3]To be precise, the expected minimum of F_0 real numbers chosen uniformly and independently at random from $[0, 1]$ is $1/(F_0 + 1)$.

[4]In this thesis $\tilde{O}(f)$ means $O(f \log^c f)$ for some constant c.

by describing the algorithm of Chazelle et al. [50] for approximating the weight of the minimum spanning tree (MST) of a graph.

We first state the main result of [50]. Given a connected graph G in adjacency list representation with average degree d and edge weights in the set $\{1, 2, \ldots, w\}$, the algorithm of [50] approximates the MST weight of G to within a relative factor of $1 + \varepsilon$ for any $0 < \varepsilon < 1/2$. The algorithm runs in time $O(dw\varepsilon^{-2} \log(dw\varepsilon^{-1}))$ and succeeds with probability at least $3/4$. The algorithm can also be extended to the case when the weights of G are in the range $[1, w]$ but not necessarily integral, at the cost of an additional multiplicative factor of ε^{-1} in the running time. The same paper also proves an almost matching lower bound for any randomized algorithm that approximates the MST weight of such a graph G to within a factor of $1 + \varepsilon$. The main idea of [50] is to reduce the computation of the MST weight of G to counting the number of connected components in several subgraphs of G. Specifically, for each $1 \le t \le w$, let G^t be the subgraph of G consisting of all the edges with weight at most t. Denote the number of connected components in G^t by c^t. It is shown in [50] that the MST weight of G is exactly $n - w + \sum_{t=1}^{w-1} c^t$. So it suffices to estimate each c^t.

By slight abuse of notation, suppose we want to estimate the number of connected components c in a graph G with vertex set V. Here we follow the description given in [60], which is a minor modification of the original algorithm in [50]. Consider the following procedure that outputs either 0 or 1:

1. Pick a vertex $v \in V$ uniformly at random.

2. Choose a random integer X according to the probability distribution $\mathrm{Prob}[X \ge k] = 1/k$.

3. Check to see if the connected component containing v has at most X vertices or not. If yes then output 1, otherwise output 0.

Let b denote the output of this procedure and $\mathcal{C}$ denote the set of connected components of G, we have,

$$\mathbf{E}[b] = \sum_{C \in \mathcal{C}} \mathrm{Prob}[v \in C] \cdot \mathrm{Prob}[|C| \leq X] = \sum_{C \in \mathcal{C}} \frac{|C|}{n} \cdot \frac{1}{|C|} = \frac{c}{n}$$

That is, the expected value of b is proportional to c. We can repeat the above procedure s times, get s binary values $b_1, \ldots, b_s$, and use $(n \sum_{i=1}^{s} b_i)/s$ as an estimation of c. This estimation has the right expected value, and the difficult part of [50] is to bound its variance and show how to make the above procedure efficient, i.e. within the desired sublinear time bound. Using similar techniques, Czumaj and Sohler [60] showed how to $(1+\varepsilon)$-estimate the MST weight of an n-point metric space in $\tilde{O}(n/\varepsilon^{O(1)})$ time. Note that this is a sublinear time algorithm because the full description of an n-point metric space is of size $\Theta(n^2)$.

Geometric Algorithms

In computational geometry, sublinear algorithms have been found for many problems when preprocessing is allowed [3, 108]. For example, checking whether a point lies in a convex 3-polyhedron can be done in logarithmic time with linear time preprocessing. However, little of this technology is of any use with massive data sets, since examining the whole input—let alone preprocessing it—is out of the question. Large geometric data sets often call for algorithms that examine only a small fraction of the input, but it is fair to say that sublinear computational geometry is still largely unchartered territory. The first part of this thesis initiates a study of sublinear algorithms for fundamental geometric problems when no preprocessing is assumed. Before presenting our new results in Section 1.1.2, we review some related results first.

Geometric property testing. There has been work on geometric property testing, both in an approximate [58, 62, 77] and exact [112] setting. For example, Czumaj et al. [62] showed how to test if a set P of n points in $\mathbf{R}^d$ is in convex position or far from being so in sublinear time. We say that P is ε-far from being in convex position if there does not exist $Q \subset P$ such that $|Q| = \varepsilon n$ and the set $P/\ Q$ is in convex position. Given any $\varepsilon > 0$, the algorithm in [62] accepts P if it is in convex position and rejects it if it is ε-far from being so. That algorithm works by selecting, uniformly at random, a subset of size $\Theta((n^d/\varepsilon)^{1/(d+1)})$ from P. It then checks whether this subset is in convex position or not. If yes then P is accepted; otherwise P is rejected.

Approximating Euclidean minimum spanning tree weight. In [57], the authors presented several sublinear time algorithms for estimating the weight of a Euclidean minimum spanning tree (EMST) of a set P of n points in $\mathbf{R}^d$. One of their main results is an algorithm that with probability at least $3/4$ estimates the weight of a EMST of P to within a relative error of $1+\varepsilon$, for any $0 < \varepsilon < 1/2$. The algorithm runs in time $\tilde{O}(\sqrt{n}/\varepsilon^{d/2+2})$. However, the algorithm assumes that two multidimensional data structures are available: (1) an *orthogonal range query* data structure which, given an axis-parallel cube in $\mathbf{R}^d$, answers whether there is a point of P within that cube; (2) a *cone approximate nearest neighbor* data structure which, given a point $p \in P$ and a cone C, returns an approximate nearest point to p in the cone $p + C$ and from P. Besides, the algorithm needs access to a *minimal bounding cube* that contains P. See the original paper for details.

Randomized point location. Point location is a classical problem in computational geometry [63]. In its full generality, the point location problem deals with locating query points in arbitrary subdivisions. In [65], Devroye et al. considered the problem of locating a query point in a two-dimensional Delaunay triangulation of n *random* points. They showed that a simple "walk-through" technique, namely random sampling followed by walking towards the query from its nearest neighbor in the sample, has expected running time close to $O(n^{1/3})$. Later, Mücke et al. [120] generalized this result to three dimension and gave a bound of roughly $O(n^{1/4})$ on the expected running time for locating a query point in a three-dimensional Delaunay triangulation of n *random* points. These algorithms do not assume any preprocessing on the Delaunay triangulations, and hence fit into our framework of sublinear geometric algorithms (see Section 1.1.2). However, these algorithms only work for Delaunay triangulations of *random* point sets. In Section 2.4.2, we give a new sublinear algorithm that works for *any* Delaunay triangulations. In Section 2.6.1, we further show that a variation of the walk-through technique actually works for *any triangulations*.

Dynamic geometric optimization. In [76], Eppstein gave sublinear algorithms for several geometric optimization problems in the dynamic setting. For example, he showed how to maintain the Euclidean bichromatic closest pair, bichromatic farthest pair, or diameter of any planar n-point set undergoing point insertions and deletions, in amortized time $O(n^{\varepsilon})$ per update for any $\varepsilon > 0$. Another result is that for any planar n-point set undergoing insertions and deletions, its EMST can be maintained in amortized time $O(n^{1/2} \log^2 n)$ per update. All these results can be extended to any fixed dimension, but with larger amortized update time (depending on the dimension).

1.1.2 New Results in this Thesis

In this thesis, we give sublinear algorithms for several fundamental geometric problems in two and three dimensions when no preprocessing is allowed. Most of these results (Chapters 2 and 3) were first published by Chazelle, Liu and Magen [48].

In our algorithms, the input is taken to be in any standard representation with no extra assumptions. For example, a planar subdivision or a polyhedron in 3D is given in classical edge-based fashion (e.g., DCEL, winged-edge), with *no extra preprocessing*. This implies that we can pick an edge at random in constant time, but we cannot sample randomly among the neighbors of a given vertex in constant time. Our motivation is two-fold: (i) we seek the minimal set of computational assumptions under which sublinearity is achievable; (ii) the assumptions should be realistic and nonrestrictive. Note, for example, that sublinear separation algorithms for convex objects are known [44, 69], but all of them require preprocessing, so they fall outside our model. Under these conditions one might ask whether there exist any interesting "offline" problems that can be solved in sublinear time. The answer is yes. Note that randomization is a necessity because, in a deterministic setting, most problems in computational geometry require looking at the entire input. As far as we know, the two papers on randomized point location [65, 120] (see Section 1.1.1) are the only previous works on sublinear geometric algorithms that fall inside of our model.

We divide our results into two groups: Las Vegas type algorithms[5] and approximation algorithms.

[5]A Las Vegas algorithm is a randomized algorithm that always produces correct results, with the only variation from one run to another being its running time.

Las Vegas Type Algorithms

Here is a summary of our Las Vegas type sublinear geometric algorithms. In all cases, n denotes the input size and all polyhedra are understood to be in $\mathbf{R}^3$:

- An optimal $O(\sqrt{n}\,)$ time algorithm for checking whether two convex polyhedra intersect. The algorithm reports an intersection point if they do and a separating plane if they don't.

- Optimal $O(\sqrt{n}\,)$ time algorithms for point location in two-dimensional Delaunay triangulations and Voronoi diagrams, and ray shooting towards a convex polytope.

- An $O(\sqrt{n}\,)$ time algorithm for finding the nearest neighbor of a given point in a convex polyhedron, and similar algorithms for related problems.

- An optimal $O(\sqrt{n}\,)$ time algorithm for point location in any triangulations in the plane. The algorithm generalizes to planar subdivisions with constant face size as well as three-dimensional triangulations. Additionally, there is a $\Omega(n)$ lower bound for point location in planar subdivisions with large faces.

In contrast with geometric property testing (see Section 1.1.1), our algorithms never err and randomization only affects the running time but not the correctness of the output.[6] Devroye et al. [65] showed that a simple walk-through technique for locating a query point in a two-dimensional Delaunay triangulation of n *random* data points has expected running time roughly $O(n^{1/3})$. This does not contradict the optimality of our $O(n^{1/2})$ bound because the data points must be chosen randomly in [65].

[6]Throughout this thesis, the running times of our sublinear geometric algorithms are understood in the expected sense. They are expected over the random bits used by the algorithms, and not over any input distribution.

Approximation Algorithms

We also consider geometric problems for which approximate solutions are sought. We give:

- An $O(\varepsilon^{-1}\sqrt{n}\,)$ time algorithm for approximating the volume of a convex polytope to within a relative error of $1+\varepsilon$ for any $\varepsilon > 0$.

- An $O(\varepsilon^{-5/4}\sqrt{n}\,) + f(\varepsilon^{-5/4})$ time algorithm for approximating the length of the shortest path between two points on the boundary of a convex polyhedron to within a relative error of $1+\varepsilon$ for any $\varepsilon > 0$. Here, $f(n)$ denotes the complexity of computing the *exact* shortest path between two points on a convex polyhedron of size n. This implies that the running time of our algorithm is $O(\sqrt{n}\,)$, for any fixed $\varepsilon > 0$.

Our approximate shortest path algorithm improves on the $(1+\varepsilon)$-approximation algorithm of Agarwal et al. [5], which runs in time $O(n\,\log\varepsilon^{-1}+\varepsilon^{-3})$, for any $\varepsilon > 0$. Our method also makes progress on an important geometric problem of independent interest.

- Given a convex polytope P of n vertices, how many vertices must an enclosing polytope Q have if it is to approximate any (large enough) shortest path on ∂P with relative error at most $1+\varepsilon$? We reduce to $O(\varepsilon^{-5/4})$ the best previous bound of $O(\varepsilon^{-3/2})$, due to Agarwal et al. [5].

1.2 Geometric Lower Bounds

Most of the research in computational geometry is devoted to algorithms and data structures, that is, finding the fastest solutions to geometric problems. The second part of this thesis, however, is concerned with lower bounds on the complexity of geometric problems. Studying lower bounds for geometric problems is also important. A high lower bound for a problem in a very general model of computation indicates that an efficient solution for this problem is impossible. In this situation, people usually turn to approximation algorithms that produce near-optimal solutions; or randomized algorithms that produce correct answers only with high probability; or more efficient algorithms for interesting special cases. Unfortunately, proving lower bounds in a very general model of computation (such as the Turing machine) could be very hard. Facing this difficulty, people turn to more specialized models of computation. These models are still general enough to encompass all known algorithms for the problems considered. Lower bounds in these specialized models are useful because they reveal the inherent limitations of current approaches, and, by demonstrating why current approaches fail, direct algorithm designers to consider new techniques. In Section 1.2.1, we review geometric lower bounds in several specialized models of computation. In Section 1.2.2 we describe our new results.

1.2.1 Geometric Lower Bounds: An Overview

Algebraic Decision Trees

Decision trees are one of the most widely studied models of computation. A k-ary decision tree is a directed tree in which every node has k children. In most cases, k is either 2 or 3. Associated with each internal node v is a query about the input that has k possible answers. Each answer is associated with one child of v. Each leaf of the tree is labelled with an output value. Given an input instance, we start at the root and traverse the tree in a top-down fashion. At each internal node, the query result on the input instance tells us which child we should proceed to. The running time is measured as the number of queries asked. In the worst case, it is the depth of the tree.

An algebraic decision tree is a ternary decision tree in which every query asks for the sign of a multivariate polynomial. Specifically, the input is assumed to be a vector $(x_1, \ldots, x_n)$ of real numbers, and each internal node queries the sign of a polynomial $f(x_1, \ldots, x_n)$. If all these polynomials are of degree at most d then it is called a degree-d algebraic decision tree. In practice, d is usually a constant. An important special case is when $d = 1$: such a tree is called a linear decision tree. A common technique for proving lower bounds in algebraic decision trees is via information-theoretic argument. For example, sorting n numbers in any comparison tree[7] requires worst case time $\log(n!) = \Omega(n \log n)$ simply because there are $n!$ different orderings of n numbers.

Yao [135] proved that any quadratic decision tree (i.e., degree-2 algebraic decision tree) for computing the convex hull of n points in the plane must use $\Omega(n \log n)$ time. This technique was later generalized to higher-order algebraic decision trees by Ben-Or [26], who also applied it to many other problems. Ben-Or also derived similar lower bounds in a stronger algebraic computation tree model. A current limitation of these information-theoretic arguments is that they cannot provide $\omega(n \log n)$ lower bounds. In fact, no $\omega(n \log n)$ lower bounds are known in the algebraic decision tree model for any natural problem solvable in polynomial time.[8] More recently, new lower bounds are proved for non-geometric problems in algebraic decision trees. These lower bounds are based on novel use of topological invariants [28] or topological complexity measures such as Betti numbers [138] and Euler characteristics [139]. However, these new tools are still inherently information-theoretic and therefore unlikely to beat the $n \log n$ bound.

[7]This is a special kind of decision tree in which every query is a comparison between two input values.

[8]Quadratic lower bounds are known for some NP-complete problems in the algebraic decision tree model.

The Semigroup Arithmetic Model

The semigroup arithmetic model was introduced to study the complexity of range searching, a classical problem in computational geometry [2, 3, 108]. Here, we need to preprocess a set P of n points in $\mathbf{R}^d$ so that, given a range R chosen from a predetermined class (e.g., all d-dimensional boxes, simplices, or halfspaces), the points of $P \cap R$ can be counted or reported quickly. In the semigroup arithmetic model, each point is given a value from a commutative semigroup, and the answer to a query range is the semigroup sum of the values associated with the points falling in that range. Note that this form of the problem is very general. For example, if we let the semigroup be $(\mathbf{Z}, +)$ and assign value 1 to each point, then we obtain range counting – count the number of points in the query range. If we let the semigroup be $(2^P, \cup)$ and assign to each point the singleton set consisting of itself, then we obtain range reporting – report all points in the query range. In the semigroup arithmetic model, the data structure is a set of precomputed partial sums in the underlying semigroup. A query is answered by adding a subset of partial sums. The size of the data structure is the number of partial sums stored. The time required to answer a query is the minimum number of partial sums that add up to the correct answer. All other computational costs (branchs, pointer traversals, selecting the right partial sums to add, etc.) are ignored from the query time. For this reason, lower bounds proved in the semigroup arithmetic model apply to all conceivable algorithms for range searching. As its name suggests, the semigroup arithmetic model disallows subtraction of semigroup values even if the underlying semigroup is actually a group (more on this later).

The semigroup arithmetic model was originally introduced by Fredman, who proved time lower bounds for dynamic orthogonal (i.e., queries are axis-parallel boxes) and half-plane range searching [85, 86]. Yao first applied this model to static range searching and proved lower bounds for orthogonal range searching in the plane [137]. For orthogonal range searching in higher dimensions, Vaidya proved nontrivial but sub-optimal lower bounds [132]. Chazelle proved tight (or almost tight) space-time tradeoff lower bounds for orthogonal and simplex (i.e., queries are d-dimensional simplices) range searching in any dimension [40, 38]. Halfspace range searching is a special case of simplex range searching: here only a sub-optimal space-time tradeoff lower bound is known [30].

A core technique in several of the above lower bounds is considering some extremal property of a bipartite graph between a set of points and a set of ranges. In such a graph, there is an edge between a point p and a range R if and only if $p \in R$. In the proofs of these lower bounds, a key step is to construct a set of points and ranges such that the corresponding bipartite graph has many edges but no large complete bipartite subgraphs. The *Discrepancy Method* plays an important role in constructing those "hard" points and ranges [42].

So far we only discussed on-line range searching, that is, queries must be answered on-line and preprocessing is done before the algorithm starts. There is also the off-line version, where all queries are given ahead of time and they are to be processed in batched mode. Chazelle proved nearly optimal lower bounds for off-line orthogonal and simplex range searching [42]. Here the inputs are n points p_1, …, p_n with semigroup values x_1, …, x_n and n ranges R_1, …, R_n, and we want to compute $\sum_{p_i \in R_j} x_i$ for each j. In other words, given $x = (x_1, \ldots, x_n) \in \mathbf{R}^n$, we want to compute Ax where A is a n by n 0/1 matrix such that $A_{ij} = 1$ if and only if $x_i \in R_j$. The model is further refined as a linear arithmetic circuit (or, equivalently, a straight-line program) where each circuit gate (or program step) performs an addition of two input values or previously computed variables. Similar to the case of on-line range searching, lower bounds for off-line range searching are also based on special properties of the incidence matrix A for a set of carefully constructed points and ranges. Specifically, in a "hard" matrix many of its entries are 1, but it does not contain a large submatrix of 1s.

A major limitation of the semigroup arithmetic model is that it disallows subtraction of semigroup values (even though the underlying semigroup is actually a group). Proving lower bounds in the group model (when subtraction is allowed) appears difficult and is still largely open. Chazelle proved nontrivial lower bounds for off-line halfplane and orthogonal range searching in the group model by examining the spectra of point-range incidence matrices [42]. As far as we know, however, this is the only work on range searching lower bounds in the group model. It was widely believed that allowing subtraction should not help much in range searching. Recently, Chazelle's work [43] casts some doubts on this belief by exhibiting the first nontrivial separation between the semigroup arithmetic and group arithmetic complexity of a natural range searching problem. This result implies that $\Theta(n \log n)$ might be the right answer for most (nonorthogonal) off-line range searching problems in the group model.

The Pointer Machine Model

The pointer machine model was originally developed by Tarjan to study the complexity of maintaining disjoint sets [131]. Chazelle used it to study the complexity of range reporting problems, a special case of range searching.[9] A data structure in the pointer machine model is a directed graph in which every node has constant outdegree. Each node in the graph is either unlabeled or labeled with the index of one point. Given a query, the algorithm starts from a distinguished node (called the source) and travers the graph in some arbitrary order. The requirement (for a valid algorithm) is that it must see the index of every point in the query range at least once during its traversal. Note that a

[9]In range reporting, we want to know which points are in the query range, not just how many of them.

new node must be visited by traversing an edge from a previously visited node. In other words, the only way to access a node is through a series of pointers. This is different from the RAM model where any memory cell can be accessed in constant time. The algorithm is also allowed to modify the data structure: for example, it can add a new node (unlabeled) whose outgoing edges point to previously visited nodes; it can also add or delete edges between previously visited nodes. Thus this model accommodates both static and self-adjusting data structures. The size of the data structure is the number of nodes in the graph, and the query time for a range R is the minimum possible number of nodes visited by a valid algorithm for R. The query time ignores all other computational costs such as the time to decide which edges to traverse. The pointer machine model is general enough to describe all known algorithms for range reporting, and lower bounds proved in this model have wide applicability.

Chazelle proved optimal lower bounds for orthogonal range reporting in the pointer machine model [39]. His techniques were later applied to simplex range reporting by Chazelle and Rosenberg, who proved a quasi-optimal lower bound [49]. In Chapter 4 of this thesis, we use these techniques to prove lower bounds for several intersection searching problems.

Other More Restricted Models

There are several very successful stories on proving geometric lower bounds in the models described in previous sections. For example, orthogonal and simplex range searching are essentially solved problems as of today (at least in theory) due to their (almost) tight lower bounds in the semigroup arithmetic model [42]. On the other hand, we cannot always derive satisfying lower bounds in these models either because we don't know how to do that or because they are inappropriate for some problems. For example, consider the following affine degeneracy testing problem: given n points in $\mathbf{R}^d$, decide if any $d+1$ of them lie on a hyperplane. This problem can be solved in time $O(n^d)$ [74, 75], but in the algebraic decision tree model, the only available lower bound is a somewhat trivial bound of $\Omega(n \log n)$. The reason is that, as mentioned in Section 1.2.1, we don't know how to prove a $\omega(n \log n)$ lower bound in the algebraic decision tree model for any natural problem solvable in polynomial time. As another example, Hopcroft's problem asks, for a set of n points and n lines in the plane, if any point lies on any line. There are several solutions that solve this problem in roughly $O(n^{4/3})$ time, the best of which is due to Matoušek [107], but $\Omega(n \log n)$ is the only lower bound in the algebraic decision tree model. The challenge here is to deal with *computationally oriented* models instead of combinatorial ones. The semigroup arithmetic model is suitable for studying range searching problems, but inappropriate for range emptiness problems where we want to know if at least one point lies inside the query range. If the query range is empty then no semigroup addition is performed; on the other hand, even a single addition is enough to indicate that the range is not empty. So there is really no lower bound for range emptiness in the semigroup arithmetic model.

Our discussion above implies that other (more restricted) models are needed to prove better lower bounds in some cases. In the following, we address a couple of such models.

Restricted decision trees. We mentioned earlier that the best lower bound for affine degeneracy testing in the general algebraic decision tree model is $\Omega(n \log n)$. On the other hand, Erickson and Seidel obtained an optimal $\Omega(n^d)$ lower bound for this problem in a restricted version of algebraic decision trees [80]. In their model, only the following type of query (called a sidedness query) is allowed: given $d+1$ points $p_0, \ldots, p_d$, decide the incidence relationship (above, on, or below) between p_0 and the oriented hyperplane defined by $p_1, \ldots, p_d$. This query essentially asks for the sign of the determinant of a $(d+1)$ by $(d+1)$ matrix formed by the coordinates of the given points. In this model, every decision is based on the result of a sidedness query. Other algebraic tests such as comparing the determinants of two sidedness queries are disallowed. Since all known algorithms for affine degeneracy testing use (or can be made to use) only sidedness queries, the lower bound in [80] implies that we cannot improve the existing algorithms unless other computational primitives are used. Erickson also proved near-optimal lower bounds for convex hull problems in the same model, and optimal lower bounds for spherical degeneracy testing in a similar model [78].

The famous 3SUM problem in computational geometry asks if a given set of n numbers has three elements whose sum is zero. While this problem can be solved in time $O(n^2)$, the best known lower bound is only $\Omega(n \log n)$ in the general algebraic decision tree model. The 3SUM problem is important because there is a multitude of problems in computational geometry (called 3SUM-hard problems) that can be reduced from it via subquadratic reductions [88]. In other words, a $\Omega(n^2)$ lower bound for 3SUM would imply a $\Omega(n^2)$ lower bound for all these problems. The class of 3SUM-hard problems includes detecting collinear points in the plane, separating line segments by a line, sorting the vertices of a line arrangement, computing the Minkowski sum of two polygons, checking for polygon containment under translation, testing if a union of triangles is simply connected, minimizing the Hausdorff distance between segment sets, moving a line segment from one position and orientation to another in the presence of polygonal obstacles, and many more. Since many of these problems can indeed be solved in $O(n^2)$ time, a $\Omega(n^2)$ lower bound for 3SUM would settle down their complexity completely.

The r-variate linear satisfiability problem (also called r-variate linear degeneracy testing) is a generalization of the 3SUM problem. Here, given n numbers and an integer

$r \leq n$, we ask if any r of these numbers satisfy a fixed linear equation $\sum_{i=1}^{r} a_i x_i = b$, with $a_i \neq 0$ for all i. A r-variate linear satisfiability problem can be solved in $O(n^{\lceil r/2 \rceil})$ time when r is odd, or $O(n^{\lceil r/2 \rceil} \log n)$ time when r is even. On the other hand, no lower bound better than $\Omega(n \log n)$ is known for this problem in the general algebraic decision tree model. Improving on previous work [84, 67], Erickson [79] proved a lower bound of $\Omega(n^{\lceil r/2 \rceil})$ for any r-variate linear satisfiability problem in the so-called r-linear decision tree model. In this model, each decision is based on the sign of a linear combination of at most r input numbers. Note that Erickson's lower bound is tight when r is odd and almost tight when r is even. An obvious limitation of Erickson's lower bound is that every linear test performed by the decision tree involves at most r numbers. Recently, Ailon and Chazelle [7] generalized Erickson's technique and established nontrivial lower bounds for r-variate linear degeneracy testing in any s-linear decision tree, for s up to $2r$. They also improved on Erickson's lower bound in r-linear decision trees from pseudopolynomial to exponential when r is large enough.

The partition graph model. We already know that the semigroup arithmetic model is inappropriate for studying range emptiness queries. To handle this problem, Erickson [78] introduced the partition graph model to study the complexity of geometric divide-and-conquer algorithms for range emptiness queries and other problems. The justification is that almost all geometric range searching (in particular, range emptiness) data structures are constructed by subdividing the space into several regions (with some desired properties) and building a data structure for each region recursively. Range queries are answered by performing depth-first search in the resulting space partition.

Formally, a partition graph is a directed acyclic graph with constant outdegree. The graph contains a distinguished node called the source, and several leaves that have no outgoing edges. Each internal node is associated with a constant number of connected subsets of $\mathbf{R}^d$ called query regions that together cover the whole space[10], and each query region is associated with an outgoing edge of that internal node. Each internal node is also labeled as either “primal” or “dual”, indicating whether its query regions reside in the primal or dual space. The actual algorithm that uses a partition graph data structure depends on the specific problem at hand (e.g., Hopcroft’s problem, halfspace range searching). Let’s take for example the hyperplane range emptiness problem: this is the problem of preprocessing a set of points so that given any query hyperplane we can quickly decide if any point lies on this hyperplane. In preprocessing, each point is fed into the data structure one at a time. We preform a depth-first search on the graph starting from the source. At each primal node, we traverse the outgoing edges corresponding to query regions that contain the point; at each dual node, we traverse the outgoing edges corresponding to query regions that intersect the point’s dual hyperplane. For each leaf of the partition graph, we maintain a set of points that reach it during preprocessing. To answer a hyperplane query, we perform a depth-first search similar to those used in preprocessing. That is, at primal nodes we traverse to query regions that intersect the hyperplane, and at dual nodes we traverse to query regions that contain the hyperplane’s dual point. The answer is determined by checking the point sets associated with all leaves reached by the query hyperplane. For example, if all such point sets are empty then the query hyperplane does not contain any point. The size of a partition graph is its number of edges. The query time for a hyperplane is the number of edges traversed during the depth-first search; other costs (such as constructing the partition graph) are ignored.

[10]These query regions are not required to be disjoint, convex, simply connected, or of constant combinatorial complexity.

In the partition graph model, Erickson [78] proved: (1) a lower bound for Hopcroft's problem that nearly matches the best known upper bound; (2) lower bounds for offline halfspace emptiness problems in dimensions five and higher.

1.2.2 New Results in this Thesis

Intersection Searching and Fractional Cascading

In Chapter 4, we prove lower bounds for a class of intersection searching problems in two and three dimensions in the pointer machine model. Several of our lower bounds are nearly optimal. These lower bounds have interesting corollaries that resolve a couple of long-standing open problems in computational geometry.

Here is a brief summary of our results (first published by Chazelle and Liu [47]). See Chapter 4 for the meaning and background of each problem, as well as the implications of our results. All lower bounds hold in the pointer machine model (Section 1.2.1).

1. There is a graph with 2D catalogs attached to its nodes such that to perform the same point location query at the k nodes of a connected subgraph in time $O(k+ \text{polylog}(n))$ requires storage $\tilde{\Omega}(n^2)$, where n is the combined size of all the catalogs.[11] This means that the two-dimensional generalization of fractional cascading is impossible.

2. For any n large enough, there is a convex planar subdivision with n vertices such that to compute all k edges intersected by a query line in $O(k+ \text{polylog}(n))$ time requires storage $\tilde{\Omega}(n^2)$. On the other hand, $O(n^2)$ storage is sufficient to achieve $O(k + \log n)$ query time (even if the query is a line segment). The same lower bound holds for intersecting a query line with a simple polygon with n vertices.

[11]The $\tilde{O}$ notation hides a polylogarithmic factor.

3. For any n large enough, there is a (nonconvex) polytope P in $\mathbf{R}^3$ with n vertices such that, given a query plane π, to compute its intersection $P \cap \pi$ in time $O(k+$ polylog$(n))$, where k is the number of vertices in $P \cap \pi$, requires storage $\Omega(n^{3-\varepsilon})$, where ε is an arbitrarily small positive constant. On the other hand, $O(n^3)$ storage is sufficient to achieve $O(k + \log n)$ query time.

4. For any n large enough, there exists a convex polytope with n vertices that admits of no boundary dominant Dobkin-Kirkpatrick hierarchy. This resolves a question that had been open since the mid-eighties.

Remarks: In all cases, the lower bounds remain true, up to a factor of n^{ε}, if we replace the polylog term in the query time by a function of the form n^{δ}, for small enough $\delta, \varepsilon > 0$. Our third result also holds for convex polytopes but the lower bound then drops to $\tilde{\Omega}(n^2)$. (We have been unable to find a matching upper bound for that restricted version of the problem.)

How do our results compare to previous work? The upper bounds are mostly applications of known techniques; in particular, duality, filtering search, navigation in arrangements. The lower bounds, on the other hand, should come as more of a surprise. Although not particularly difficult technically, they are rather counter-intuitive. Pointer machine lower bounds exist for all sorts of problems, including union-find [111, 131] and range searching [39, 49]; see surveys [3, 108]. However, we are not aware of previous lower bounds for intersection searching or 2D fractional cascading. Our proofs rely on a general volume argument developed in [49] and a Heilbronn-type result proven in [38]. The rest is new and self-contained.

Approximate Nearest Neighbor Searching

In Chapter 5, we prove a new lower bound for Approximate Nearest Neighbor Searching (ANNS), a basic problem in computational geometry with a variety of applications. This result was published by the author [106]. To be precise, we prove a lower bound of $d^{1-o(1)}$ on the query time for any deterministic algorithms that solve ANNS in Yao's cell probe model [136]. Our result holds for ANNS with a very loose approximation factor and in the special case of a Hamming Cube. (Note that this makes our lower bound even stronger.) Our result greatly improves on the previously best known lower bound for this problem (in the same model and with the same assumptions), which is $\Omega(\frac{\log \log d}{\log \log \log d})$ [33]. Moreover, our proof is much simpler than the proof in [33]. Chapter 5 presents all the details.

Part I

Sublinear Geometric Algorithms

Chapter 2

Las Vegas Type Algorithms

2.1 A Warm-up: Successor Searching

As a warm-up exercise, consider the classical *successor searching* problem: Given a sorted (doubly-linked) list of n keys and a number x, find the smallest key $y \geq x$ (the *successor* of x) in the list or report that none exists. Although we could not find a reference, the following algorithm is probably folklore. Choose $\sqrt{n}$ list elements at random, and find the predecessor and successor of x among those (perhaps only one exists). This provides an entry point into the list, from which a naive search takes us to the successor. To make random sampling possible, we may assume that the list elements are stored in consecutive locations (say, in a table). However—and this is the key point—no assumption is made on the ordering of the elements in the table (otherwise we could do a binary search).

Lemma 2.1.1 *Successor searching can be done in $O(\sqrt{n}\,)$ expected time per query, which is optimal.*

Proof: For $i \geq 1$, let Q_i be the set of all elements that are at distance at most i away from the answer on the list (in either direction). Let $P_{>i}$ be the probability of not hitting Q_i after $\sqrt{n}$ random choices of the list elements, with $P_{>0} = 1$. The expected distance of the answer to its nearest neighbor in the random sample is $\sum_{i \geq 1} i(P_{>i-1} - P_{>i}) = \sum_{i \geq 0} P_{>i}$. The first $\sqrt{n}$ terms sum up to $O(\sqrt{n})$. The rest of the sum is upper bounded by $\sqrt{n} \sum_{c \geq 1} P_{>c\sqrt{n}} \leq \sqrt{n} \sum_{c \geq 1} (1 - c/\sqrt{n})^{\sqrt{n}} = \sqrt{n} \sum_{c \geq 1} 2^{-\Omega(c)} = O(\sqrt{n})$. This immediately implies that the expected time of the algorithm is $O(\sqrt{n}\,)$.

For the lower bound, we use Yao's minimax principle [134]. We fix a distribution on the input, and we lower-bound the expected complexity of any deterministic algorithm. The input is a linked list containing the numbers 1 through n in sorted order. In our model, the list is represented by a table $T[1 \cdots n]$, with the i-th element in the list stored in location $\sigma(i)$ of the table; hence, $T[\sigma(i)] = i$. The input distribution is formed by choosing the permutation σ uniformly from the symmetric group on n elements. The query is set to be n. In other words, the problem is to locate the last element in the list. A deterministic algorithm can be modeled as a sequence of steps of the form: (A) pick a location $T[k]$ already visited and look up the next (or previous) item, i.e., $T[\sigma(i \pm 1)]$, where $k = \sigma(i)$; (B) compute a new index k and look up $T[k]$. Each step may involve the consideration of every piece of information gathered so far. In particular, in a B-step we may not consult either one of the adjacent items in the list before computing k (unless, of course, these items were visited earlier). In this way, $\sigma^{-1}(k)$ is equally likely to lie anywhere in the portion of the list still unvisited. For this reason, after a A-steps and b B-steps, there is a probability at least $\left(1 - \frac{\sqrt{n}+a+b}{n}\right)^b$ that none of the last $\sqrt{n}$ elements in the list has been visited in a B-step. Right after the last B-step, either the total number of A- and B-steps exceeds $\sqrt{n}$ or, with constant nonzero probability, at least $\sqrt{n}$ A-steps (some of which may have already been taken) are required to reach the last element in

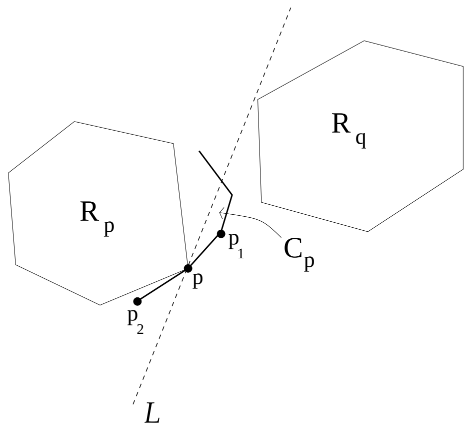

Figure 2.1: Intersecting two convex polygons

the list. This immediately implies that the expected time of any deterministic algorithm is $\Omega(\sqrt{n})$. $\square$

2.2 Intersection Detection for Convex Polygons

We can generalize the successor searching algorithm to polygon intersection. Given two convex polygons P and Q in the plane, with n vertices each, determine whether they intersect or not and, if they do, report one point in the intersection. We assume that P and Q are given by their doubly-linked lists of vertices (or edges) such that each vertex points to its predecessor and successor in clockwise order. As in successor searching, we assume that the two lists are stored in two tables to allow random sampling.

Choose a random sample of r vertices from each polygon, and let $R_p \subseteq P$ and $R_q \subseteq Q$ denote the two corresponding convex hulls. By two dimensional linear programming, we can test R_p and R_q for intersection without computing them explicitly. This can be done probabilistically (or even deterministically) in linear time. There are many ways of doing that (see [42] for references). It is easy to modify the algorithm (of, say, [126]) so that in $O(r)$ time it reports a point in the intersection of R_p and R_q if there is one, and a bi-tangent separating line $\mathcal{L}$ otherwise (Figure 2.1). Let p be the vertex of R_p in $\mathcal{L}$, and let p_1, p_2 be their two adjacent vertices in P. If neither of them is on the R_q side of $\mathcal{L}$, then define C_p to be the empty polygon. Otherwise, by convexity exactly one of them is; say, p_1. We walk along the boundary of P starting at p_1, away from p, until we cross $\mathcal{L}$ again. This portion of the boundary, clipped by the line $\mathcal{L}$, forms a convex polygon, also denoted by C_p. A similar construction for Q leads to C_q.

It is immediate that $P \cap Q \neq \emptyset$ if and only if P intersects C_q or Q intersects C_p. We check the first condition and, if it fails, check the second one. We restrict our explanation to the case of $P \cap C_q$. First, we check whether R_p and C_q intersect, again using a linear time algorithm for LP, and return with an intersection point if they do. Otherwise, we find a line $\mathcal{L}'$ that separates R_p and C_q and, using the same procedure as described above, we compute the part of P, denoted C'_p, on the C_q side of $\mathcal{L}'$. Finally, we test C'_p and C_q for intersection in time linear on their sizes, using LP or any other straightforward linear-time algorithms for intersection detection of convex polygons. Correctness is immediate. The running time is $O(r+|C_p|+|C'_p|+|C_q|+|C'_q|)$. It follows from a standard union bound that $\mathbf{E}\,|C_p| = O(n/r)\log n$, but a more carefully analysis shows that in fact $\mathbf{E}\,|C_p| = O(n/r)$. (The three-dimensional case discussed in the next section will subsume this result, so there is no need for a proof now.) Similarly, $\mathbf{E}\,|C'_p| = \mathbf{E}\,|C_q| = \mathbf{E}\,|C'_q| = O(n/r)$. The overall complexity of the algorithm is $O(r + n/r)$, and choosing $r = \lfloor \sqrt{n}\, \rfloor$ gives the desired bound of $O(\sqrt{n}\,)$. To show optimality, consider the following distributions on pairs of polygons. One polygon is fixed and degenerate: all its vertices lie in the origin. The other polygon (also degenerate) has $n-1$ vertices on the positive y-axis, and one vertex, p, in the origin or in $(0, \delta)$, where δ is a positive number small enough so that p is the lowest vertex of the polygon. Moreover, the edges of this polygon are randomly orderd in the edge table. Clearly, these two polygons interesct if and only if p is in the origin. Since nothing in the structure of the input except the geometry of p reveals whether it is indeed the origin, any algorithm that detects intersection must have access to p. Now recall that the only operations allowed are the random sampling of edges and edge-traversing via links, which means that, as in Lemma 2.1.1, an expected $\Omega(\sqrt{n})$ time is needed to access p. Optimality of subsequent results follow these lines very closely and shall not be proved again. We have:

Theorem 2.2.1 *To check whether two convex n-gons intersect can be done in $O(\sqrt{n}\,)$ time, which is optimal.*

To put Theorem 2.2.1 in perspective, recall that the intersection of two convex polygons can be determined in logarithmic time if the vertices are stored in an array in cyclic order [44]. The key point of our result is that, in fact, a linked list is sufficient for sublinearity. Similarly, if polyhedra are preprocessed à la Dobkin-Kirkpatrick then fast intersection detection is possible [69]. What we show in the next section is that sublinearity is achievable even with no preprocessing at all. Again, we use a two-stage process: In the first stage we break up the problem into r subproblems of size roughly n/r, and then identify which ones actually need to be solved; in the second stage we solve these subproblems in standard (i.e., non-sublinear) fashion. Their number is constant; hence the square root complexity. What prevents us from solving these subproblems recursively is the model's restriction to *global* random sampling. In other words, one can sample efficiently for the main problem but *not* for the subproblems.

2.3 Intersection Detection for Convex Polyhedra

2.3.1 The Algorithm

Given two n-vertex convex polyhedra P and Q in $\mathbf{R}^3$, determine whether they intersect or not. If they intersect then we report one point in the intersection; otherwise we report a plane that separates them. We assume that a convex polyhedron is given in any classical edge-based fashion (e.g., DCEL, winged-edge), but with no extra preprocessing. The main structure is a table of edges that allows us to pick an edge at random in constant time. There are also two tables for vertices and faces. Moreover, these tables are interconnected via pointers to make various local operations possible. For example, each edge points to its two vertices and two adjacent faces. It also points to its predecessor and successor edges in its two adjacent faces. Such a structure is a standard representation for convex polyhedra in computational geometry. It allows us to traverse a portion of a convex polyhedron in a local fashion and in time linear in the number of edges visited.

Choose a random sample of $r = \lfloor \sqrt{n} \rfloor$ edges from P and Q, and let R_p and R_q denote the convex hulls of these random edges in P and Q, respectively. We do not compute R_p and R_q explicitly, but merely use their vertices to get an LP as described in the last section for the case of polygons. We use this LP to detect intersection of R_p and R_q in $O(r)$ time, by invoking a linear-time algorithm for low-dimensional linear programming. We stop with a point of intersection if there is one. Otherwise, we find a separating plane $\mathcal{L}$ that is tangent to both R_p and R_q. It is important to choose the plane $\mathcal{L}$ in a canonical fashion. To do that, we set up the linear program so as to maximize, say, the coefficient a in the equation[1] $ax + by + cz = 1$ of $\mathcal{L}$.

[1]With perturbation techniques, we can always assume general position, and hence avoid having a solution passing through the origin. We will also assume that the relative position of P and Q is general.

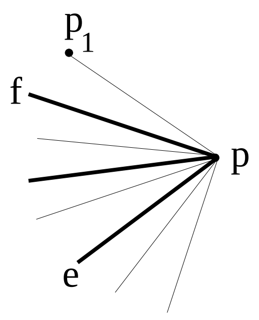

Figure 2.2: The edges of P incident to p; the thick lines form the random sample.

Next, choose a plane π normal to $\mathcal{L}$ and consider projecting P and Q onto it. (Of course, we do not actually do it.) Let p be a vertex of R_p in $\mathcal{L}$ (there could be two of them, but not more if we assume general position between P and Q) and let p^* be its projection onto π. Let p_1^*, p_2^*, . . ., p_k^* be the set of vertices adjacent to p^* in the projection of P onto π. We test to see if any of them is on the R_q side of $\mathcal{L}$, and identify one such point, p_1, if the answer is yes (more on that below). If none of them is on the R_q side, then we define C_p to be the empty polyhedron. This is because in this case, P is completely on the other side of $\mathcal{L}$. Otherwise, we construct the portion of P, denoted C_p, that lies on the R_q side of $\mathcal{L}$. Note that C_p is a convex polytope, not just the boundary of P cut off by $\mathcal{L}$. We compute C_p by using a standard flooding mechanism. Beginning at p_1, we perform a depth-first search through the facial structure of P, restricted to the relevant side of $\mathcal{L}$. Because C_p is convex, the edges form a single connected component, so we never need to leave C_p. This allows us to build the entire facial representation of C_p in time proportional to its number of edges. From then on, the algorithm has the same structure as its polygonal counterpart, i.e., we compute C_p, C_p', C_q, C_q' and perform the same sequence of tests.

The question is now: how do we find p_1 if it exists? To simplify the analysis, once we have p, we resample by picking r edges in P at random; let E be the subset of those incident to p. To find p_1, we project on π all of the edges of E. If there exists an edge of E that is on the R_q side of $\mathcal{L}$, then we identify its endpoint as p_1. Otherwise all the edges of E lie on one side of $\mathcal{L}$. We then identify the two extreme ones (e and f in Figure 2.2); being extreme means that all the other projected edges of E lie in the wedge between e and f in π. Assume that e and f are well defined and distinct. Consider the cyclic list V of edges of P incident to p. The edges of E break up V into blocks of consecutive edges. It is not hard to prove that pp_1 lies in blocks starting or ending with e or f, if such a p_1 (as defined above) exists. So, we examine each of these relevant blocks (at most four) exhaustively. If e and f are not both distinct and well defined, we may simply search for p_1 by checking every edge of P incident to p. This completes the description of the algorithm. In the next section we give the analysis.

2.3.2 The Analysis

Theorem 2.3.1 *Two convex n-vertex polyhedra in $\mathbf{R}^3$ can be tested for intersection in $O(\sqrt{n}\,)$ time; this is optimal.*

Proof: Optimality was already discussed in the polygonal case and correctness follows from elementary convex geometry, so we limit our discussion to the complexity of the algorithm. Because of the resampling, the expected sizes of the blocks next to e and f (or alternatively the expected size of the neighborhood of p if the blocks are not distinct) are $O(n/r)$, so the running time is $O(r + n/r + \mathbf{E}\,|C_p|)$, where $|C_p|$ denotes the number of edges of C_p. We may exclude the other terms $|C_p'|$, $|C_q|$, and $|C_q'|$, since our upper bound on $\mathbf{E}\,|C_p|$ will apply to them as well. The naive bound of $O((n/r)\log n)$ on $\mathbf{E}\,|C_p|$ can be improved to $O(n/r)$. Here is how.

We modify the sampling distribution a little. Then we argue that reverting back to the original setting does not change the asymptotic value of the upper bound. The modification is two-fold: (i) we view P as a multiset M where each vertex appears as many times as its number of incident edges; (ii) R_p is formed by picking each point of M independently with probability r/n. With respect to the modified distribution, $|C_p|$ is proportional to the number of constraints in M that violate the linear program $\mathcal{P}(R_p, R_q)$ used to define $\mathcal{L}$ (with each point of R_p and R_q defining a linear constraint). Consider a subset $M' \subseteq M$ such that the solution of $\mathcal{P}(M', R_q)$ (if it exists) has exactly k violations in M. We distinguish between the solutions with one point in M and two in R_q and those with two points in M and one in R_q. (Assuming general position between P and Q, these are the only possibilities.) Let f_k and g_k count the number of solutions of the first and second type respectively, and let $f_{\leq k} = f_0 + \cdots + f_k$ (with the same definition for g). For example, we have $f_0 + g_0 = 1$ and $f_{|M|} = g_{|M|} = 0$. We can prove by standard arguments [54, 121] that

$$f_{\leq k} = O(k) \qquad \text{and} \qquad g_{\leq k} = O(k^2). \tag{2.1}$$

To see why, form a random sample S by picking each point of M independently with probability s/n. Obviously, the number of solutions of $\mathcal{P}(S, R_q)$ is one; therefore, so is its expected value. This gives us

$$\sum_{j \leq |M|} f_j \Big(\frac{s}{n}\Big)\Big(1 - \frac{s}{n}\Big)^j + \sum_{j \leq |M|} g_j \Big(\frac{s}{n}\Big)^2 \Big(1 - \frac{s}{n}\Big)^j = 1.$$

Choosing $s = n/k$ we have,

$$\sum_{j\le k} f_j\Big(\frac{s}{n}\Big)\Big(1-\frac{1}{k}\Big)^j + \sum_{j\le k} g_j\Big(\frac{s}{n}\Big)^2\Big(1-\frac{1}{k}\Big)^j \le 1.$$

it follows easily that

$$\Big(\frac{s}{n}\Big)f_{\le k} + \Big(\frac{s}{n}\Big)^2 g_{\le k} = O(1),$$

which proves (2.1).

Returning to our modified distribution, which assigns probability r/n to each point of M, we now have

$$\mathbf{E}\,|C_p| = O(1)\sum_{k\le |M|}\Big\{kf_k\Big(\frac{r}{n}\Big)\Big(1-\frac{r}{n}\Big)^k + kg_k\Big(\frac{r}{n}\Big)^2\Big(1-\frac{r}{n}\Big)^k\Big\}$$

The first $k_0 = O(n/r)$ summands add up to

$$\Big(\frac{r}{n}\Big)k_0 f_{\le k_0} + \Big(\frac{r}{n}\Big)^2 k_0 g_{\le k_0} = O\Big(\frac{n}{r}\Big).$$

Setting $u_k = k\Big(1-\frac{r}{n}\Big)^k$, we can use summation by parts to bound the contribution of the last $|M| - k_0$ summands. This gives an upper bound of

$$\Big(\frac{r}{n}\Big)\sum_{k_0\le k<|M|}\Big\{(u_k - u_{k+1})f_{\le k}\Big\} + \Big(\frac{r}{n}\Big)u_{|M|}f_{\le |M|}$$

$$+\Big(\frac{r}{n}\Big)^2\sum_{k_0\le k<|M|}\Big\{(u_k - u_{k+1})g_{\le k}\Big\} + \Big(\frac{r}{n}\Big)^2 u_{|M|}g_{\le |M|}.$$

By (2.1) and the inequality

$$u_k - u_{k+1} \le \frac{rk}{n}\,e^{-kr/n},$$

this is also bounded by

$$O\Big(\frac{r}{n}\Big)^2 \sum_{k_0 \le k < |M|} k^2 e^{-kr/n} + O\Big(\frac{r}{n}\Big)^3 \sum_{k_0 \le k < |M|} k^3 e^{-kr/n} + O(1)$$

$$= O\Big(\frac{n}{r}\Big).$$

This implies that

$$\mathbf{E}\,|C_p| = O\Big(\frac{n}{r}\Big). \tag{2.2}$$

Let $\mathcal{D}$ be the original distribution (the one used by the actual algorithm) with r replaced by $7r$. Of course, by (2.2) this scaling has no asymptotic effect on the upper bound for $\mathbf{E}\,|C_p|$. We define an intermediate distribution $\mathcal{D}_1$ by going through each edge (u, v) of P twice, selecting it with probability r/n, and then throwing into the sample both u and v, provided that the edge (u, v) has not been selected yet. (Note that this implies that u and v are kept out with probability $(1 - r/n)^2$.) There are at most $3n$ edges in P, so the probability that a sample from $\mathcal{D}_1$ is of size less than $7r$ is overwhelmingly high. Since all equal-size subsets of edges are equally likely to be chosen, $\mathbf{E}_{\mathcal{D}}\,|C_p|$ is nonincreasing with the sample size, and so, $\mathbf{E}_{\mathcal{D}}\,|C_p| = O(\mathbf{E}_{\mathcal{D}_1}\,|C_p|)$. Let $\mathcal{D}_2$ denote the modified distribution used in the calculations. Observe that $\mathcal{D}_2$ is derived from $\mathcal{D}_1$ by picking only u if (u, v) is chosen the first time it is considered for selection, and then only v if it is picked the second time around. By monotonicity, we then have $\mathbf{E}_{\mathcal{D}_1}\,|C_p| = O(\mathbf{E}_{\mathcal{D}_2}\,|C_p|)$. This proves that (2.2) holds in the distribution used by the algorithm.

Recall that the running time is $O(r + n/r + \mathbf{E}\,|C_p|)$ which is $O(r + n/r)$ by the above analysis. For $r = \lfloor \sqrt{n} \rfloor$ it is $O(\sqrt{n})$. $\square$

When the two convex polyhedra intersect the algorithm reports a point in the intersection. On the other hand when they are disjoint, we can report a plane that separates them. Here is a brief description on how to do that. Note that we cannot simply return a separating plane for C_p and C'_q (or C'_p and C_q) because it is not necessarily separating for P and Q. Instead, we resort to geometric duality to compute the desired plane in expected $O(\sqrt{n})$ time. In a standard geometric duality transform, a vertex in the primal space is mapped to a plane in the dual space and vice versa. Moreover, the upper (resp. lower) hull of a convex polyhedron is transformed to a lower (resp. upper) envelope [63]. When P and Q are disjoint, at least one of the following must be true: (1) there exists a plane above the upper hull of P and below the lower hull of Q; (2) there exists a plane below the lower hull of P and above the upper hull of Q. Since they are symmetric, it suffices to consider the first one. By duality, such a plane dualizes to a point in the common intersection of an upper and a lower envelope, which is a convex polyhedron. Although this polyhedron is not available explicitly, we have access to its geometric features (vertices, edges, etc.) in constant time via the corresponding features in the primal space. Hence we can apply the previous algorithm to find an intersection point in $O(\sqrt{n})$ time, which is the dual of a separating plane for P and Q.

2.4 Ray Shooting Applications

2.4.1 Ray Shooting towards a Convex Polytope

Given a convex polyhedron P with n vertices and a directed line ℓ in $\mathbf{R}^3$, the ray shooting problem asks for the point on (the boundary of) P hit by ℓ if it exists. We apply essentially the same techniques as in convex polyhedral intersection to ray shooting and solve it in expected $O(\sqrt{n})$ time. Choose a random sample of $\lfloor \sqrt{n} \rfloor$ edges from P and let R_p denote the convex hull of these edges. We first use LP to detect intersection of R_p and ℓ in time $O(\sqrt{n})$. There are two cases. If R_p and ℓ do not intersect, we get a plane $\mathcal{L}$

that separates them and passes through a vertex q of R_p. Starting from q we construct the intersection C_p of P with the halfspace bounded by $\mathcal{L}$ that contains ℓ. We already explained how to do that in the previous section. Finally, we solve ray shooting for C_p and ℓ. Now suppose that R_p and ℓ intersect. We first find the point p on R_p hit by ℓ in time $O(\sqrt{n})$. We cannot afford to compute an explicit representation of R_p in time $\Omega(\sqrt{n}\log n)$. To find p we again use LP. We can assume that ℓ is the positive x-axis by rotating the coordinate system. Of course we do not rotate the whole polytope P. Instead, we maintain such a rotation transform implicitly. In other words, whenever we need a geometric feature (vertex, edge, etc.) of P after the rotation, we compute it from its corresponding feature on the original input in constant time. Finding p is equivalent to finding a plane $\mathcal{L}$ such that: (1) all vertices of R_p are on one side of $\mathcal{L}$ (the side that contains $(+\infty, 0, 0)$); (2) the intersection point of $\mathcal{L}$ with the x-axis has its x coordinate as large as possible. In fact, p is that intersection point. It is straightforward to formulate this problem as a three-dimensional LP and solve it in time $O(\sqrt{n})$. In particular, to ensure (2) above we minimize the coefficient a in the equation $ax + by + cz = 1$ for $\mathcal{L}$. Once we have $\mathcal{L}$ and p, we construct C_p as before and solve the problem for C_p and ℓ. An analysis very similar to the proof of Theorem 2.3.1 shows that the expected size of C_p is $O(\sqrt{n})$. We thus have:

Theorem 2.4.1 *Given a convex polyhedron with n vertices and a directed line, we can compute their intersection explicitly in optimal $O(\sqrt{n}\,)$ time.*

2.4.2 Point Location in 2D Delaunay Triangulations and Voronoi Diagrams

The above sublinear time algorithm for ray shooting towards a convex polyhedron gives us useful ammunition for all sorts of location problems.

Given the Delaunay triangulation $\mathcal{T}$ of a set S of n points in the plane and a query point q, consider the problem of locating q, i.e., retrieving the triangle of $\mathcal{T}$ that contains it. The Delaunay triangulation can be given in any classical edge-based data structure (e.g. DCEL) that supports $O(1)$ time access to a triangle from a neighboring triangle. We use the close relationship between Delaunay triangulations and convex hulls given by the mapping $h : (x, y) \mapsto (x, y, x^2 + y^2)$. As is well known [63], the Delaunay triangulation of S is facially isomorphic to the lower hull of $h(S)$ (i.e., the part of the convex hull that sees $z = -\infty$). In this way, point location in $\mathcal{T}$ is equivalent to ray shooting towards the convex hull, where the ray originates from the query point q and shoots in the positive z direction. Obviously, any facial feature of the convex hull can be retrieved in constant time from its corresponding feature in the Delaunay triangulation. (The one exception is the set of faces outside the lower hull: we can simplify matters by adding a dummy vertex to the hull at $z = \infty$.)

The same argument can be used for point location in Voronoi diagrams. Recall that each point (p_x, p_y) is now lifted to the plane $Z = 2p_xX + 2p_yY - (p_x^2 + p_y^2)$, which is tangent to the paraboloid $Z = X^2 + Y^2$. The Voronoi diagram of S is isomorphic to the lower envelope of the arrangement formed by the n tangent planes [63]. Note that any vertex (resp. edge) of the envelope can be derived in constant time from the three (resp. two) faces incident to the corresponding vertex (resp. edge).

Theorem 2.4.2 *Point location in the Delaunay triangulation or Voronoi diagram of n points in the plane can be done in optimal $O(\sqrt{n}\,)$ time.*

2.5 Nearest Neighbor Searching and Related Problems

We consider the following problem, which will arise in our discussion of volume approximation and shortest paths algorithms in Chapter 3. Given a convex polyhedron P with n vertices and a point q, let $n_P(q)$ denote the (unique) point of P that is closest to q. Of course, we can assume that q does not lie inside P, which we can test by using the previous algorithm. To compute $n_P(q)$ we extract a sample polyhedron R_p of size $\sqrt{n}$ (as we did before) and find $n_{R_p}(q)$. Since we just have a collection of vertices of R_p instead of its full facial representation, it is not obvious how to find $n_{R_p}(q)$ in time $O(\sqrt{n})$. For this purpose, we express this problem as a *LP-type* problem and solve it using the method in [42] (see Chapter 8). A reformulation of the problem would be to seek the plane $\mathcal{L}$ that separates q from the vertices of R_p and maximizes the distance from q to it. To apply the method in [42], we view each vertex of R_p as a constraint. We also check that all the assumptions (i.e. monotonicity, locality, violation test and range space oracle) needed to solve this problem efficiently hold. See [42] for details. Thus we get $\mathcal{L}$ in time $O(\sqrt{n})$: it is tangent to R_p at $n_{R_p}(q)$ and normal to the segment $qn_{R_p}(q)$. Next, we compute the intersection C_p of P with the halfspace bounded by $\mathcal{L}$ that contains q. Again, a similar analysis shows that the expected size of C_p is $O(\sqrt{n})$. Obviously, $n_P(q) = n_{C_p}(q)$, so we can finish the work by exhaustive search in C_p.

Theorem 2.5.1 *Given a convex polyhedron P with n vertices and a point q, the nearest neighbor of q in P can be found in $O(\sqrt{n}\,)$ time.*

We can compute a related function by similar means. Given a directed line ℓ, consider an orthogonal system of coordinates with ℓ as one of its axes (in the positive direction), and define $\xi_P(\ell)$ to be any point of P with maximum ℓ-coordinate. If we choose a point q at infinity on ℓ, then $\xi_P(\ell)$ can be chosen as $n_P(q)$, and so we can apply Theorem 2.5.1.

Another function we can compute in this fashion maps a plane $\mathcal{L}$ and a direction ℓ in $\mathcal{L}$ to the furthest point of P in $\mathcal{L}$ along ℓ: in other words, $\xi_P(\mathcal{L}, \ell) = \xi_{\mathcal{P}\cap\mathcal{L}}(\ell)$. Again, the nonobvious part is computing $\xi_{R_p}(\mathcal{L}, \ell)$ in time $O(\sqrt{n})$ for a sample polytope R_p. As in the case of ray shooting, we can assume without loss of generality that $\mathcal{L}$ is the xy plane and ℓ is the positive x direction. Finding $\xi_{R_p}(\mathcal{L}, \ell)$ is the same as finding a plane $\mathcal{L}'$ such that: (1) all vertices of R_p are on one side of $\mathcal{L}'$ (the side that contains $(-\infty, 0, 0)$); (2) $\mathcal{L}'$ is parallel to the y axis; (3) the intersection point of $\mathcal{L}'$ with the x-axis has its x coordinate as small as possible. We solve this problem in time $O(\sqrt{n})$ by formulating it as a three-dimensional LP. Other parts of the algorithm (e.g. constructing C_p) and its analysis are similar to other problems discussed in this section. We summarize our results.

Theorem 2.5.2 *Given a convex polyhedron P with n vertices, a directed line ℓ, and a plane π, the points $\xi_P(\ell)$ and $\xi_P(\pi, \ell)$ can be found in $O(\sqrt{n}\,)$ time.*

2.6 Point Location in Planar Subdivisions

Given a convex planar subdivision $\mathcal{S}$ with n edges and a query point q, the point location problem asks for the face of $\mathcal{S}$ that contains q [63]. The subdivision $\mathcal{S}$ can be given in any classical edge-based data structure such as DCEL.

In [65], Devroye et al. showed that a simple walk-through technique locates a query point in a Delaunay triangulation of n *random* points in expected (roughly) $O(n^{1/3})$ time. In Section 2.6.1 below, we show that a variation of the walk-through technique actually locates any query point in any planar triangulations in expected $O(\sqrt{n})$ time, which is optimal. Our result thus removes the two fold assumption on the input triangulation: being Delaunay and formed by random points. Our algorithm also generalizes to planar subdivisions with constant face size as well as three-dimensional triangulations. In Section 2.6.2, we show a $\Omega(n)$ lower bound for point location in planar subdivisions with large faces.

2.6.1 Point Location in Triangulations

First we need some definitions. Let $\mathcal{S}$ be a planar triangulation with n edges. For some edge e of $\mathcal{S}$, let p_e be the nearest neighbor of q on e. In other words, $|p_e q| \leq |pq|$ for any $p \in e$. The Euclidean distance between e and q is just $|p_e q|$. We also define a "crossing distance" between e and q: it is the number of edges of $\mathcal{S}$ intersected by the segment $p_e q$. Given a subset $\mathcal{S}' \subseteq \mathcal{S}$, we call an edge $e \in \mathcal{S}'$ the nearest edge of q if e has the smallest Euclidean distance from q among all edges of $\mathcal{S}'$.

Now consider the following algorithm. Sample $\sqrt{n}$ edges of $\mathcal{S}$ at random. Let e be the nearest edge of q in this random sample. We identify p_e, the nearest neighbor of q on e. We then walk from p_e towards q by traversing all triangles intersected by the segment $p_e q$. Note that this traversal is easy to implement given an edge-based data structure for $\mathcal{S}$ such as DCEL. Suppose we enter a triangle Δ from one of its neighboring triangles Γ, then in constant time we can decide which of the other two neighboring triangles of Δ we will enter next. Continuing in this way, we will finally reach the triangle that contains q. The time complexity of this traversal is proportional to the number of triangles crossed by $p_e q$, or the crossing distance between q and e.

To prove that this algorithm runs in expected time $O(\sqrt{n})$, it suffices to show that the expected crossing distance between q and e is $O(\sqrt{n})$. To do this, we rank each edge of $\mathcal{S}$

according to its Euclidean distance to q. In other words, we sort the n Euclidean distances (from the n edges to q) and the rank of e is the rank of its Euclidean distance to q in this sorted list. The crossing distance between q and e is at most the rank of e, because for every edge intersected by $p_e q$ that edge has smaller Euclidean distance to q than e. It is a simple fact that the expected minimum rank of $\sqrt{n}$ numbers chosen randomly from an n-element list is $O(\sqrt{n})$ (see also the proof of Lemma 2.1.1). Finally, our algorithm generalizes to point location in planar subdivisions with constant face size as well as point location in three-dimensional triangulations.

Theorem 2.6.1 *Point location in an n-edge planar subdivision with constant face size or an n-facet triangulation in $\mathbf{R}^3$ can be done in optimal $O(\sqrt{n}\,)$ time.*

Remark. Our algorithm is a variation of the walk-through technique used by several previous algorithms. However, in our algorithm it is crucial to walk from the nearest neighbor of q on its nearest edge. This makes our algorithm run in expected time $O(\sqrt{n})$ for *any* triangulations. In contrast, previous algorithms either walk from an endpoint (or the midpoint) of the nearest sample edge, or sample by vertices and walk from the nearest sample vertex. These algorithms cannot guarantee sublinear expected running time for arbitrary triangulations.

2.6.2 Point Location in Subdivisions with Large Faces

We have seen sublinear time point location in planar subdivisions with constant face size. What if the subdivision has large faces: Is sublinear time point location still possible? The answer is no.

Theorem 2.6.2 *There exists an n-edge planar subdivision such that any randomized algorithm for point location in this subdivision has expected running time $\Omega(n)$. The subdivision contains faces of size $\Omega(n)$.*

Proof: We first consider the following problem. We are given a doubly-linked list of numbers. We know that exactly one number is nonzero and would like to find out its sign (positive or negative). It is obvious that a deterministic algorithm has to check every number in the worst case, but what about randomized algorithms? Here is a lower bound argument showing that any randomized algorithm has $\Omega(n)$ expected running time.

We use Yao's minimax principle. We fix a distribution on the input list and lower-bound the expected complexity of any deterministic algorithm. The input distribution is formed by picking a list L uniformly at random from the following $2n$ lists: for $1 \leq i \leq n$, A_i represents the list in which the i-th number is 1 and all the others are 0; similarly, B_i represents the list in which the i-th number is -1 and all the others are 0. A deterministic algorithm can be modeled as a sequence of steps of the form: (A) compute a index k and look up list element $L[k]$; (B) pick an element $L[k]$ already visited and look up the next (or previous) element, i.e., $L[k+1]$ or $L[k-1]$. Since the algorithm must give the correct answer, it can only terminate after seeing the nonzero element. Moreover, since the algorithm is deterministic, its behavior is completely determined by its internal state and the list elements it visited so far. In other words, if we run the algorithm on a list of all zeros, then we obtain a *fixed* ordering among the (indices of) list elements. This is the order on which the elements are visited. Now we change an element to 1 (or -1) and run the algorithm on the new list. The algorithm will visit the elements according to the fixed ordering until it hits the nonzero element. Viewing this ordering as a permutation, the time taken by the algorithm on a list is the shortest prefix of this permutation that contains the nonzero element. This implies that the expected running time (over the input distribution) is at least $(\sum_{i=1}^{n} i)/n$, or $\Omega(n)$.

Now we return to the point location problem. Again we use Yao's minimax principle and pick a distribution of planar subdivisions as follows. First, we identify a rectangle R in the plane whose corners are: $(-1, n+1)$, $(-1, 0)$, $(1, n+1)$, $(1, 0)$. We then break each vertical side of R into $n+1$ unit length segments. For example, there is a segment that connects $(-1, j)$ to $(-1, j+1)$ for every $0 \leq j \leq n$. In othe words, R is now a face with $2n+4$ edges. Finally, We obtain a two-face subdivision $\mathcal{S}_i$ ($1 \leq i \leq n$) by adding to R the segment that connects $(-1, i)$ to $(1, i)$. The input distribution is the uniform distribution among all $\mathcal{S}_i$, and the query point is $(0, (n+1)/2)$, the center of R. Given an (unknown) subdivision $\mathcal{S}_i$, any deterministic algorithm must find the "middle" edge (from $(-1, i)$ to $(1, i)$) to locate the query point. Since it is hopeless to hit that edge with good probability by sampling $o(n)$ edges, the only way to discover it is through its four adjacent edges: from $(-1, i-1)$ to $(-1, i)$; from $(-1, i)$ to $(-1, i+1)$; from $(1, i-1)$ to $(1, i)$; from $(1, i)$ to $(1, i+1)$. Now we view the vertical edges of $\mathcal{S}_i$ as a list of numbers, and the four adjacent edges as nonzero elements. We then essentially need to solve the problem discussed at the beginning of this proof.[2] We thus get the desired lower bound of $\Omega(n)$. $\square$

2.7 Conclusion and Future Work

We have presented sublinear algorithms for several fundamental geometric problems in two and three dimensions when no preprocessing is assumed. The problems we con-

[2]The point location problem is slightly different. For example, there are two lists of vertical edges instead of one; there are four special edges; etc., but these are minor issues.

sider include intersection detection of convex polygons and polyhedra, ray shooting towards convex polytopes, point location in two-dimensional triangulations and Voronoi diagrams, as well as nearest neighbor type problems. Our randomized algorithms read only a small fraction of the input and this makes them appealing to large geometric data sets. In contrast with geometric property testing, our algorithms never err and randomization only affects the running time but not the correctness of the output. In most cases, our algorithms achieve optimal expected running time.

There are many interesting open problems concerning sublinear geometric algorithms. Here are a few of them:

- The scope of geometric problems for which sublinear algorithms exist is not well understood. It is therefore important to identify the class of geometric problems that can be solved in sublinear time.

- A classical result of Gary Miller states that every 2-connected triangulated planar graph with n vertices has a simple cycle separator of size $O(\sqrt{n})$ [113]. Given the importance of planar separators in divide-and-conquer algorithms, a sublinear time algorithm for finding small separators in a planar graph would be very interesting.

- Our algorithm for polyhedral intersection relies heavily on the convexity of input polytopes. On the other hand, it is not hard to show that sublinear time intersection detection for arbitrary non-convex polytopes is impossible. But what happens if the input polytopes are just slightly non-convex? (In practice, convex polytopes may come as being slightly non-convex due to various causes such as roundoff errors, bursty noise, and aliasing.) To state the problem more precisely, we say that a polytope P is ε-close to being convex if the Hausdorff distance[3] between P and its convex hull is at most $\varepsilon\,\mathtt{diam}\,(P)$, where $\mathtt{diam}\,(P)$ denotes the diameter of P. Intuitively, given two polytopes that are ε-close to being convex for small ε, it should be possible to adapt our algorithm to detect their intersection in sublinear time. Now the question is: how does the running time depend on n and ε?

[3]For two point sets A, B in $\mathbf{R}^3$, their Hausdorff distance is defined as:

$$H(A, B) = \max\{\max_{a\in A}\ \min_{b\in B}\ |ab|,\ \ \max_{a\in B}\ \min_{b\in A}\ |ab|\}$$

.

Chapter 3

Approximation Algorithms

3.1 Dudley's Construction

Given a n-vertex convex polytope P in $\mathbf{R}^3$, a classical result of Dudley [72] states that there exists a low complexity polytope Q that approximates P fairly well. Specifically, for any $\varepsilon > 0$, there exists a convex polytope Q with $O(1/\varepsilon)$ vertices such that $P \subseteq Q$ and the Hausdorff distance[1] between P and Q is at most ε. This result plays an important role in our algorithms to be presented in this chapter. In the following we review the method of constructing such a polytope Q.

We may assume that the diameter of P is 1. Let S be a sphere of radius 2 centered at some arbitrary point in P. Draw a grid of longitudes and latitudes on S, so that each grid cell is of length $\sqrt{\varepsilon}$ by $\sqrt{\varepsilon}$. In other words, the length of a side of a cell (a circular arc) is $\sqrt{\varepsilon}$. Let V be the set of grid points, we have $|V| = O(1/\varepsilon)$. For each point $v \in V$, we

[1]See the footnote in Section 2.7 for the defi nition of Hausdorff distance.

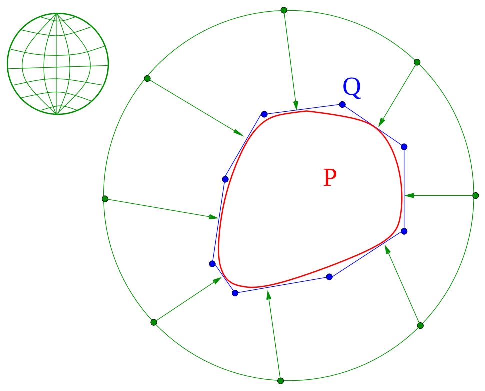

Figure 3.1: Dudley's construction.

compute $n_P(v)$, its nearest neighbor in ∂P (see Section 2.5), and define

$$Q = \bigcap \{ H^+_{n_P(v)} \,|\, v \in V \}. \tag{3.1}$$

Here $H_{n_P(v)}$ is the plane tangent to P at $n_P(v)$, and $H^+_{n_P(v)}$ is the halfspace bounded by $H_{n_P(v)}$ that contains P. See Figure 3.1 for illustration. It is immediate that $P \subseteq Q$ and Q has $O(1/\varepsilon)$ vertices.

One property of V is that for every point $s \in S$, there exists a grid point $v \in V$ such that the Euclidean distance between v and s is at most $\sqrt{\varepsilon}$. Based on this property, Dudley proved that the Hausdorff distance between P and Q is $O(\varepsilon)$ [72].

3.2 Volume Approximation

We seek to approximate the volume of a convex polytope P. We proceed in two stages. First, we compute a large enough enclosed ellipsoid, which we use to rescale P affinely. This is intended to make P *round* enough so that good Hausdorff distance approximation yields good volume approximation. Second, we use Dudley's construction to find, via the methods of Section 2.5, an enclosing polytope of $O(1/\varepsilon)$ vertices whose boundary is at Hausdorff distance at most ε from P.

3.2.1 A Constant Factor Approximation

We begin by computing, in $O(\sqrt{n})$ time, a polytope $P' \subseteq P$, such that $\mathrm{vol}\,(P') \geq c_0\,\mathrm{vol}\,(P)$ for some constant $c_0 > 0$. Compute the six points $\xi_P(\ell)$, for $\ell = \pm x$, $\pm y$, $\pm z$. (See Section 2.5 for the meaning of $\xi_P(\ell)$ and $\xi_P(\pi, \ell)$ below.) These points come in pairs, so let w_1, w_2 be the pair forming the largest distance. Given a point w on the line $\mathcal{L}$ passing through w_1 and w_2, let P_w denote the intersection of P with the plane through w that is orthogonal to $\mathcal{L}$. Let w_0 be the midpoint of w_1w_2 (Figure 3.2). We first show that if S is a set of points in P_{w_0} such that

$$\texttt{area}\,(\texttt{conv}\,(S)) \geq c_1\,\texttt{area}\,(P_{w_0}), \tag{3.2}$$

for some constant $c_1 > 0$, then $\mathrm{vol}\,(\texttt{conv}\,(S \bigcup \{w_1, w_2\})) \geq c_2 \mathrm{vol}\,(P)$, for some other constant $c_2 > 0$. Therefore, we can take $P' = \texttt{conv}\,(S \bigcup \{w_1, w_2\})$ to achieve our goal. Indeed, assume we have such a set S. As a straightforward consequence of Pythagoras' theorem, we find that $\texttt{diam}\,(P) \leq \sqrt{3}\,d(w_1, w_2)$; therefore, the orthogonal projection of P on $\mathcal{L}$ is a segment $v_1v_2 \supseteq w_1w_2$ of length at most $\sqrt{3}\,d(w_1, w_2)$. This implies that, for any w in $\mathcal{L}$, $\texttt{area}\,(P_w) \leq 12\,\texttt{area}\,(P_{w_0})$. To see why, observe that if, say, $w \in v_1w_0$, then, by convexity, P_w is enclosed in the cone with apex w_2 and base P_{w_0}. Therefore, P_w lies in a copy of P_{w_0} scaled by at most $d(w, w_2)/d(w_0, w_2) \leq 2\sqrt{3}$, which proves our claimed upper bound on $\texttt{area}\,(P_w)$. Of course, the same argument can be repeated if

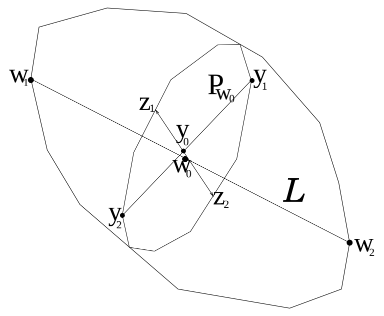

Figure 3.2: Approximating P from within.

$w \in w_0v_2$. Since vol$\,(P) = \int_{v_1}^{v_2} \texttt{area}\,(P_w)\,dw$, we can conclude that the 4 quantities

$$\text{vol}\,(P), \qquad \text{vol}\,(\texttt{conv}\,(P_{w_0} \cup \{v_1, v_2\})),$$

$$\text{vol}\,(\texttt{conv}\,(P_{w_0} \cup \{w_1, w_2\})), \quad \text{vol}\,(\texttt{conv}\,(S \cup \{w_1, w_2\}))$$

are all equal up to within constant factors.

We now show how to find a set S satisfying (3.2). We essentially repeat in 2D what we did so far in 3D. Specifically, we take a, b to be two mutually orthogonal vectors both normal to $\mathcal{L}$, and let π be the plane spanned by a and b. We compute the four points (two pairs) $\xi_P(\pi, \ell)$, for $\ell = a, -a, b, -b$. Let y_1, y_2 be the more distant pair (analogous to w_1, w_2 before). Let y_0 be the midpoint of y_1, y_2 and let segment ℓ_{y_0} be the intersection of P with the line in π orthogonal to y_1y_2. We can find the two endpoints z_1, z_2 of ℓ_{y_0} using ray shooting. Using almost the same argument as the one showing that $\texttt{conv}\,(P_{w_0} \cup \{w_1, w_2\})$ has a volume proportional to vol$\,(P)$, we get that the quadrilateral with vertex set $S = \{y_1, y_2, z_1, z_2\}$ has an area proportional to $\texttt{area}\,(P_{w_0})$, and thus satisfies (3.2). We comment that a similar approach to the one we described above was used by Barequet and Har-Peled in [20]. The difference is that they approximate the volume of a convex polytope from outside by a bounding box, whereas we approximate it from within.

Let $\mathcal{E}$ be the largest ellipsoid enclosed in $P' = \mathsf{conv}\,(\{y_1, y_2, z_1, z_2, w_1, w_2\})$, also known as the Löwner-John ellipsoid. It is computable in constant time within any fixed relative error by solving a constant-size quadratic program [93]. As is well known, its volume is at least $(1/\mathrm{dim})^2$ times that of the enclosing polytope; therefore,

$$\mathrm{vol}\,(\mathcal{E}) \geq \frac{1}{9}\,\mathrm{vol}\,(P') \geq c \cdot \mathrm{vol}\,(P),$$

for some constant $c > 0$. Make the center of the ellipsoid the origin of the system of coordinates, and use the ellipsoid's positive semidefinite matrix to rescale P. To do that, we consider the linear transformation that takes the ellipsoid into a ball of the same volume. Specifically, if $x^T A^T A x \leq 1$ is the equation of the ellipsoid, then we consider the transformation $T = A/(\det A)$. The polytope TP has the same volume as P, but it is *round*, namely it contains a ball B of volume $\Omega(\mathrm{vol}\,(TP))$. Thus, we might as well assume that P has this property to begin with. Note that P is also enclosed in a concentric ball B' that differs from B by only a constant-factor scaling. (If not then TP would contain a point p so far away from B that the convex hull of p and B, although contained in P, would have volume much larger than $\mathrm{vol}\,(B)$ and hence $\mathrm{vol}\,(P)$, which would give a contradiction.) Finally, by rescaling we can also assume that P is enclosed in the unit ball and its volume is bounded below by a positive constant. By Theorems 2.4.1 and 2.5.2, all of the work in this section can be done in $O(\sqrt{n}\,)$ time.

3.2.2 Sublinear Time $(1+\varepsilon)$-Approximation

We implement Dudley's construction of a convex polytope Q such that: (i) $Q \supseteq P$; (ii) $Q \subset P_\varepsilon$, where P_ε is the Minkowski sum of P with a ball of radius ε; (iii) Q has $O(1/\varepsilon)$ vertices. Dudley's result was used constructively in [5]. The difference here is that our implementation is sublinear. We compute an $\sqrt{\varepsilon}$-net on the unit sphere,[2] and project this net down to ∂P, using the nearest-neighbor function n_P as a projection map (Section 2.5). Finally, we form Q as the intersection of the $O(1/\varepsilon)$ halfspaces bounded by the appropriate tangent planes passing through the vertices of the projected net. With suitable use of the nearest neighbor algorithm of Theorem 2.5.1, we can implement the entire construction in time $O(\varepsilon^{-1}\sqrt{n}\,)$ for the projection construction (since the facial representations of P and TP are the same, the algorithm can use TP as though it had its full facial representation at its disposal) and $O(\varepsilon^{-1}\log\varepsilon^{-1})$ for intersecting the halfspaces needed to form Q. Since we can obviously assume that Q does not have more vertices than P, there is no need for ε to be smaller than, say, $1/n^2$. This implies that the entire construction time is in fact $O(\varepsilon^{-1}\sqrt{n}\,)$.

We now show that $\text{vol}\,(Q) = (1+O(\varepsilon))\text{vol}\,(P)$. Recall that P is "sandwiched" between two concentric balls B and B' such that $rad(B') = 1$ and $rad(B) = \Omega(1)$. We may assume that B and B' are centered at the origin. We claim that $Q \subset (1 + 2\varepsilon/rad(B))P$ which will immediately show that $\text{vol}\,(Q) = (1+O(\varepsilon))\text{vol}\,(P)$. To prove the claim, consider any point $q \in Q$ and let p be its nearest neighbor in P (so that $|pq| \leq 2\varepsilon$ by condition (ii) above), and p' be the intersection of ∂P and the segment Oq. Also, let r be the point on the line pp' so that Or is parallel to pq. Then by convexity of P, r

[2]This is a collection of $O(\varepsilon^{-1})$ points on the sphere such that any spherical cap of radius $\sqrt{\varepsilon}$ contains at least one of the points.

is outside of P and hence $|Or| > rad(B)$. Then, $|qp'|/|Op'| = |pq|/|Or| < 2\varepsilon/rad(B)$. Since this holds for any q, we have $Q \subset (1 + 2\varepsilon/rad(B))P = (1 + O(\varepsilon))P$.

Theorem 3.2.1 *Given any $\varepsilon > 0$, it is possible to approximate the volume of an n-vertex convex polytope with arbitrary relative error $\varepsilon > 0$ in time $O(\varepsilon^{-1}\sqrt{n}\,)$.*

3.3 Approximate Shortest Paths on a Convex Polytope

3.3.1 The Shortest Path Problem

The shortest path problem for polyhedral surfaces has been extensively studied, drawing its motivation from applications in route planning, injection molding, computer assisted surgery [4, 94, 117]. An interesting special case is computing a shortest path between two points along the boundary of a n-vertex convex polyhedron. In this case, an $O(n^3 \log n)$ algorithm was given by Sharir and Schorr [128], later improved by Mitchell et al. [118] to $O(n^2 \log n)$ and by Chen and Han [51] to $O(n^2)$. More recently, Kapoor [102] has announced an (unverified) algorithm that works in time $O(n \log^2 n)$.

These exact shortest path algorithms are too complicated to be practical. This motivates research on faster and simpler algorithms for finding an approximate shortest path. Hershberger and Suri [97] presented a simple algorithm that computes a 2-approximate shortest path between two points on a n-vertex convex polytope in time $O(n)$. Agarwal et al. [5] developed an algorithm that, given any $\varepsilon > 0$, computes a $(1 + \varepsilon)$-approximate shortest path in time $O(n \, \log \varepsilon^{-1} + \varepsilon^{-3})$ for any two points that are not too close. Their algorithm is based on the aforementioned Dudley's approximation polytope. Specifically, given two points $s, t \in \partial P$, they first compute an approximating polytope Q for P such that Q has $O(\varepsilon^{-3/2})$ vertices and the Hausdorff distance between Q and P is at most $\varepsilon^{3/2}$.

They compute an exact shortest path between s and t along the boundary of Q and project it onto ∂P without increasing its length.[3] It is proved in [5] that this projected path is a $(1+\varepsilon)$-approximate shortest path.

3.3.2 Our New Results

Given a convex polyhedron P with n vertices and two points s and t on its boundary ∂P, find the shortest path between s and t outside the interior of P. It is well known that the shortest path lies on the boundary ∂P. In fact, it is easy to construct instances where any reasonable approximation of the shortest path on ∂P involves $\Omega(n)$ edges. This rules out sublinear algorithms, unless we are willing to follow paths outside of P. We show how to compute a path between s and t whose length exceeds the minimum by a factor of at most $1+\varepsilon$, for any $\varepsilon > 0$.

Our algorithm relies on a new result of independent interest. Let $d_P(s,t)$ denote the length of the shortest path between s and t in ∂P. Given a point $v \in \partial P$, let H_v be the supporting plane of P at v (or any such plane if v is a vertex), and let H_v^+ denote the halfspace bounded by H_v that contains P. Given $\varepsilon > 0$, we say that a convex polytope Q is an ε-*wrapper* of P if: (c_0 is an absolute constant discussed below.)

(i) Q encloses P;

(ii) the Hausdorff distance between ∂P and ∂Q is $O(\varepsilon\,\mathtt{diam}\,(P))$;

(iii) given any $s,t \in \partial P$ such that $d_P(s,t) \geq c_0\,\mathtt{diam}\,(P)$, $d_{\widehat{Q}}(s,t) \leq (1+\varepsilon)d_P(s,t)$, where $\widehat{Q} = Q \cap H_s^+ \cap H_t^+$.

Lemma 3.3.1 *Any convex 3-polytope has an ε-wrapper of size $O(1/\varepsilon^{5/4})$, for any $\varepsilon > 0$.*

[3]The construction of Q ensures that $s,t \in \partial Q$.

This result improves on the $O(1/\varepsilon^{3/2})$ bound of Agarwal et al. [5]. The use of a wrapper is self-evident. First, we clip the polytope to ensure that $d_P(s,t) \geq c_0\, \mathtt{diam}\,(P)$ (Section 3.3.3). Next, we compute an ε-wrapper (Section 3.3.4) and approximate the shortest path between s and t by computing the shortest path between the two points in $\partial\widehat{Q}$. This can be done in quadratic time by using an algorithm by Chen and Han [51]. The resulting path, which is of length $(1+O(\varepsilon))d_P(s,t)$, can be shortened to $(1+\varepsilon)d_P(s,t)$ by rescaling ε suitably. Note that in (iii) the condition on s and t being sufficiently far apart is essential. It is a simple exercise to show that no variant of a wrapper can accommodate all pairs (s,t) simultaneously. If $f(n)$ denotes the complexity of the *exact* version of the shortest path problem, then we have,

Theorem 3.3.2 *Given any $\varepsilon > 0$ and two points s,t on the boundary of a convex polytope P of n vertices, it is possible to find a path between s and t outside P of length at most $(1+\varepsilon)d_P(s,t)$ in time $O(\varepsilon^{-5/4}\sqrt{n}\,) + f(\varepsilon^{-5/4})$.*

As mentioned in Section 3.3.1, it is known for sure that $f(n) = O(n^2)$ [51]. On the other hand, Kapoor's (unverified) algorithm [102] implies that $f(n) = O(n \log^2 n)$, which would make our algorithm run in expected time $O(\varepsilon^{-5/4}\sqrt{n}\,)$. This improves on Agarwal et al.'s algorithm [5], which runs in $O(n\, \log \varepsilon^{-1} + \varepsilon^{-3})$ time , for any $\varepsilon > 0$.

3.3.3 Computing Short Paths

Given two points $s,t \in \partial P$, our first task is to ensure that $d_P(s,t) \geq c_0\, \mathit{diam}\,(P)$, for some constant $c_0 > 0$. To do this, we first compute a value δ such that $\delta \leq d_P(s,t) \leq 8\delta$. We will substitute for P the intersection P' of P with a box centered at s of side length 16δ. Obviously, the shortest path between s and t relative to P and P' are identical. The only computational primitive we need is the nearest neighbor function of Theorem 2.5.1.

It is clear that if we can compute the function relative to P, then we can do it with respect to P' with only constant-time overhead.

To compute a constant-factor approximation for $d_P(s,t)$, we adapt an algorithm of Har-Peled [94] to our sublinear setting. All that is needed is an implementation of the following primitive: Given two rays r_1, r_2 from a fixed point $p \in P$, let H be the plane spanned by these two rays and let C denote the two-dimensional cone in H wedged between r_1 and r_2. Given an additional query ray $r \in H$ (not necessarily emanating from p), we need to compute $\xi_{C \cap P}(H, r)$. By Theorem 2.5.2, this can be done in $O(\sqrt{n}\,)$ time.

3.3.4 Sublinear Time $(1+\varepsilon)$-Approximation

Assuming without loss of generality that $\mathtt{diam}\,(P) = 1$, it suffices to prove the following:

Theorem 3.3.3 *Given any $\varepsilon > 0$ and a convex polytope P of n vertices, there exists a convex polytope Q with $O(\varepsilon^{-5/4})$ vertices such that:* (i) *$Q \supseteq P$;* (ii) *the Hausdorff distance between ∂P and ∂Q is $O(\varepsilon)$; and* (iii) *given any $s, t \in \partial P$ such that $d_P(s,t) \geq c_0$ for some constant c_0, $d_{\widehat{Q}}(s,t) \leq (1 + O(\varepsilon)) d_P(s,t)$, where $\widehat{Q} = Q \cap H_s^+ \cap H_t^+$.*

We first show how to construct Q. Let S be a sphere of radius 2 centered at some arbitrary point in P. Draw a grid G of longitudes and latitudes on S, so that each cell is of length $\sqrt{\varepsilon}$ by $\sqrt{\varepsilon}$ (with an exception made for the last latitude and longitude, if $\sqrt{\varepsilon}$ does not divide π). All lengths in this discussion are Euclidean, *except* in this case where the length of a circular arc refers to its corresponding angle. We choose a parameter $\lambda = \varepsilon^{3/4}$ and subdivide each side of a cell into sub-arcs of length λ (Figure 3.3). In this way each cell has $O(\sqrt{\varepsilon}/\lambda)$ vertices, and the whole construction defines a set V of $O(1/\lambda\sqrt{\varepsilon}\,)$ vertices. For each point $v \in V$, we compute $n_P(v)$, its nearest neighbor in

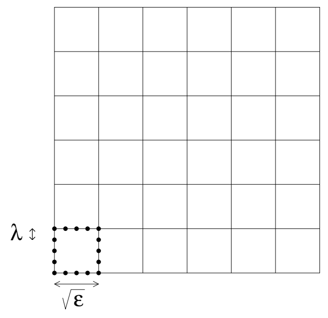

Figure 3.3: The grid G.

∂P, and define

$$Q = \bigcap \{ H^+_{n_P(v)} \mid v \in V \}. \tag{3.3}$$

It is immediate from our choice of λ that Q has $O(\varepsilon^{-5/4})$ vertices. Every point of the sphere S has at least one vertex of G at distance $O(\sqrt{\varepsilon}\,)$. By a result of Dudley [72], this implies part (ii) of Theorem 3.3.3. Since (i) is obvious, it remains for us to prove (iii).

Borrowing terminology from Agarwal et al. [5], we say that a pair $(\sigma, \mathcal{H})$ forms a *supported path* of P if $\sigma = p_1,q_1,p_2,q_2,\ldots, q_{m-1},p_m$ is a polygonal line disjoint from the interior of P and $\mathcal{H} = H_{p_1}, \ldots, H_{p_m}$ is a sequence of supporting planes of P, such that $q_{i-1}p_i$ and p_iq_i both lie in H_{p_i}, with $q_0 = p_1$ $q_m = p_m$ (Figure 3.4). For $0 < i < m$, the *folding angle* α_i at q_i is the dihedral angle of the wedge between H_{p_i} and $H_{p_{i+1}}$ (the one outside P). The folding angle of σ is defined as $\alpha(\sigma) = \sum_{0<i<m} \alpha_i$.

Lemma 3.3.4 (Agarwal et al. [5]) *Given $s, t \in \partial P$, there exists a supported path σ of P with $O(1/\varepsilon)$ edges, joining s and t, such that:*

$$d_P(s,t) \le |\sigma| \le (1+\varepsilon) d_P(s,t) \qquad \textit{and} \qquad \alpha(\sigma) = O(\varepsilon^{-1/2}).$$

To help build intuition for the remainder of our discussion, it is useful to sketch the proof of the lemma. Mapping the grid G to P via the nearest neighbor function n_P creates a grid $n_P(G)$ on ∂P (with curved, possibly degenerate edges). It is convenient to think of P as a smooth manifold by infinitesimally rounding the vertices and edges. It does not much matter how we do that as long as the end result endows each point $p \in \partial P$ with an (outward) unit normal vector η_p that is a continuous function of p. Note that in this way, for any $u \in S$, the vectors $un_P(u)$ and $\eta_{n_P(u)}$ are collinear, and the function n_P is a bijection. The fundamental property of the nearest-neighor function is that it is non-expansive. We need only a weak version of that fact, which follows directly from Lemmas 4.3 and 4.4 in [72].

Lemma 3.3.5 (Dudley [72]) *Given two points $p, q \in \partial P$, $|pq|$ and $\angle(\eta_p, \eta_q)$ are both in $O(\,|n_P^{-1}(p)n_P^{-1}(q)|\,)$.*

This implies that, for any two points $p, q \in \partial P$ in the same cell of the mapped grid $n_P(G)$, both $|pq|$ and $\angle(\eta_p, \eta_q)$ are in $O(\sqrt{\varepsilon}\,)$. We shortcut the shortest path on ∂P from s to t to form a supported path σ that passes through each cell at most once. In this manner, we identify $O(1/\varepsilon)$ points $p_1, \ldots, p_m$ on ∂P, where p_i (resp. p_{i+1}) is the entry (resp. exit) point of the path through the i-th cell in the sequence. The points p_i's lie on the edges of $n_P(G)$. There are two exceptions, $p_1 = s$ and $p_m = t$, which might lie in the interior of the cell. Next, we connect each pair (p_i, p_{i+1}) by taking the shortest path on $H_{p_i} \cup H_{p_{i+1}}$. The path intersects $H_{p_i} \cap H_{p_{i+1}}$ at a point denoted q_i. (Note that q_i might be infinitesimally close to p_i.) This forms a supported path σ with $O(1/\varepsilon)$ vertices $s = p_1, q_1, p_2, q_2, \ldots, q_{m-1}, p_m = t$. The only real difference with the proof in [5] is that we skip the final "trimming" step and keep the points p_i's unchanged. We mention two useful, immediate consequences of Lemma 3.3.5.

- The folding angle at q_i is $O(\sqrt{\varepsilon}\,)$.

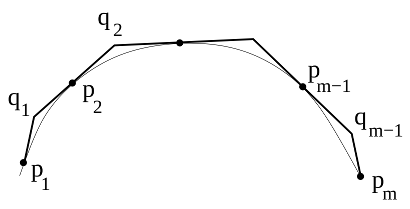

Figure 3.4: The path σ.

- For each $1 \leq i \leq m$, the point p_i belongs to ∂P and, for $i \neq 1, m$, there exists a point $w_i = n_P(v_i)$, where $v_i \in V$, such that both $|p_i w_i|$ and $\angle(\eta_{p_i}, \eta_{w_i})$ are in $O(\lambda)$.

From σ we build a curve σ' of length $(1 + O(\varepsilon))|\sigma|$ that joins s and t outside the interior of $\widehat{Q}$. The classical result below shows that the shortest path on $\partial\widehat{Q}$ from s to t cannot be longer than σ', which proves Theorem 3.3.3.

Theorem 3.3.6 (Pogorelov [124]) *Given a convex body C, let γ be a curve joining two points $s, t \in \partial C$ outside the interior of C. Then the length of γ is at least that of the shortest path joining s and t on ∂C.*

We now explain how to construct σ'. For $0 < i < m$, let (p_i, η_{p_i}) and (q_i, η_{p_i}) be the rays emanating from p_i and q_i, respectively, in the direction normal to H_{p_i} away from P. Together with the segments $p_i q_i$ and $q_i p_{i+1}$, the four rays (p_i, η_{p_i}), (q_i, η_{p_i}), $(q_i, \eta_{p_{i+1}})$, and $(p_{i+1}, \eta_{p_{i+1}})$ define a polyhedral surface Σ_i, which consists of two unbounded rectangles, Σ_i^1 and Σ_i^3, joined together at q_i by an unbounded triangle, Σ_i^2 (Figure 3.5). Note that the surface is in general nonplanar but Σ_i^2 is always normal to the line $H_{p_i} \cap H_{p_{i+1}}$. Out of Σ_i we carve a polyhedral strip S_i as follows. Fix a large enough constant $c > 0$, and let K_i denote the plane $H_{p_i} + c\lambda^2 \eta_{p_i}$. In other words, K_i is a parallel copy of H_{p_i} translated by $c\lambda^2$ away from P: As usual, the superscripted K_i^+ denotes the halfspace enclosing P.

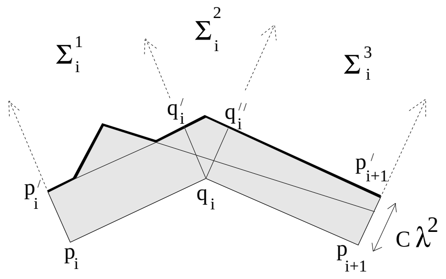

Figure 3.5: The curve σ_i'.

Recall that w_i is the nearest neighbor of v_i defined earlier. We need to consider

$$S_i = \Sigma_i \cap \Big\{ (K_i^+ \cap K_{i+1}^+) \cup (H_{w_i}^+ \cap H_{w_{i+1}}^+) \Big\}.$$

Again, we have two exceptions for $i = 1, m-1$, where we use $H_{p_1}^+$ instead of $H_{w_1}^+$ and $H_{p_m}^+$ instead of $H_{w_m}^+$.

Let $p_i p_i'$ be the edge of S_i incident to p_i collinear with η_{p_i}. We denote by σ_i' the portion of ∂S_i between p_i' and p_{i+1}', and define σ' as $\bigcup_{0<i<m} \sigma_i'$. To provide a connection to s and t, we also add to σ' the segments $p_1 p_1'$ and $p_m p_m'$. To show that σ' is a connected curve outside the interior of $\widehat{Q}$ of length $(1 + O(\varepsilon))|\sigma|$ requires a simple technical lemma.

Lemma 3.3.7 *Given an orthogonal system of reference (O, xyz), assume that P is tangent to the xy plane at O and lies below it. Given a point p on ∂P, if $|n_P^{-1}(O) n_P^{-1}(p)| < \delta$, for some small enough $\delta > 0$, then the intersection of H_p with the xz plane has for equation, $Z = aX + b$, where $|a| = O(\delta)$ and $0 \le b = O(\delta^2)$.*

Proof: By Lemma 3.3.5, the normal to H_p forms a small angle $\theta = O(\delta)$ with the z axis, so the plane H_p, being nonparallel to the z axis, can be expressed as $Z = aX + cY + b$. The cross product between the normal $(a, c, -1)$ and the z-axis vector is the vector $(c, -a, 0)$. By the cross product formula, its length, which is $\sqrt{a^2 + c^2}$, is also equal to

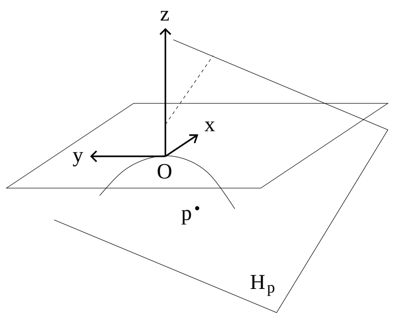

Figure 3.6: How H_p intersects the xz plane.

$\sqrt{a^2+c^2+1}\,\sin\theta$. It follows that $a^2+c^2 = O(a^2+c^2+1)\delta^2$; therefore,

$$a^2+c^2 = \frac{O(\delta^2)}{1-O(\delta^2)} = O(\delta^2), \tag{3.4}$$

and hence $|a| = O(\delta)$. By convexity of P, the plane H_p intersects the nonnegative part of the z axis, and p_z, the z coordinate of p, is nonpositive. By (3.4) and $|Op| = O(\delta)$, it follows that

$$0 \le b = p_z - ap_x - cp_y \le \sqrt{a^2+c^2}\sqrt{p_x^2+p_y^2} = O(\delta^2).$$

□

We examine each σ_i' separately, omitting the cases $i = 1, m-1$, which are trivial modifications of the general case $1 < i < m-1$. The curve σ_i' lies outside the interior of $H_{w_i}^+ \cap H_{w_{i+1}}^+$ and hence of $\widehat{Q}$. It is naturally broken up into three parts, $\sigma_i^j \subset \Sigma_i^j$ ($j = 1,2,3$), each one of them being a polygonal curve whose edges lie in any one of four planes: K_i, K_{i+1}, H_{w_i}, and $H_{w_{i+1}}$. Applying Lemma 3.3.7 with $(p_i, \overrightarrow{p_iq_i}, \overrightarrow{p_ip_i'})$ in the role of (O,x,z) and w_i in the role of p, we find that H_{w_i} intersects the segment p_ip_i', for c large enough; similarly, $H_{w_{i+1}}$ intersects $p_{i+1}p_{i+1}'$. This shows that p_i' is the intersection of the ray (p_i, η_{p_i}) with the plane K_i; therefore, p'_i is *the same point* in the definition of σ'_i and σ'_{i-1}, thus proving that the curve σ' is, indeed, connected. (The danger was having p'_i defined by $H_{w_{i+1}}$.) We now bound the length of σ'_i.

- By Lemma 3.3.7 the slopes of the edges of σ_i^1 are chosen among: 0 for K_i; $O(\sqrt{\varepsilon}\,)$ for K_{i+1}; $O(\lambda)$ for H_{w_i}; and $O(\sqrt{\varepsilon}\,)$ for $H_{w_{i+1}}$. It follows that $|\sigma_i^1| \le |p_i q_i| / \cos\theta$, where $\theta = O(\sqrt{\varepsilon}\,)$; therefore, $|\sigma_i^1| = (1 + O(\varepsilon))|p_i q_i|$. The same argument shows that $|\sigma_i^3| = (1 + O(\varepsilon))|q_i p_{i+1}|$.

- Let q'_i, q''_i be the endpoints of the curve σ_i^2 (Figure 3.5), and let a, a', b, b' be the distances along the ray (q_i, η_{p_i}) from q_i to K_i, K_{i+1}, H_{w_i}, and $H_{w_{i+1}}$, respectively. By definition of S_i,

  $$|q_i q'_i| = \max\Big\{ \min\{a, a'\}, \min\{b, b'\} \Big\}.$$

  Obviously, $a = c\lambda^2$ and, by Lemma 3.3.7, $b = O(\lambda|p_i q_i| + \lambda^2)$. This implies that $|q_i q'_i| = O(\lambda|p_i q_i| + \lambda^2)$ and, by the same argument,

  $$|q_i q'_i| + |q_i q''_i| = O(\lambda(|p_i q_i| + |q_i p_{i+1}|) + \lambda^2).$$

  Within Σ_i^2, the curve σ_i^2 is a polygonal line consisting of at most a constant number of edges. It is easy to see that for any vertex v of σ_i^2 (including q'_i and q''_i), the angle between $q_i v$ and edges of σ_i^2 incident to v is $\pi/2 \pm O(\sqrt{\varepsilon}\,)$. This follows from a simple geometric observation: given any plane H whose normal makes with $q_i v$ an angle at most α, the angle formed by $q_i v$ and any line on H lies in the range $[\pi/2 - \alpha, \pi/2 + \alpha]$. Since any of the edges of σ_i^2 lies on one of four planes: K_i, K_{i+1}, H_{w_i} and $H_{w_{i+1}}$, and the normal of each of them makes an angle of $O(\sqrt{\varepsilon})$

with $q_i v$, the claim follows. Because the folding angle of $O(\sqrt{\varepsilon}\,)$ can be assumed to be less than, say, $\pi/2$, this implies that the curve σ_i^2 lies entirely at a distance $O(|q_i q_i'| + |q_i q_i''|)$ from q_i. It follows that $|\sigma_i^2| = O(|q_i q_i'| + |q_i q_i''|)\sqrt{\varepsilon}$.

Putting everything together we find that

$$|\sigma_i'| = (1 + O(\varepsilon) + O(\lambda\sqrt{\varepsilon}\,))(|p_i q_i| + |q_i p_{i+1}|) + O(\lambda^2\sqrt{\varepsilon}\,).$$

In view of the fact that $|p_1 p_1'| = |p_m p_m'| = c\lambda^2$, summing up over all $|\sigma_i'|$'s (there are $O(1/\varepsilon)$ of them),

$$\begin{aligned}
|\sigma'| &= (1 + O(\varepsilon) + O(\lambda\sqrt{\varepsilon}\,))|\sigma| + O(\lambda^2/\sqrt{\varepsilon}\,) \\
&= (1 + O(\varepsilon))|\sigma| + O(\varepsilon) \\
&= (1 + O(\varepsilon))|\sigma|,
\end{aligned}$$

which completes the proof of Theorem 3.3.3. Note that the setting of λ is made to ensure that the additive term $O(\lambda^2/\sqrt{\varepsilon}\,)$ is $O(\varepsilon)$. $\square$

3.4 Open Problems

We have seen that an ε-wrapper of a convex polytope P can be used to approximate the shortest path between any two (not too close) points on ∂P. Our result provides an improved upper bound of $O(\varepsilon^{-5/4})$ on the size of an ε-wrapper. On the other hand, Dudley's result [72] implies that the lower bound is $\Omega(\varepsilon^{-1})$. An obvious open problem is: what is the true bound? We conjecture that it is $O(\varepsilon^{-1})$.

Another open problem is whether we can extend the $(1 + \varepsilon)$-approximate shortest path algorithms in this thesis (or in previous work) to nonconvex polyhedra or polyhedral terrain surfaces. Varadarajan and Agarwal [133] gave a subquadratic algorithm that computes a $O(1)$-approximate shortest path between two points on the surface of a nonconvex polyhedron. It is open whether we can compute a $(1 + \varepsilon)$-approximate shortest path in near-linear time for nonconvex polyhedra or polyhedral surfaces.

Part II

Geometric Lower Bounds

Chapter 4

Lower Bounds for Intersection Searching and Fractional Cascading

4.1 Intersection Searching

Intersection searching refers to the following type of problem: Given a set S of geometric "objects" (e.g., edges of a planar subdivision, facets of a polytope), we wish to preprocess S into a data structure so that, given any query q from a predefined class (line segment, hyperplane), the set of all objects in S intersected by q can be reported efficiently. It is understood that the set S is fixed once for all, while queries are presented and answered on-line. Note that this definition need not make any reference to geometry; in fact it is often useful to view intersection searching abstractly by simply assuming a relation between each q and a subset of S (the "intersected" objects). In the pointer machine framework [131], a data structure for intersection searching is modeled as a directed graph G with bounded outdegree. The graph G is referred to as a *search structure for* S. Some of the nodes are assigned objects of S (not necessarily injectively). Given a query q, the algorithm navigates through the graph G, beginning at some start node, until the subgraph traversed contains at least one node assigned to each object of S intersected by q. For lower bound purposes, it suffices to count the number of nodes visited as a conservative estimate of the query time. (This also covers randomized query-answering algorithms as well.)

We first give a general lemma on the size of any pointer machine data structure for efficient intersection searching; then we establish the desired storage lower bound by exhibiting a set of "hard" queries.

Definition 4.1.1 *A search structure G for a set S is (α, ω)-*effective*, with α a positive constant and ω an additive overhead, if for any query q, we have $|G(q)| \leq \alpha(k + \omega)$. Here $G(q)$ is the set of nodes visited in G while answering query q, and k is the output size, i.e., the number of objects in S intersected by q.*

This definition expresses our focus on search structures that answer queries in time linear in the output size, aside from an additive overhead of $\omega = \omega(|S|)$. Of course, we can also handle arbitrary multiplicative overheads by setting $\alpha = \alpha(|S|)$.

Definition 4.1.2 *A collection of queries $Q = \{q_i\}$ is (m, ω)-favorable for S, with $m > 1$, if Q satisfies the following relevance and independence conditions.*

1. Relevance*: $|S \cap q_i| \geq \omega$, for any query $q_i \in Q$.*

2. Independence*: $|S \cap q_{i_1} \cap \cdots \cap q_{i_m}| = O(1)$, for all possible $i_1 < \cdots < i_m$.*

Here $S \cap q_i$ denotes the set of objects in S intersected by q_i, and $S \cap q_{i_1} \cap \cdots \cap q_{i_m}$ has a similar meaning.

The *relevance* condition means that each query intersects enough objects so that the output size dominates the additive overhead. The *independence* condition gets to the heart of the matter: It is a Zarankiewicz-type condition stating that the bipartite graph induced by objects and queries should be free of large complete subgraphs. The motivation for these definitions comes from a general volume argument [39, 49] about pointer machines. Adapted to our purposes it states that:

Lemma 4.1.3 *Given a search structure G for S and a set Q of queries, assume that G is (α, ω)-effective for some constant α, and Q is (m, ω)-favorable for S. For any ω large enough, the size of G is $\Omega(|Q|\, \omega/m)$.*

Lemma 4.1.3 is the main vehicle for proving lower bounds on the storage required by a search structure with a given query time. Complexity questions are in this way reduced to purely combinatorial ones, involving the existence of "favorable" sets of queries.

The rest of this chapter is organized as follows. In Section 4.2 we establish lower bounds on the complexity of intersecting a planar subdivision (resp. convex polytope) with a query line segment (resp. query plane). These results give us the stepping stone from which we can conclude that 2D fractional cascading is impossible (Section 4.3), and that not all convex polytopes admit of a boundary dominant Dobkin-Kirkpatrick hierarchy (Section 4.4). We establish the $\Omega(n^{3-\varepsilon})$ lower bound for (nonconvex) polytopes in Section 4.5.1, and also provide a nearly matching upper bound (Section 4.5.2).

4.2 Planar Subdivisions and Convex Polytopes

Theorem 4.2.1 *Given a planar subdivision with n vertices, to compute all k edges intersected by a query line in $O(k + \omega)$ time requires $\Omega(n^2/\omega^2)$ storage.*

Theorem 4.2.2 *Given an n-vertex convex polytope in $\mathbf{R}^3$, to compute all k edges intersected by a query plane in $O(k + \omega)$ time requires $\Omega(n^2/\omega^2)$ storage.*

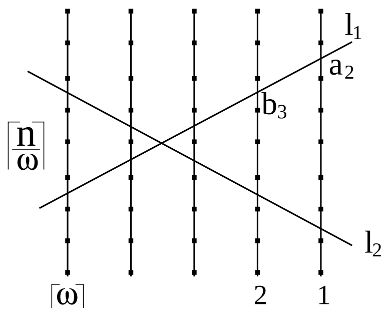

Figure 4.1: Towards a hard convex subdivision.

The lower bound for planar subdivisions is essentially optimal: an $O(k+\log n)$ query time solution using $O(n^2)$ storage is described in [37]. Note that the lower bound also holds for the problem of intersecting a query line with a simple polygon, since any n-vertex planar subdivision can be easily transformed into a simple $O(n)$-gon (by essentially duplicating each edge and creating a polygon of very small area). For convex polytopes the complexity gap is still large. The best solution with $O(k+$ polylog$(n))$ query time requires $O(n^3)$ storage and is obtained by applying the solution for the nonconvex case described in Section 4.5.2.

Both theorems use the same basic starting construction. Take $\lceil \omega \rceil$ congruent vertical segments evenly distributed horizontally, labeled $1, \ldots, \lceil \omega \rceil$, from right to left (Fig. 4.1). Next, decompose each segment into $t = \lceil n/\omega \rceil$ subsegments of the same length, bringing the total number of edges to $\Theta(n)$. Let a_i (resp. b_i) denote the midpoint of the i-th subsegment on segment 1 (resp. 2), counting top down. Of the t^2 lines passing through pairs (a_i, b_j), any one that intersects segment $\lceil \omega \rceil$ is called *hitting*. For example, line l_1 in Fig. 4.1 is such a hitting line. A few simple observations:

Fact 4.2.3 *Every hitting line intersects all vertical segments* 1 *to* $\lceil \omega \rceil$*, and every intersection point is the midpoint of some subsegment.*

To bound the number of hitting lines, observe that any point on segment 1 can join, via a hitting line, at least $\lceil n/\omega \rceil / (\lceil \omega \rceil - 1)$ points on segment 2; and hence,

Fact 4.2.4 *The number of hitting lines is at least* n^2/ω^3.

Let the *canonical subset* of a hitting line refer to the set of subsegments that intersect it. By Fact 4.2.3, intersections take place at midpoints, and so two lines that intersect the same subsegments pass through the same two points and therefore are identical.

Fact 4.2.5 *Each canonical subset is of size* $\lceil \omega \rceil$ *and the intersection of any two of them is of size at most* 1.

Now let S (resp. Q) be the set of subsegments (resp. hitting lines). Fact 4.2.5 implies that Q is $(2, \omega)$-favorable for S. We are now ready to build a convex subdivision around S. In this subdivision, as well as all the others in this chapter, we shall ensure that the subdivision is in "general position", meaning that no two adjacent edges are collinear. Thicken the vertical segments in S, turn them into thin ladder-like concave strips, and add horizontal rungs to ensure the convexity of their decompositions (Fig. 4.2). Finally, cap the top and bottom parts of the ladders with two suitable convex pieces. We redefine S to be the left subsegments of the ladders, and check that, by keeping the ladders thin enough, Q still is $(2, \omega)$-favorable for S: we then are ready to apply Lemma 4.1.3 to derive a lower bound. The only problem is that the intersection search problem defined by Q and S is not, properly speaking, a "line-intersects-subdivision" problem since the output contains edges outside of S. However, by keeping the ladders thin enough, we see that each set of intersected edges in S is a subset at least half the size of the actual set of intersected edges. And so the lower bound for the (S, Q) problem also applies to the "line-intersects-subdivision." Theorem 4.2.1 follows from the fact that the number n of vertices in the resulting subdivision is $O(|S|)$.

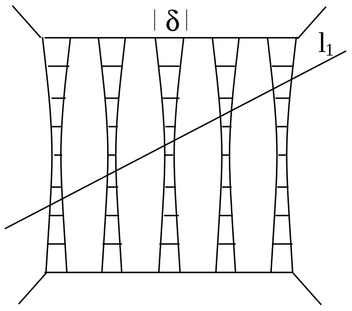

Figure 4.2: Completing the convex subdivision.

The reader might wonder why we did not choose to augment S by including in it all the edges of the subdivision, and then check again that the relevance and independence conditions still hold. The reason is that although augmenting S works in this case, it does not in the more complex configurations discussed below. So, we prefer to use a reduction argument, where the actual geometric problem is reduced to an abstract intersection searching problem specified by a map $Q \mapsto 2^S$.

It is easy to modify the construction used in Theorem 4.2.1 to derive Theorem 4.2.2. The modifications are sufficiently straightforward to dispense with a formal explanation. The idea is to raise the planar subdivision to form the "roof" of a shed-like polytope in $\mathbf{R}^3$. Each hitting line is replaced by a plane passing through that line and perpendicular to the ground, then the set of planes is favorable for the box. A brief sketch follows:

We begin with the polytope of Fig. 4.3, whose xy-projection reproduces the configuration of segments in Fig. 4.1. Next, we replace each rectangle on the roof by a convex polygon with $2(\lceil n/\omega \rceil + 1)$ edges, as shown in Fig. 4.4. (The bounding box is drawn only for illustration purposes.)

We patch the gaps between consecutive polygons on the roof by taking the convex hull (Fig. 4.5). Note that the polygonal chains $p_1, p_2, \ldots$ and $q_1, q_2, \ldots$ both lie in a plane but are not coplanar. Our construction is a lifted version of the hard convex subdivision

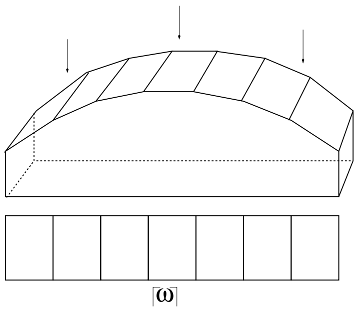

Figure 4.3: A “house” and its projection.

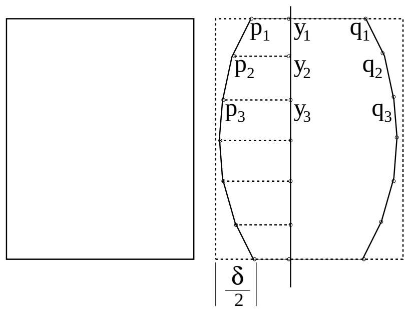

Figure 4.4: Carving out the details.

of Fig. 4.2. The floor and vertical walls of the shed add only a constant number of edges to the intersection with any hitting plane, and so the same lower bound argument applies, leading to Theorem 4.2.2.

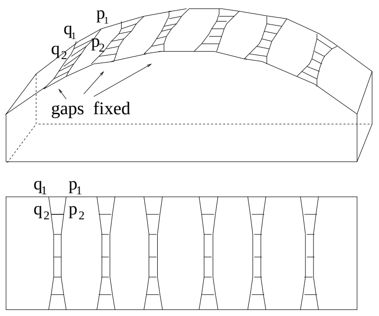

Figure 4.5: Gaps are filled by taking the convex hull.

4.3 Fractional Cascading in Higher Dimension

Fractional cascading is a general technique for speeding up lookup queries in catalogs associated with the nodes of a graph [46]. Specifically, suppose that G is a bounded-degree graph, where each node v is associated with a catalog C_v (which is just a sorted list of numbers). The successor of x in C_v is defined as $\min\{y \in C_v \cup \{\infty\} \mid y \geq x\}$. Given a subset of k nodes in G whose induced subgraph is connected, it is easy, by repeated binary search, to compute the successors of any query x in the k catalogs in time $O(k \log n)$, where n is the combined size of all the catalogs. Fractional cascading reduces the query time to $O(k + \log n)$ while increasing the storage by only a constant factor. The technique has found numerous applications in computational geometry (from which it originated), but also in constraint databases [27], IP routing [31], packet classification [105], data mining [119], geographical information systems [14], etc. This versatility has motivated the design of all sorts of variants: dynamic, probabilistic, parallel, external-memory fractional cascading [17, 14, 64, 92, 110, 130, 127].

An outstanding open problem has been the two-dimensional generalization of fractional cascading: what if we replaced the catalogs at the nodes by planar subdivisions? The question is motivated by more than mere curiosity. Indeed, the wide applicability of the technique stems from the fact that many data structures for multidimensional searching [63, 109] consist of a skeleton graph whose nodes are themselves repositories of auxiliary (often recursively defined) data structures. As long as linear lists reside at the bottom of the hierarchy, fractional cascading can be called into action. In some cases, however, the bottom layer consists of planar maps (i.e., 2D catalogs) and the current technology fails.

2D fractional cascading would immediately lead to improved algorithms for nearest neighbor searching in E^3, intersection search for query segments in planar line arrangements and convex subdivisions, intersection search for query planes and polygons in polytopes, fixed-directional ray shooting in 3D, not to mention numerous instances of range searching. Our result dashes all such hopes. We show that not only generalizing FC to 2D catalogs is impossible but that no pointer machine solutions can provide the sort of logarithmic speed-up associated with fractional cascading.

Furthermore, the counterexamples are hardly pathological: in fact, catalogs consisting of simple parallel strips are enough to break the whole fractional cascading scheme apart. This can seem rather surprising in view of previous results suggesting that such generalizations might, indeed, be possible. For example, navigation among geodesic triangles as described in [45, 96] can be naturally viewed as an instance of 2D fractional cascading. So, it appears that the catalog graphs of geodesically triangulated polygons are more exceptional than previously thought.

We use the results of the previous section to show that 2D fractional cascading is impossible. The construction of Fig. 4.1 tells the whole story: The graph G is a simple path whose nodes are associated with the $\lceil \omega \rceil$ vertical segments. The catalog at a node consists of the lines dual to the midpoints of the $\lceil n/\omega \rceil$ subsegments. We use the duality:

$$(a, b) \mapsto y = ax + b \quad \text{and} \quad y = ax + b \mapsto (-a, b)$$

because it respects above/below relationships. In this way, a catalog appears as a collection of parallel strips, i.e., a planar convex subdivision (Fig. 4.6). A hitting query line dualizes to a query point, and the line/subsegment intersections identify which of the strips

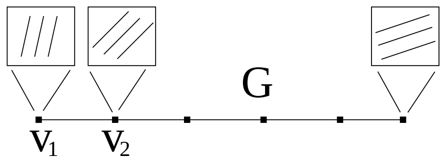

Figure 4.6: A catalog graph in which 2D fractional cascading is impossible.

contain the dual point. Thus, 2D fractional cascading is seen to suffer from exactly the same lower bound as intersection searching for line/planar subdivision (Theorem 4.2.1).

Theorem 4.3.1 *There is a graph with planar subdivisions attached to its nodes such that to perform the same point location query at the k nodes of a connected subgraph in time $O(k+$ polylog$(n))$ requires storage $\tilde{\Omega}(n^2)$, where n is the combined size of all the catalogs.*

4.4 The Dobkin-Kirkpatrick Hierarchy

Let P be an n-vertex convex polytope in $\mathbf{R}^3$ whose vertex set is $V(P)$. A sequence of convex polytopes, $\mathcal{H}(P) = P_0, \ldots, P_k$, is called a (polyhedral) *hierarchy* if: (i) $P_0 = P$ and P_k is a simplex; (ii) $P_{i+1} \subset P_i$ and $V(P_{i+1}) \subset V(P_i)$ for $0 \leq i < k$; (iii) $V(P_i) \setminus V(P_{i+1})$ forms an independent set with respect to the facial graph of ∂P_i. The *size* of $\mathcal{H}(P)$ is defined as $\sum_{i=0}^{k} |V(P_i)|$, its *height* is k, and its *degree* is:

$$\max_{0 \leq i \leq k} \#\text{edges of } P_i \text{ incident to a vertex of } V(P_i) \setminus V(P_{i+1}).$$

The hierarchy is called *Dobkin-Kirkpatrick* (or DK for short) if its size is $O(n)$, its degree is $O(1)$, and its height is $O(\log n)$. It is well known [68] that any convex polytope P admits of a DK hierarchy, which is an essential tool for all sorts of polyhedral operations, mesh simplification, etc. In fact, a polytope will typically have an exponential number of distinct DK hierarchies. Of particular interest are the *boundary dominant* hierarchies: These have the additional property that, given any plane π,

$$\sum_{i=0}^{k} |P_i \cap \pi| = O(|P \cap \pi| + \log n),$$

where | polygon | denotes the number of vertices in the polygon. Intuitively, it means that any planar cross-section should essentially be like the 2D equivalent of a DK hierarchy: a collection of concentric layers with a fraction of the complexity on the outermost layer.

It is folk knowledge that the existence of a *boundary dominant* DK hierarchy would carry with it all sorts of algorithmic benefits: For example, we could use such a hierarchy to compute, in optimal time, the shadow of a polytope cast by a single light source. Among the (usually) exponentially many possible DK hierarchies for a convex polytope, could it be that at least one of them is boundary dominant? We prove that the answer is no by exhibiting polytopes with no boundary dominant DK hierarchies. Of anecdotal interest, we should mention that this purely combinatorial question is resolved by using algorithmically inspired arguments.

A remarkable feature of a DK hierarchy is that it lends itself naturally to navigation along piecewise linear curves and surfaces. Specifically, the pockets formed by $P_i \setminus P_{i+1}$ can be triangulated so as to turn the whole polytope P into a simplicial cell complex $\mathcal{C}$ (with the standard glueing properties one expects of a cell complex). Given any plane π, one can find, in $O(\log n)$ time, a starting point in $P \cap \pi$, and then explore every simplex of $\mathcal{C}$ that intersects π in a breadth first search traversal that takes constant time per simplex [41]. Suppose now that the DK hierarchy is boundary dominant. Because the pockets are each of constant size, the number of intersected simplices would be $O(|P \cap \pi| + \log n)$. So, we could compute the intersection between a query plane and a convex polytope in $O(k+\log n)$ time, using only $O(n)$ storage. This would stand in contradiction with Theorem 4.2.2.

Theorem 4.4.1 *For any n large enough, there exists a convex polytope with n vertices that admits of no boundary dominant DK hierarchy.*

4.5 Nonconvex Polytopes

4.5.1 A Quasi-Optimal Lower Bound

We consider the problem of intersecting a (possibly nonconvex) n-vertex polytope in $\mathbf{R}^3$ with a query plane. We prove a lower bound first; then we give a nearly matching upper bound in Section 4.5.2.

Theorem 4.5.1 *Given an n-vertex polytope in $\mathbf{R}^3$, to compute all k edges intersected by a query plane in $O(k+\omega)$ time requires $\Omega(n^{3-\varepsilon}/\omega^3)$ storage, where ε is an arbitrarily small positive constant.*

Note that this implies a $\Omega(n^{3-\varepsilon})$ lower bound when $\omega(n)$ is polylogarithmic or even of the form n^δ, for any constant $\delta > 0$ with $\varepsilon = \varepsilon(\delta)$. From now on, ε denotes an arbitrarily small positive constant. Since the storage is at least linear in n, we may assume with no loss of generality that $\omega/n^{(2-\varepsilon)/3}$ is small enough. Our proof relies on the construction of a "hard" polytope P and a set Q of query planes that is $(\Theta(\log n), \omega)$-favorable (see definition 4.1.2) for some *designated* edges of P that constitute the set S. For notational convenience, we use n as a parameter that differs from the number of vertices of P by at most a constant factor. As usual, we ensure that of all the edges intersected by any query plane of Q, at least a fixed fraction of them are designated. To achieve a query time of

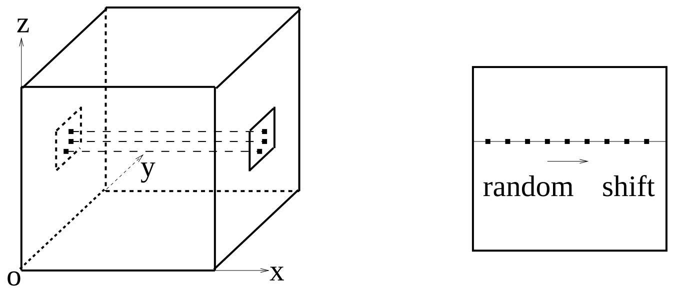

Figure 4.7: Carving P out of a cube.

$O(|P \cap \pi| + \omega)$, we know from Lemma 4.1.3 that the size of the data structure should be $\Omega(|Q|\omega/\log n)$.

The input polytope P is carved out of the unit cube $\mathcal{C} = [0,1]^3$ in several "carving" steps. Let $\varepsilon_0 > 0$ be a small enough absolute constant (i.e., with no dependence on ε). To begin with, we choose $\lceil \omega \rceil$ random points in the subsquare $|2y-1| \leq \varepsilon_0$, $|2z-1| \leq \varepsilon_0$ of the face $x = 0$ of $\mathcal{C}$, independently and uniformly, and join them to the face $x = 1$ by x-parallel segments. Next, decompose each such unit-length segment s into $\lceil 6n/\omega \rceil$ congruent subsegments. For technical reasons, we need to perform a random shift of this decomposition: For each s, pick a random real α uniformly between $-\frac{1}{2}\lceil 6n/\omega \rceil^{-1}$ and $\frac{1}{2}\lceil 6n/\omega \rceil^{-1}$ and move each of the $\lceil 6n/\omega \rceil - 1$ interior endpoints along s (not including the two endpoints at each end of s) by α (Fig. 4.7). Note that each interior endpoint in a given s is shifted by the same amount, but that the $\lceil \omega \rceil$ random shifts are independent.

The next step is to turn this configuration of edges into a bona fide polytope. First, make each horizontal segment into a cylinder with a tiny square base, say, of side length $1/n^2$. Next, attach these cylinders to the face $x = 0$ but *not* to the face $x = 1$; let it come very near the latter but not touch it. What happens to the (randomly shifted) decompositions? They naturally partition the boundary of each cylinder into rectangles. The polytope P consists of the unit cube with the protrusions through $x = 0$ formed by

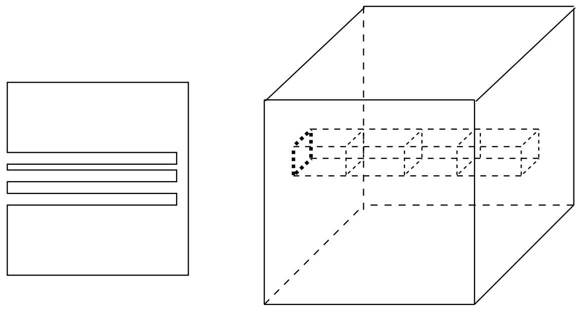
Figure 4.8: Turning transversals into protrusions.

the cylinders (Fig. 4.8). Each shifted endpoint gives rise to four vertices of P: all the x-parallel edges that are incident upon any of them are called *designated* and form the set S. Note that many faces are coplanar in this construction but it is routine to perturb the polytope P to make it simplicial and in general position. Trivally,

Fact 4.5.2 *The number of vertices of P and the number of designated edges are both in $\Theta(n)$. Furthermore, no designated edge is of length larger than $\omega/2n$.*

Next, we define a large set Q of query planes that satisfy both the relevance and independence conditions (Definition 4.1.2). Choose t random points $q_1, \dots, q_t$ uniformly in the square $1 - \varepsilon_0 \leq x, y \leq 1$ on the face $z = 1$, where $t = \lceil n^{2-\varepsilon}/\omega^3 \rceil$. Next, along each segment Oq_i, add the points $(3j\omega/n)q_i$, for all integers $n/6\omega < j < n/5\omega$. This defines a set $\{q\}$ of $\Theta(tn/\omega)$ points,[1] which in turn specifies the set Q of query planes, each defined by the equation $\pi_q : \langle p, q \rangle = \|q\|_2^2$ (Fig. 4.9). In other words, π_q is the plane passing through q and normal to Oq.

Let S_q be the slab consisting of all points at distance at most ω/n from the plane π_q. By using a Heilbronn-type argument [38], we can prove along the lines of [49]:

[1]By abuse of terminology we also use q to denote the vector Oq.

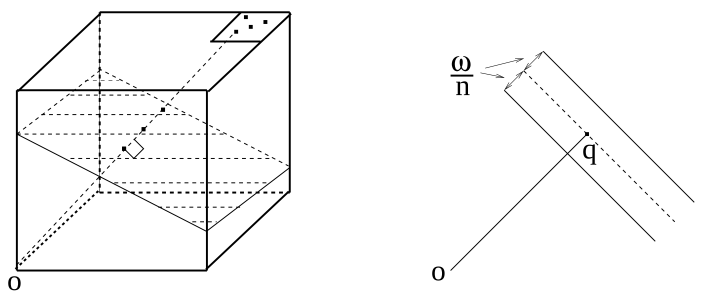

Figure 4.9: The query set Q.

Lemma 4.5.3 *With probability $1 - o(1)$ the query set Q satisfies the following property: for some fixed c large enough, any subset of at least $c \log n$ planes in Q contains three planes $\pi_{\widehat{q}_1}, \pi_{\widehat{q}_2}, \pi_{\widehat{q}_3}$ such that the volume of the polyhedron $S_{\widehat{q}_1} \cap S_{\widehat{q}_2} \cap S_{\widehat{q}_3}$ is $O(1/n^{1+\varepsilon})$.*

Note that the probability given in the lemma is over the initial random choices of the set Q of query planes, and the statement is over *all* large enough subsets of Q.

Proof: We may allow ourselves a subset $\mathcal{S}$ of size $c \log t$ planes, since $\log t = O(\log n)$. Recall that each plane is normal to some segment Oq_i where q_i is a point on the face $z = 1$. Slabs normal to the same Oq_i do not intersect, and so we may assume without loss of generality that the planes of $\mathcal{S}$ are normal to distinct segments $Oq_1, Oq_2, \ldots, Oq_{|\mathcal{S}|}$. Because the q_i's are chosen randomly in an ε_0-by-ε_0 square at $z = 1$, they have the following Heilbronn-type property (proven in [38]). For c large enough, with probability $1 - o(1)$, the convex hull of $\{ q_1, \ldots, q_{|\mathcal{S}|} \}$ is a polygon of area $\Omega(\varepsilon_0^2 (\log t)/t)$. A triangulation of this convex hull produces at least one triangle of area $\Omega(\varepsilon_0^2/(ct))$, which we relabel $q_1 q_2 q_3$. We now show that the corresponding planes $\pi_{\widehat{q}_1}, \pi_{\widehat{q}_2}, \pi_{\widehat{q}_3}$ satisfy the desired condition (note that $\widehat{q}_i$ is a point that lies on segment Oq_i, for $i = 1, 2, 3$):

$$\text{vol} \bigcap_{i=1}^{3} S_{\widehat{q}_i} = O(1/n^{1+\varepsilon}). \tag{4.1}$$

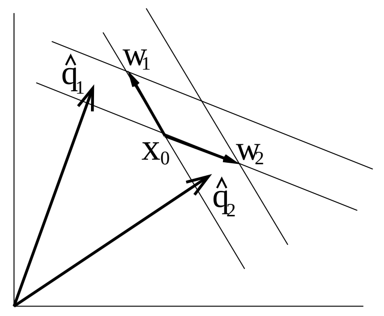

Figure 4.10: Intersection parallelotope.

The polyhedron $S_{\widehat{q}_1} \cap S_{\widehat{q}_2} \cap S_{\widehat{q}_3}$ is spanned by w_1, w_2, w_3 where,

$$\langle w_j, \widehat{q}_i \rangle = \begin{cases} (2\omega/n)\|\widehat{q}_i\|_2 & \text{if } i = j, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

In other words, $S_{\widehat{q}_1} \cap S_{\widehat{q}_2} \cap S_{\widehat{q}_3}$ is the set of linear combinations $x_0 + \sum_i \alpha_i w_i$, for $0 \leq \alpha_i \leq 1$, where x_0 is the unique point satisfying

$$\langle x_0, \widehat{q}_i \rangle = \left(\|\widehat{q}_i\|_2 - \frac{\omega}{n} \right) \|\widehat{q}_i\|_2$$

for $1 \leq i \leq 3$ (Fig. 4.10). Denote by $[w]$ the 3-by-3 matrix (w_1, w_2, w_3) and define the matrices $[q]$ and $[\widehat{q}]$ similarly. Finally, let Λ be the diagonal matrix with $\Lambda_{ii} = \|\widehat{q}_i\|_2$. It follows from $[w]^T [\widehat{q}] = (2\omega/n)\Lambda$ that

$$\det [w] \det [\widehat{q}] = \left(\frac{2\omega}{n} \right)^3 \prod_{i=1}^{3} \|\widehat{q}_i\|_2$$

Recall that q_i ($i = 1, 2, 3$) is a point in the small square in the plane $z = 1$, and $\widehat{q_i}$ lies on Oq_i. It then follows $\prod_i \|\widehat{q_i}\|_2(\det[q]) = \prod_i \|q_i\|_2(\det[\widehat{q}])$, and each $\|q_i\|_2$ is $\Theta(1)$,

$$\det[w]\det[q] = \Big(\frac{2\omega}{n}\Big)^3 \prod_{i=1}^{3} \|q_i\|_2 = \Theta\Big(\frac{\omega^3}{n^3}\Big)$$

The determinant of $[q]$ is, in absolute value, at least proportional to the area of triangle $q_1q_2q_3$, which is $\Omega(1/t)$. By our choice of $t = \lceil n^{2-\varepsilon}/\omega^3\rceil$, $|\det[w]| = O(t\omega^3/n^3) = O(1/n^{1+\varepsilon})$. Note that vol $\bigcap_i S_{\widehat{q_i}} = |\det[w]|$, which proves (4.1). $\square$

Lemma 4.5.3 shows that, for any subset $K \subseteq Q$ of size at least $c\log n$, the volume of $\bigcap\{\, S_q \mid \pi_q \in K \,\}$ is itself in $O(1/n^{1+\varepsilon})$. We now verify that with high enough probability the relevance and independence conditions hold.

Lemma 4.5.4 *Any plane in Q intersects at least $\lceil\omega\rceil$ designated edges.*

Proof: It suffices to show that, given any y, z with $|2y - 1| \le \varepsilon_0$ and $|2z - 1| \le \varepsilon_0$, the points $p_0 = (0, y, z)$ and $p_1 = (1, y, z)$ lie cleanly on opposite sides of any plane $\pi_{\widehat{q}}$ of Q. (The adjective "cleanly" refers to the fact that this must remain true after turning segments into thin cylinders and perturbing P slightly.) Recall that $\widehat{q} = (\beta u, \beta v, \beta)$, where

$$1 - \varepsilon_0 \le u, v \le 1 \qquad \text{and} \qquad \frac{1}{2} < \beta < \frac{3}{5}\,.$$

We easily verify that for some small enough $c_0 > 0$:

$$\langle p_0, \widehat{q}\rangle - \|\widehat{q}\|_2^2 = \beta v y + \beta z - \beta^2(u^2 + v^2 + 1) = -\frac{\beta}{2} + O(\varepsilon_0) < -c_0.$$

$$\langle p_1, \widehat{q}\rangle - \|\widehat{q}\|_2^2 = \beta u + \beta v y + \beta z - \beta^2(u^2 + v^2 + 1) = \frac{\beta}{5} - O(\varepsilon_0) > c_0.$$

$\square$

Lemma 4.5.5 *With probability* $1 - o(1)$*, given any set of at least* $c \log n$ *planes in* Q*, at most a constant number of designated edges intersect all of them.*

Proof: Let K be a set of at least $c \log n$ planes in Q and let K^* be the (interior of the) polyhedron $\mathcal{C} \cap \bigcap\{\, S_q \mid \pi_q \in K \,\}$. Recall that to build P we choose random points on the face $x = 0$ of $\mathcal{C}$. Let $(0, y, z)$ be one of them and let $\ell = \ell(y, z)$ be the line passing through it parallel to the x-axis. The construction of P proceeds by choosing points on ℓ at regular intervals of length $\lambda = \lceil 6n/\omega \rceil^{-1}$ and shifting the $\lceil 6n/\omega \rceil - 1$ endpoints by a fixed, random amount between $-\lambda/2$ and $\lambda/2$. At the end of this process, let $N(K^*, \ell)$ denote the number of endpoints in $K^* \cap \ell$. A simple calculation shows that the expected value of $N(K^*, \ell)$ (over the random shift) is at most $|K^* \cap \ell|/\lambda$, where $|K^* \cap \ell|$ designates the length of the corresponding segment (we say “at most” and not “equal to” for the rare case where K^* intersects ℓ very near the boundary of $\mathcal{C}$). It follows that

$$\begin{aligned} \operatorname{vol} K^* &= \int_{[0,1]^2} |K^* \cap \ell(y,z)| \, dy \, dz \\ &\geq \lambda \int_{[0,1]^2} \mathbf{E}\, N(K^*, \ell(y,z)) \, dy \, dz \,. \end{aligned}$$

Restricting the integration domain to the ε_0-by-ε_0 squares to which each ℓ is actually adjacent and identifying $\mathbf{E}\, N(K^*, \ell(y,z))$ with the conditional expectation of $N(K^*, \ell)$ given $\ell = \ell(y, z)$, we find that

$$\begin{aligned} \operatorname{vol} K^* &\geq \lambda \int_{[(1-\varepsilon_0)/2, (1+\varepsilon_0)/2]^2} \mathbf{E}\, N(K^*, \ell(y,z)) \, dy \, dz \\ &= \Omega(\varepsilon_0^2 \lambda)\, \mathbf{E}_{(y,z)}\, \mathbf{E}\, [\, N(K^*, \ell) \mid \ell = \ell(y,z) \,] \\ &= \Omega(\varepsilon_0^2 \lambda)\, \mathbf{E}\, N(K^*, \ell) \,, \end{aligned}$$

where ℓ is a random x-parallel segment of the type used in the construction (i.e., with endpoints in the tiny squares at $x = 0, 1$). It follows that

$$\mathrm{Prob}[\, N(K^*, \ell) > 0\,] = O\Big(\frac{\mathrm{vol}\, K^*}{\varepsilon_0^2 \lambda}\Big)\,.$$

On the other hand, a simple geometric argument shows that $|S_q \cap \ell| = (2\omega/n)/\cos\theta$, where θ is the angle between Oq and the x axis.

$$\cos\theta = q_x/\|q\|_2 = \Theta(1),$$

and so

$$|S_q \cap \ell| = \Theta(\lambda)$$

and, therefore, any given ℓ can contribute only $O(1)$ such endpoints. By Lemma 4.5.3, it follows that a given ℓ contributes either no endpoint to K^*, or if it does, the number of endpoints is $O(1)$ and this event happens with probability $O(\mathrm{vol}\, K^*)/\varepsilon_0^2\lambda = O(\varepsilon_0^{-2})/\omega n^{\varepsilon}$. The choice of ℓ is repeated $\lceil \omega \rceil$ times independently, so the expected number is $O(n^{-\varepsilon})$. By a Chernoff-type estimate (Theorem A.1.12 [13]), the probability that the number of endpoints in K^* exceeds $n^{-\varepsilon}\Delta$ is at most $O(1/\Delta)^{O(n^{-\varepsilon}\Delta)}$. Setting $\Delta = bn^{\varepsilon}$, for some large enough constant $b = b(\varepsilon)$, we find that $O(1)$ endpoints lie in K^* with probability at most n^{-10}.

The size of Q is $O(n^3)$, which bounds by $O(n^9)$ the number of triplets of slabs that can be formed in Lemma 4.5.3. We can thus ensure that with high probability no K^* contains more than a constant number of endpoints. By Fact 4.5.2, we know that the designated edges of P are all of length at most $\omega/2n$ and so at most $O(1)$ of them can intersect any set of query planes of size at least $c\log n$. (The 2 in the denominator is an overly generous slack factor to account for the distortions resulting from turning ℓ into a cylinder.) $\Box$

We observed earlier that, because of Lemma 4.1.3, to achieve a query time of $O(|P \cap \pi| + \omega)$ requires $\Omega(|Q|\omega/\log n)$ storage. By Lemmas 4.5.4 and 4.5.5, we now have a polytope P and a set Q of query planes that is $(\Theta(\log n), \omega)$-favorable for the $\Theta(n)$ designated edges of P. The storage requirement is $\Omega(|Q|\omega/\log n) = \Omega(n^{3-\varepsilon}/\omega^3 \log n)$, which, after suitably readjusting ε, proves Theorem 4.5.1.

4.5.2 The Upper Bound

We prove a nearly matching upper bound for the case where $\omega = O(\log n)$.

Theorem 4.5.6 *Given an n-vertex polytope in $\mathbf{R}^3$, there is a data structure of size $O(n^3)$ that allows us to compute all k edges intersected by a query plane in $O(k + \log n)$ time.*

Proof: Actually, our solution is more general than that: the input can be any set of line segments in $\mathbf{R}^3$. We begin with an $O(n^4)$ solution, which we improve to $O(n^3)$ in a second stage. We dualize the problem by transforming the endpoints of the input segments into $2n$ planes, using the polarity $ax + by + cz = 1$. A line segment is dualized into a double wedge. We can preprocess the arrangement of $2n$ planes for fast point location [42] so that, given a point (here, the dual of the query plane), the convex cell that contains it can be found in $O(\log n)$ time. If each cell keeps a list of the double wedges that enclose it, then an intersection query can be answered in time $O(k + \log n)$, using $O(n^4)$ storage.

We use filtering search to reduce the storage to $O(n^3)$. Think of the cells as forming the nodes of a graph, with an edge joining two nodes whose corresponding cells share a facet. By doubling each edge, we can form an Eulerian tour that visits all the nodes. Now observe that the wedge lists in the $O(n^4)$ solution have a great deal of “coherence.” Indeed, fix a double wedge and observe the nodes of the tour in whose lists it appears: these nodes form intervals along the tour, whose endpoints correspond to an entry into or an exit from the wedge in question. Thus, there are only $O(n^3)$ such intervals. So the problem is reduced to this: given m intervals on a line, where $m = O(n^3)$, build a data structure of size $O(m)$, such that, given a query x, all k intervals that contain x can be reported in $O(k + \log m)$ time. The *window list* of [37] does just that. □

4.6 Open Problems

We have proven a $\Omega(n^{3-\varepsilon})$ space lower bound for any algorithm in the pointer machine model that finds the set of k edges of a n-vertex polytope intersected by a query plane in time $O(k+\text{ polylog}(n))$, and it matches the upper bound to within a factor of n^{ε} for arbitrarily small $\varepsilon > 0$ in the general (nonconvex) case. However, in the convex case the lower bound drops to (roughly) $\Omega(n^2)$ and the best known upper bound is still the general $O(n^3)$ bound. The main open problem is to bridge this big gap.

In the general case, the $n^{-\varepsilon}$ factor in the lower bound was introduced in the construction of a hard query set in order to make sure that certain probability is small enough. We believe that there might be a better construction (some deterministic number-theoretical construction?) that will improve the lower bound from $\Omega(n^{3-\varepsilon})$ to $\Omega(n^3/\text{ polylog}(n))$. To remove this extra factor completely (i.e., get a tight bound of $\Omega(n^3)$), however, seems very difficult.

Finally, we mention an open problem in the related area of range searching. In halfspace reporting, we need to preprocess a set of n points in $\mathbf{R}^d$ so that, given any halfspace we can quickly report all the k points inside of it, preferably in time $O(k+\text{ polylog}(n))$. We expect similar techniques to give a space lower bound $\Omega(n^{\lfloor d/2 \rfloor - \varepsilon})$ for this problem in the pointer machine model when $O(k+\text{ polylog}(n))$ time is used. This would match the best known upper bound to within a factor of n^{ε}. Note that for orthogonal and simplex range reporting, nearly optimal lower bounds are already known [39, 49].

Chapter 5

A Lower Bound for Approximate Nearest Neighbor Searching

5.1 The Approximate Nearest Neighbor Searching Problem

5.1.1 Introduction

Nearest neighbor searching (henceforth, NNS) is a fundamental problem in computational geometry with applications to a number of areas [56, 66, 55, 83, 25, 81]. A typical setting for this problem is: given a set of n points (called a database) in a vector space of dimension d (for example, d-dimensional Euclidean space), we want to preprocess the database so that given any query point q, we can quickly find the closest point to q in the database. In low dimensions the problem is well solved [73]. On the other hand, many applications involve database points represented as vectors in high dimensional spaces. Unlike the low dimensional case, the problem is considerably more difficult in high dimensions [70, 140, 52, 6]. To our knowledge, there is currently no algorithm with storage $poly(n, d)$ and query time $poly(\log n, d)$. This is often referred to as the “curse of dimensionality”.

It is then natural to consider the *approximate* nearest neighbor searching problem (henceforth, ANNS): instead of insisting on a true closest point to the query q, we only seek a β-approximate nearest neighbor (henceforth, an β-ANN) of q, for some approximation factor $\beta > 1$. An β-ANN of q is any point p such that $dist(p, q) \leq \beta \cdot dist(p', q)$ for any p' in the database. There has been considerable research on ANNS [15, 16, 53, 35, 103], culminating in a couple of recent papers [101, 104] that give randomized algorithms with the following guarantee: approximation factor $\beta = 1 + \varepsilon$ for any $\varepsilon > 0$, space $d^{O(1)}n^{O(1/\varepsilon^2)}$ and query time $((d \log n)/\varepsilon)^{O(1)}$. Thus ANNS does not suffer from the curse of dimensionality, at least from the point of view of randomized algorithms and a fixed $\varepsilon > 0$.

5.1.2 Previous Lower Bounds

Several lower bounds for NNS and ANNS are known in Yao’s cell-probe model [136]. The cell-probe model is a general data structure model for measuring the number of memory accesses required by a search algorithm. Given a database of n points, a table T is built in preprocessing: it consists of a set of cells with possibly different sizes. To answer a query, the algorithm makes several probes to T. Each probe is made by computing an index k and looking up the entry $T[k]$. The running time is defined as the maximum number of probes needed (in the worst case) to correctly answer any possible query. Because of its generality, a lower bound proved in the cell-probe model can be applied to any sequential algorithms. In the cell-probe model there are three parameters

s, b and t: s denotes the number of cells in the table; b denotes the maximum cell size in terms of the number of bits[1]; and t denotes the number of probes made on the table.

All previous lower bounds for NNS and ANNS are proved in the Hamming cube [29, 33, 21]. In this setting, the database points and the query are from the Hamming cube C_d (the d-dimensional binary cube $\{0,1\}^d$), and the distance function is the Hamming distance. All previous lower bounds [29, 33, 21] are established in the following scenario: $s = (nd)^{O(1)}$ and $b = (d \log n)^{O(1)}$, and lower bounds are given for t. They also make the assumption that $d = \omega(\log n)$. This assumption is necessary because for $d = O(\log n)$, the trivial solution of storing all possible answers uses space $n^{O(1)}$ and makes a single probe for each query. Here we also make this assumption.

Depending on which problem is considered (NNS or ANNS), and whether the algorithm is deterministic or randomized, there are four cases.

- *NNS, deterministic.* A lower bound of $t = \Omega(d/\log n)$ is proved by Borodin et al. [29] for any deterministic algorithm that solves NNS, when $s = poly(n, d)$ and $b = poly(\log n, d)$.

- *NNS, randomized.* A lower bound of $t = \Omega(d/\log n)$ is proved by Barkol and Rabani [21] for any randomized algorithm that solves NNS, when $s = poly(n, d)$ and $b = poly(\log n, d)$.

- *ANNS, deterministic.* The only previous lower bound in this case is proved by Chakrabarti et al. [33] (and presented in much more detail in [42, 32]). Specifically, it is shown that $t \geq \delta \log\log d/(2 \log\log\log d)$ when the approximation factor is at most $2^{\lfloor (\log d)^{1-\delta} \rfloor}$, for any fixed constant $\delta > 0$.

[1]The cells may have different sizes. This assumption makes the model even more general.

- *ANNS, randomized.* Very recently (after the result in this thesis was published), Chakrabarti and Regev [34] proved an optimal lower bound of $\Omega(\log\log d/\log\log\log d)$ on the query time of any randomized algorithm that solves ANNS. The approximation factor may be as loose as $2^{\lfloor(\log d)^{1-\delta}\rfloor}$, for any fixed $\delta > 0$.

5.1.3 The New Result

We prove a new lower bound for ANNS in the deterministic case in the Hamming cube C_d. Thus the problem considered in this thesis is the same as [33], and our lower bound is significantly stronger than that one. Here is our result[2].

Theorem 5.1.1 *For the β-approximate nearest neighbor searching problem with n points in $C_d = \{0,1\}^d$, any deterministic algorithm in the cell-probe model that uses $(nd)^{O(1)}$ cells with maximum cell size $d^{O(1)}$, must make $\Omega(d/(\beta^2 \log n))$ probes in the worst case. This holds for any $\beta \le \sqrt{d}/20$. Specifically, for any fixed $\varepsilon > 0$ and any β up to $2^{\lfloor(\log d)^{1-\varepsilon}\rfloor}$, the lower bound is $d^{1-o(1)}$.*

Our result is a significant improvement on the previous lower bound [33], which says that, under exactly the same conditions as stated in Theorem 5.1.1, for any fixed $\varepsilon > 0$ and any approximation factor up to $2^{\lfloor(\log d)^{1-\varepsilon}\rfloor}$, the query time is $\Omega(\frac{\log\log d}{\log\log\log d})$. Here our result implies that for any approximation factor up to $2^{\lfloor(\log d)^{1-\varepsilon}\rfloor}$, the query time lower bound is $d^{1-o(1)}$.

Our proof is much simpler than the proof in [33], which is based on a complicated hierarchical structure. On the other hand, our proof uses the standard richness technique [115] and follows essentially the same line as the proof in [29]. Note that the

[2]As usual, $\log$ refers to logarithm to the base 2.

approximation factor in Theorem 5.1.1 is quite generous: in fact, it is trivial to achieve d-approximation by returning an arbitrary database point.

We comment that ANNS can be solved by a *randomized* algorithm that makes only $O(\log \log d)$ probes to a table with parameters $s = (nd)^{O(1)}$ and $b = (d \log n)^{O(1)}$ [101, 104]. Thus our lower bound gives a sharp contrast of $d^{1-o(1)}$ versus $O(\log \log d)$ between the search time complexity of deterministic versus randomized algorithms for ANNS. This shows that randomization is essential in the algorithms of [101, 104].

5.2 Preliminaries for the Proof

Our proof is very similar to the lower bound proof for NNS by Borodin et al. [29]. The basic idea there is to consider the decision version of NNS which asks, for some given $\lambda > 0$, whether or not there exists a database point within distance λ from the query. The advantage of considering the decision version of NNS is that it has a binary communication matrix[3], so that we can apply the richness technique from asymmetric communication complexity to derive a communication lower bound for the decision version. Such a communication lower bound immediately implies the desired cell probe lower bound due to a reduction in [115].

In the attempt to apply the same proof technique to ANNS, we need to define its decision version first. However, the meaning of the decision version of ANNS is not as clear as that of NNS. This is because a correct answer to an ANNS query could be any database point in the allowed approximation region. To handle this problem, we exploit a certain gap regarding the distance relations that can hold among all possible queries and

[3]This is a matrix of all possible problem instances of the decision version of NNS, with each row being a possible query and each column a possible database, and the entry indexed by that row and column being the answer for that particular instance.

databases. This idea leads to the following definition of the decision version of ANNS: given $\lambda > 0$ and an approximation factor $\beta > 1$, if all database points are at distance more than λ from the query then output 1; if there exists a database point within distance λ/β from the query then output 0; otherwise output 0 or 1 arbitrarily. Note that unlike the NNS case, there are many possible decison versions of ANNS. We prove a lower bound that holds for all of them using the framework of [29]. This lower bound then holds for ANNS.

In the following we review the necessary preliminaries for the proof. Most of them have appeared in [29], and we include them here for completeness. We begin with a few useful facts. For $0 < p < 1$, the entropy function is $H(p) = -p \log p - (1-p) \log (1-p)$. The following fact is an estimation on the entropy function when p is close to $1/2$.

Fact 5.2.1 *For $x > 0$ small enough, $H(1/2 - x) = 1 - (2 \log e) x^2 + O(x^4)$.*

A proof of this fact follows by approximating $\log (1/2 - x)$ and $\log (1/2 + x)$ by their respective Taylor's series expansions for small $x > 0$. Let $B_d(\lambda)$ denote the Hamming ball of radius λ centered at an arbitrary point in C_d. We need to bound $|B_d(\lambda)|$, the number of points inside this Hamming ball. The following fact is probably folklore.

Fact 5.2.2 *For any $0 < p < 1/2$, $|B_d(pd)| \leq 2^{dH(p)}$.*

Proof: Since $p < 1/2$, for any $i \leq pd$, we have $p^{pd}(1-p)^{(1-p)d} \leq p^i (1-p)^{d-i}$. Thus,

$$\begin{aligned} |B_d(pd)| 2^{-dH(p)} &= |B_d(pd)| p^{pd} (1-p)^{(1-p)d} \\ &= \sum_{i=0}^{pd} \binom{d}{i} p^{pd} (1-p)^{(1-p)d} \\ &\leq \sum_{i=0}^{d} \binom{d}{i} p^i (1-p)^{d-i} = 1. \end{aligned}$$

$\square$

We also need the following standard Chernoff bound.

Fact 5.2.3 (Chernoff bound [42]) *For every* $a > 0$, $|B_d(d/2 - a)| \leq e^{-a^2/(2d)} \cdot 2^d$.

Another useful result is Harper's isoperimetric inequality [95, 29].

Fact 5.2.4 (Harper's isoperimetric inequality [95]) *For* $A \subset C_d$*, let* $r > 0$ *be such that* $A \geq |B_d(r)|$*. Then for every* $\lambda > 0$, $|B_d(A, \lambda)| \geq |B_d(r + \lambda)|$*, where* $B_d(A, \lambda)$ *denotes the set of cube points at distance at most* λ *from a point in* A*.*

Next we state a relationship between *asymmetric communication complexity* and cell-probe complexity. Let a function f: $\mathcal{X} \times \mathcal{Y} \to U$. In the asymmetric communication complexity model there are two players, Alice and Bob. Alice gets an input $x \in \mathcal{X}$ and Bob gets an input $y \in \mathcal{Y}$, and the goal is to compute $f(x, y) \in U$ (that is, at least one player should know $f(x, y)$ at the end of this communication game). The complexity measure is the total number of bits communicated by each side, and an $[a, b]$-protocol is one in which Alice sends a bits and Bob sends b bits. The following observation [114] shows that lower bounds in asymmetric communication complexity lead to lower bounds in the cell-probe model.

Lemma 5.2.5 (Miltersen [114]) *For any function* f *if there is a deterministic (resp. randomized) solution in the cell probe model with parameters* s*,* b*, and* t*, then there is a deterministic (resp. randomized)* $[t\lceil \log s \rceil, tb]$*-protocol for the corresponding communication problem.*

It thus suffices to show a strong lower bound for ANNS in the asymmetric communication complexity model. A general technique for this purpose is the richness technique [115]. We now briefly review it. Let f: $\mathcal{X} \times \mathcal{Y} \to \{0, 1\}$. We associate a communication matrix M_f with the function f. The rows of M_f are indexed by the possible inputs to Alice and the columns by the possible inputs to Bob, and the (x, y) entry of M_f is $f(x, y)$. Following the notations in [29], we say that a communication problem f is $[u, v]$-rich if M_f has at least v columns each containing at least u ones. The richness technique is given by the following lemma [115].

Lemma 5.2.6 (Miltersen et al. [115]) *Let f be $[u, v]$-rich. If f has a deterministic $[a, b]$-protocol then M_f contains a submatrix of dimension at least $u/2^{a+2} \times v/2^{a+b+2}$ containing only 1-entries.*

To use this lemma to prove lower bounds for f in the asymmetric communication complexity model, we need to show: (1) M_f is rich enough; (2) M_f does not contain any large 1-monochromatic submatrix.

5.3 A Near-Optimal Lower Bound for ANNS

5.3.1 The (λ, β)-Approximate Neighbor Problem

We define the (λ, β)-approximate neighbor problem by defining its communication function f. Suppose Alice has a query $x \in C_d$ and Bob has a database $D \in C_d^n$ (we allow repetitions in the database). Let $h(x, D)$ be the distance between x and its nearest neighbor in D. A function $f : x \times D \to \{0, 1\}$ is called a *valid* communication function for the (λ, β)-approximate neighbor problem if:

$$f(x,D) = \begin{cases} 1 & \text{if } h(x,D) > \lambda; \\ 0 & \text{if } h(x,D) \le \lambda/\beta; \\ \text{arbitrary } (0 \text{ or } 1) & \text{otherwise.} \end{cases}$$

Note that this definition gives a family $\mathcal{F}$ of functions. We will prove a communication complexity lower bound that holds for *every* $f \in \mathcal{F}$. Before this we first show a reduction from the (λ, β)-approximate neighbor problem to β-ANNS.

Fact 5.3.1 *Suppose there is a $[a, b]$-protocol for β-ANNS in the asymmetric communication complexity model, then there is a $[a, b+d]$-protocol for some valid communication function f of the (λ, β)-approximate neighbor problem. This is true for any $\lambda > 0$.*

To see why, suppose we have a $[a, b]$-protocol for β-ANNS. We run it on input (x, D) and get a point $p \in D$ on at least one side (Alice or Bob), such that p is a β-approximate nearest neighbor of x. In case Bob is the only one who knows p, he sends it to Alice using d bits. Thus both sides know p after a $[a, b+d]$-protocol. Alice compares $H(x,p)$ with λ and outputs $f(x,D) = 1$ if and only if $H(x,p) > \lambda$. The correctness of this protocol is justified as follows: Let p^* be a true nearest neighbor of x. If $H(x,p^*) > \lambda$ then the protocol is supposed to output 1, it does so because $H(x,p) \ge H(x,p^*) > \lambda$. On the other hand if $H(x,p^*) \le \lambda/\beta$ then the protocol is supposed to output 0, it does so because $H(x,p) \le \beta H(x,p^*) \le \lambda$.

In view of the discussions above, a good communication lower bound for β-ANNS follows immediately from a good communication lower bound that applies to *every* valid communication function for the (λ, β)-approximate neighbor problem.

5.3.2 The Proof of the Lower Bound

Let f be any valid communication function for the (λ, β)-approximate neighbor problem. We prove the following lower bound for any asymmetric communication protocol that computes f.

Lemma 5.3.2 *Let $\beta \leq \sqrt{d}/20$. There exists a λ such that in any communication protocol that computes f, either Alice sends $\Omega(d/\beta^2)$ bits or Bob sends $\Omega(nd/\beta^2)$ bits. In particular, for any fixed $\varepsilon > 0$ and any β up to $2^{\lfloor (\log d)^{1-\varepsilon} \rfloor}$, either Alice sends at least $d^{1-o(1)}$ bits or Bob sends at least $nd^{1-o(1)}$ bits.*

Note that the lower bound in Lemma 5.3.2 is nearly optimal (resp. asymptotically optimal) when $\beta = d^{o(1)}$ (resp. when $\beta = O(1)$). In fact, any function of the query and the database can be computed after Alice sends her input using d bits or Bob sends his input using nd bits. Below is the proof of Lemma 5.3.2.

Proof: We set the relevant parameters as follows:

- n and d are large enough integers such that $d = \omega(\log n)$ and d is even.
- $\beta \leq \sqrt{d}/20$; it is also larger than any constant appearing in the proof.
- $\lambda = d/2 - \sqrt{2d \ln (2n)}$; $\lambda_0 = \lambda/\beta$.

We look at the communication matrix M_f that has 2^d rows and 2^{nd} columns.

Claim 5.3.3 *M_f is $[2^{d-1}, 2^{nd}]$-rich.*

To see this, we fix a column (a database). For each point p in the database, the number of queries within distance at most λ from p is $|B_d(\lambda)| \leq 2^{d-1}/n$ by Fact 5.2.3, thus the number of queries within distance at most λ from at least one point in the database is at most 2^{d-1}. This means that there are at least 2^{d-1} queries such that for each such a query its nearest neighbor in the chosen database is at distance more than λ from it. By the definition of f, all the entries indexed by these 2^{d-1} queries in that fixed column are 1. Claim 5.3.3 follows. The next claim is that M_f does not contain any large 1-monochromatic submatrix.

Claim 5.3.4 *Every* $2^{d-d/(169\beta^2)} \times 2^{nd-nd/(32\beta^2)}$ *submatrix of* M_f *contains a zero entry.*

Proof: Consider any fixed set Q of queries of cardinality $2^{d-d/(169\beta^2)}$. Let λ_Q be the largest integer such that $|B_d(\lambda_Q)| \leq |Q|$. It must be the case that $\lambda_Q < d/2-1$. Otherwise we have $|Q| = 2^{d-d/(169\beta^2)} \geq |B_d(d/2-1)| > 2^{d-1} - \binom{d}{d/2} \geq 2^{d-2}$, a contradiction with $\beta < \sqrt{d}/20$.

Let $p = (\lambda_Q + 1)/d$, then $0 < p < 1/2$. By Fact 5.2.2, $|Q| < |B_d(\lambda_Q + 1)| \leq 2^{dH(p)}$. We then have $1 - 1/(169\beta^2) < H(p)$. This shows that $H(p)$ is close to 1 for β large enough, and hence p is close to $1/2$. Applying Fact 5.2.1 we have $1 - (1/2 - p)^2 > H(p) > 1 - 1/(169\beta^2)$, and so

$$1/2 - p < 1/(13\beta). \tag{5.1}$$

We now consider the set $B_d(Q, \lambda_0)$. Recall that $B_d(Q, \lambda_0) = \bigcup_{q \in Q} B_d(q, \lambda_0)$. We say that a point $p \in C_d$ is *good* for Q if $p \notin B_d(Q, \lambda_0)$. We say that a database D is *good* for Q if every point in D is good for Q. Note that, from the above definitions, if a database D is *not good* for Q, then there must exist $p \in D$ and $q \in Q$ such that $H(p, q) \leq \lambda_0$. Since f is a valid communication function, then by its definition (also recall $\lambda_0 = \lambda/\beta$), $f(q, D) = 0$. This shows that once we pick a column indexed by a "not good" database for Q, there exists a zero entry in the $|Q| \times 1$ submatrix formed by Q and that column, and this is true for any Q. In the following we show that for any Q of cardinality $2^{d-d/(169\beta^2)}$, the number of good databases for Q is less than $2^{nd-nd/(32\beta^2)}$. It then follows immediately that every $2^{d-d/(169\beta^2)} \times 2^{nd-nd/(32\beta^2)}$ submatrix contains at least one zero entry.

We first bound the number of points in C_d that are “not good” for Q. Clearly this number is $|B_d(Q, \lambda_0)|$ which is at least $|B_d(\lambda_Q + \lambda_0)|$ by Fact 5.2.4. We have:

$$\begin{aligned}
|B_d(\lambda_Q + \lambda_0)| &\geq |B_d(pd - 1 + d/(3\beta))| \\
&= |B_d(d/2 + d/(3\beta) - (1/2 - p)d - 1)| \\
&\geq |B_d(d/2 + d/(3\beta) - d/(13\beta) - 1)| \\
&\geq |B_d(d/2 + d/(4\beta))|.
\end{aligned}$$

The first inequality above follows from the fact that $\lambda_0 = \lambda/\beta > d/(3\beta)$, and the second follows from inequality 5.1. So the number of good points for Q is at most:

$$\begin{aligned}
2^d - |B_d(d/2 + d/(4\beta))| &\leq |B_d(d/2 - d/(4\beta))| \\
&\leq e^{-d/(32\beta^2)} 2^d \\
&< 2^{d-d/(32\beta^2)}.
\end{aligned}$$

The second inequality above follows from Fact 5.2.3. Since each good database has n good points, it is immediate that the number of good databases for Q is less than $2^{nd-nd/(32\beta^2)}$. This completes the proof of Claim 5.3.4. $\square$

Combining Claim 5.3.3, Claim 5.3.4, Lemma 5.2.6 we then prove Lemma 5.3.2. $\square$

In view of Fact 5.3.1, we have the same lower bound for the asymmetric communication complexity of β-ANNS. Finally by applying Lemma 5.2.5 we then establish Theorem 5.1.1.

5.4 Concluding Remarks

We have proven a near-optimal deterministic query time lower bound for ANNS in the Hamming cube. Along with the recent randomized lower bound of Chakrabarti and Regev [34], this settles down the status of ANNS in the cell probe model completely.

All known lower bounds for NNS and ANNS (and other related problems such as "partial match") in the cell probe model are proved using tools from communication complexity. As a result, any such lower bound cannot exceed $\Omega(d/\log n)$ for the reason below. In the communication complexity interpretation, Alice holds the query point q and Bob holds the database P. The goal is to let Alice learn the nearest neighbor (or approximate nearest neighbor) of q in P. Since the data structure has size $n^{O(1)}$, each access to its memory cells is equivalent to Alice sending $O(\log n)$ bits (the index of a cell) to Bob. If we could show that Alice has to send at least k bits to Bob to solve the problem, then we obtain a $\Omega(k/\log n)$ lower bound on the query time. However, since obviously $k \leq d$ (Alice can send q to Bob), $\Omega(d/\log n)$ is the best we can hope for using the communication complexity approach. One possible solution for this problem is to consider other models that are less general than the cell probe model but more suitable for NNS and ANNS. Recently, a step in this direction was made by Beame and Vee [24]. They proved lower bounds for NNS in the *branching programs* model of computation. Their lower bounds[4] are slightly better than the $\Omega(d/\log n)$ bound in the cell probe model [21].

Eventually, we would like to see the "curse of dimensionality" conjecture for NNS being solved. In the Hamming cube C_d, it states that (assuming $d \in n^{o(1)} \cap \omega(\log n)$) any data structure with query time $d^{O(1)}$ must use space $n^{\omega(1)}$.

[4]They proved several different lower bounds depending on the underlying metric space: $\Omega(d)$, $\Omega(d\sqrt{\log d/\log\log d})$, or $\Omega(d \log d)$.

Chapter 6

Future Directions

We have described our work in sublinear geometric algorithms and geometric lower bounds. We have also mentioned some specific open problems in these areas throughout this thesis. Finally, we conclude with a few possible directions of future research. These directions are not meant to be representative. They are chosen based on personal preference.

6.1 Sublinear Algorithms

Recall that in the streaming model (Section 1.1.1), data comes as a continuous stream and the algorithm does not have enough space to archive the whole input. Early streaming algorithms focused on computing simple statistics of the input such as its frequency moments. Recently, researchers considered interesting geometric problems in the streaming model. Among other results, efficient approximation algorithms are now known in the streaming model for range counting [129, 18], geometric optimization [36], and dynamic geometric problems [100]. One possible research direction is to identify more geometric problems amenable to the streaming model, and design efficient streaming algorithms for them.

Property testing is a well-developed field now and efficient property testers are known for a wide range of problems. However, a general and unifying theory seems to be lacking here. For example, for graph properties in the dense graph model (i.e., the adjacency matrix representation of graphs), property testing can be either very efficient or impossible: Recall that in time $O(1/\varepsilon^2)$ we can test the (NP-hard) property of "3-colorability" for a graph (Section 1.1.1); on the other hand, it was shown that there exists a monotone graph property in NP which does not admit of any sublinear property tester [91]. Therefore, it is very interesting (and challenging) to obtain some "structural" results regarding easily testable properties in the dense graph model. There has been some progress towards this goal in the last few years. For example, in [91] it was shown that for every graph property whose query complexity is independent of the size of the graph, it can be so tested by uniformly selecting a subset of vertices and checking the induced subgraph for some fixed graph property (which is not necessarily the same as the one being tested). In [10], it was shown that all first order graph properties of type '$\exists\forall$' are testable with a number of queries independent of the size of the graph; on the other hand, there exists a '$\forall\exists$' property that is not thus testable. In [59], a framework for analyzing (one-sided error) property testing algorithms was given. In that paper, the authors defined a class of problems called *abstract combinatorial programs* and proved that a property admits of an efficient tester if its testing can be reduced to an abstract combinatorial program of small dimension. In spite of all these progress, however, we are still far from having a general principle for classifying all easily testable graph properties in the dense graph model.

A weakness of property testing is that it does not deliver any information on the true distance of the object to the property being tested, because a property tester could say “no” for every object that does not have the property. Very recently, people studied a generalization of property testing [8, 123]: the new property tester (called a *tolerant* property tester) computes an estimation of the distance of the object to the property, and so it provides more information than a standard property tester. Efficient tolerant property testers are given for testing monotonicity of functions as well as clustering problems [8, 123]. One research direction is to design tolerant testers for all sorts of properties testable in standard property testing. This immediately brings in a list of open problems.

In algorithm design and analysis, it is usually the case that more efficient algorithms exist for data sets with some structural properties. For example, collision detection of convex polytopes is much easier than general nonconvex polytopes. On the other hand, those algorithms rely heavily on the expected property and may break down completely even when the data set violates the property slightly (e.g., due to bursty noise or data processing errors). To make the computation robust, it is desirable to insert a filter between the algorithm and the original input such that the algorithm is always provided with some “filtered” input that has the expected property. Of course, the filtered input should differ as little as possible from the original input. In an effort to formalize this idea, an on-line variant of property testing (called on-line property-preserving data reconstruction) was proposed in [9]. There, an unknown fraction of the data is assumed to be in violation with the expected property, and we want to design efficient filters that answer data probing queries on-line when no preprocessing is done on the data set. In [9], this new algorithmic paradigm was investigated in the context of maintaining monotone functions. In future work, we could study on-line property-preserving data reconstruction for more challenging tasks such as maintaining convex polytopes in $\mathbf{R}^3$.

6.2 Geometric Lower Bounds

Many challenging problems in geometric lower bounds await answers. For example, we can study on-line and off-line range searching in the semigroup and group arithmetic models. This gives us four versions of range searching. While lower bounds are known for three versions, no results exist for on-line range searching in the group model. It is therefore important to fill up this gap. We expect to adapt the techniques in [42] to prove nontrivial space-time tradeoff lower bounds for on-line range searching in the group model.

As a special case of range searching, halfspace range searching is arguably the only case that is still largely open in the semigroup model and the pointer machine model. The main reason is that the core technique used in lower bound proofs of other range searching problems, namely the construction of a set of points and ranges such that their incidence relationship bipartite graph does not contain large complete bipartite subgraphs, does not apply to halfspace queries. In fact, in halfspace range searching such large complete bipartite subgraphs do exist. In [30], this problem was avoided by a clever weighting strategy on the graph edges. However, the bound thus obtained is only sub-optimal. The situation is even worse for halfspace range reporting, no lower bound has been established in the pointer machine model (or in any other model). For this reason, we single out halfspace range searching as a future research direction in geometric lower bounds.

In Chapter 5, we proved a nearly optimal query time lower bound for approximate nearest neighbor searching (ANNS) in the cell probe model. As pointed out in Section 5.4, lower bounds proved in the cell probe model for ANNS are inherently low. The same is true for nearest neighbor searching (NNS) lower bounds. In order to go around this difficulty, we think that it is necessary to consider specific models for ANNS and NNS.[1]

Given n points in $\mathbf{R}^d$, there are several algorithms [3] that solve NNS with (roughly) $O(m)$ space and $\tilde{O}(n/m^{1/\lfloor d/2 \rfloor})$ query time, for any $n \leq m \leq n^{\lfloor d/2 \rfloor}$. All these algorithms are based on geometric divide-and-conquer. In a nutshell, these algorithms divide the space (primal or dual) into several regions, identify a region relevant to the query point, and solve NNS recursively for data points in that region. This suggests that we could model these algorithms as partition graphs [78]: the space is the size of the whole graph and the query time is the size of the largest subgraph visited over all possible queries. It would be great to prove a strong (or even matching) space-time tradeoff lower bound in this model.

Returning to ANNS, given n points in $\mathbf{R}^d$ and $\varepsilon > 0$, a couple of algorithms compute an $(1+\varepsilon)$-approximate nearest neighbor in time $((d \log n)/\epsilon)^{O(1)}$ with high probability [101, 104]. They use space $d^{O(1)} n^{O(1/\epsilon^2)}$. These algorithms are based on dimensionality reduction techniques such as random linear projections. To get a better lower bound for ANNS (than what we could prove in the cell probe model), we can define a new model that captures the essential features of dimensionality reduction. Once we have such a model, we can also use it to prove space lower bounds for ANNS. Note that for a fixed ε the space used by the aforementioned algorithms is polynomial, but the exponent is quadratic on $1/\varepsilon$. Is this quadratic dependence necessary? Or, could we improve the exponent to $O(1/\varepsilon)$? A good space lower bound in a model for dimensionality reduction might answer this question.

[1]Of course, such models must be general enough to describe all existing algorithms.

Bibliography

[1] D. Achlioptas and F. McSherry. Fast computation of low rank matrix approximations. In *Proc. 33rd Annual ACM Symposium on the Theory of Computing*, pages 611–618, 2001.

[2] P. Agarwal. Range searching. In J. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd ed.* Chapman and Hall, CRC, 2004.

[3] P. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. Goodman, and R. Pollack, editors, *Contemporary Mathematics: Advances in Discrete and Computational Geometry*, volume 223. Amer. Math. Soc., 1999.

[4] P. Agarwal, S. Har-Peled, and M. Karia. Computing approximate shortest paths on convex polytopes. *Algorithmica*, 33:227–242, 1999.

[5] P. Agarwal, S. Har-Peled, M. Sharir, and K. Varadarajan. Approximating shortest paths on a convex polytope in three dimensions. *Journal of the ACM*, 44:567–584, 1997.

[6] P. Agarwal and J. Matoušek. Ray shooting and parametric search. In *Proc. 24th Annual ACM Symposium on the Theory of Computing*, pages 517–526, 1992.

[7] N. Ailon and B. Chazelle. Lower bounds for linear degeneracy testing. In *Proc. 36th Annual ACM Symposium on the Theory of Computing*, pages 554–560, 2004.

[8] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Estimating the distance to a monotone function. In *Proc. 8th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 229–236, 2004.

[9] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Property-preserving data reconstruction. In *Proc. 15th International Symposium on Algorithms and Computation*, 2004.

[10] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica*, 20:451–476, 2000.

[11] N. Alon and M. Krivelevich. Testing k-colorability. *SIAM Journal on Discrete Mathematics*, 15:211–227, 2002.

[12] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. 28th Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996.

[13] N. Alon and J. Spencer. *The Probabilistic Method*. Wiley-Interscience, 2000.

[14] L. Arge, D. Vengroff, and J. Vitter. External-memory algorithms for processing line segments in geographic information systems. In *Proc. 3rd Annual European Symposium on Algorithms*, pages 295–310, 1995.

[15] S. Arya and D. Mount. Approximate nearest neighbor searching. In *Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 271–280, 1993.

[16] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. In *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.

[17] M. Atallah, R. Cole, and M. Goodrich. Cascading divide-and-conquer: A technique for designing parallel algorithms. *SIAM Journal on Computing*, 18:499–532, 1989.

[18] A. Bagchi, A. Chaudhary, D. Eppstein, and M. Goodrich. Deterministic sampling and range counting in geometric data streams. In *Proc. 20th Annual ACM Symposium on Computational Geometry*, pages 144–151, 2004.

[19] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proc. 6th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1–10, 2002.

[20] G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Journal of Algorithms*, 38:91–109, 2001.

[21] O. Barkol and Y. Rabani. Tighter lower bounds for nearest neighbor search and related problems in the cell probe model. In *Proc. 32nd Annual ACM Symposium on the Theory of Computing*, pages 388–396, 2000.

[22] T. Batu, S. Dasgupta, R. Kumar, and R. Rubinfeld. The complexity of approximating the entropy. In *Proc. 34th Annual ACM Symposium on the Theory of Computing*, pages 678–687, 2002.

[23] T. Batu, F. Ergun, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami. A sublinear algorithm for weakly approximating edit distance. In *Proc. 35th Annual ACM Symposium on the Theory of Computing*, pages 316–324, 2003.

[24] P. Beame and E. Vee. Time-space tradeoffs, multiparty communication complexity, and nearest-neighbor problems. In *Proc. 34th Annual ACM Symposium on the Theory of Computing*, pages 688–697, 2002.

[25] J. Beis and D. Lowe. Shape indexing using approximate nearest-neighbor search in high-dimensional spaces. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 1000–1006, 1997.

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

[27] E. Bertino, B. Catania, and B. Shidlovsky. Towards optimal indexing for segment databases. *Technical Report, University of Milano, Italy*, 1998.

[28] A. Bjorner, L. Lovász, and A. Yao. Linear decision trees: Volume estimates and topological bounds. In *Proc. 24th Annual ACM Symposium on the Theory of Computing*, pages 170–177, 1992.

[29] A. Borodin, R. Ostrovsky, and Y. Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *Proc. 31st Annual ACM Symposium on the Theory of Computing*, pages 312–321, 1999.

[30] H. Bronnimann, B. Chazelle, and J. Pach. How hard is halfspace range searching. *Discrete and Computational Geometry*, 10:143–155, 1993.

[31] M. Buddhikot, S. Suri, and M. Waldvogel. Fast layer-4 packet classification using space decomposition techniques. In *Proc. Protocols for High Speed Networks*, 1999.

[32] A. Chakrabarti. *Limitations of non-uniform computational models*. Ph.D. thesis, Computer Science Department, Princeton University, 2002.

[33] A. Chakrabarti, B. Chazelle, B. Gum, and A. Lvov. A lower bound on the complexity of approximate nearest-neighbor searching on the hamming cube. In *Proc. 31st Annual ACM Symposium on the Theory of Computing*, pages 305–311, 1999.

[34] A. Chakrabarti and O. Regev. An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 473–482, 2004.

[35] T. Chan. Approximate nearest neighbor queries revisited. In *Proc. 13th Annual ACM Symposium on Computational Geometry*, pages 352–358, 1997.

[36] T. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. In *Proc. 20th Annual ACM Symposium on Computational Geometry*, pages 152–159, 2004.

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

[38] B. Chazelle. Lower bounds on the complexity of polytope range searching. *Journal of the American Mathematical Society*, 2:637–666, 1989.

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

[40] B. Chazelle. Lower bounds for orthogonal range searching: Ii. the arithmetic model. *Journal of the ACM*, 37:439–463, 1990.

[41] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM Journal on Computing*, 21:671–696, 1992.

[42] B. Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press, Cambridge, 2000.

[43] B. Chazelle. The power of nonmonotonicity in geometric searching. *Discrete and Computational Geometry*, 31:3–16, 2004.

[44] B. Chazelle and D. Dobkin. Intersection of convex objects in two and three dimensions. *Journal of the ACM*, 34:1–27, 1987.

[45] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12:54–68, 1994.

[46] B. Chazelle and L. Guibas. Fractional cascading: I. a data structuring technique, ii. applications. *Algorithmica*, 1:133–191, 1986.

[47] B. Chazelle and D. Liu. Lower bounds for intersection searching and fractional cascading in higher dimension. *Journal of Computer and System Sciences*, 68:269–284, 2004.

[48] B. Chazelle, D. Liu, and A. Magen. Sublinear geometric algorithms. In *Proc. 35th Annual ACM Symposium on the Theory of Computing*, pages 531–540, 2003.

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

[50] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. In *Proc. 28th International Colloquium on Automata, Languages and Programming*, pages 190–200, 2001.

[51] J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proc. 6th Annual ACM Symposium on Computational Geometry*, pages 360–369, 1990.

[52] K. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17:830–847, 1988.

[53] K. Clarkson. An algorithm for approximate closest-point queries. In *Proc. 10th Annual ACM Symposium on Computational Geometry*, pages 160–164, 1994.

[54] K. Clarkson and P. Shor. Applications of random sampling in computational geometry, ii. *Discrete and Computational Geometry*, 4:387–421, 1989.

[55] S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–67, 1993.

[56] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[57] A. Czumaj, F. Ergun, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler. Sublinear-time approximation of euclidean minimum spanning tree. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 813–822, 2003.

[58] A. Czumaj and C. Sohler. Property testing with geometric queries. In *Proc. 9th Annual European Symposium on Algorithms*, pages 266–277, 2001.

[59] A. Czumaj and C. Sohler. Abstract combinatorial programs and efficient property testers. In *Proc. 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 83–92, 2002.

[60] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. In *Proc. 36th Annual ACM Symposium on the Theory of Computing*, pages 175–183, 2004.

[61] A. Czumaj and C. Sohler. Sublinear-time approximation for clustering via random sampling. In *Proc. 31st International Colloquium on Automata, Languages and Programming*, pages 396–407, 2004.

[62] A. Czumaj, C. Sohler, and M. Ziegler. Property testing in computational geometry. In *Proc. 8th Annual European Symposium on Algorithms*, pages 155–166, 2000.

[63] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry*. Springer-Verlag, 1997.

[64] F. Dehne, A. Ferreira, and A. Rau-Chaplin. Parallel fractional cascading on hypercubemultiprocessors. *Computational Geometry: Theory and Applications*, 2:141–167, 1992.

[65] L. Devroye, E. Mucke, and B. Zhu. A note on point location in delaunay triangulations of random points. *Algorithmica*, 22:477–482, 1998.

[66] L. Devroye and T. Wagner. Nearest neighbor methods in discrimination. In P. Krishnaiah and L. Kanal, editors, *Handbook of Statistics*, volume 2. North Holland, 1982.

[67] M. Dietzfelbinger. Lower bounds for sorting of sums. *Theoretical Computer Science*, 66:137–155, 1989.

[68] D. Dobkin and D. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *Journal of Algorithms*, 6:381–392, 1985.

[69] D. Dobkin and D. Kirkpatrick. Determining the separation of preprocessed polyhedra – a unified approach. In *Proc. 17th International Colloquium on Automata, Languages and Programming*, pages 400–413, 1990.

[70] D. Dobkin and R. Lipton. Multidimensional search problems. *SIAM Journal on Computing*, 5:181–186, 1976.

[71] P. Drineas and R. Kannan. Fast monte carlo algorithms for approximate matrix multiplication. In *Proc. 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 452–459, 2001.

[72] R. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approx. Theory*, 10:227–236, 1974.

[73] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[74] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15:341–363, 1986.

[75] H. Edelsbrunner, R. Seidel, and M. Sharir. On the zone theorem for hyperplane arrangements. *SIAM Journal on Computing*, 22:418–429, 1993.

[76] D. Eppstein. Dynamic euclidean minimum spanning trees and extrema of binary functions. *Discrete and Computational Geometry*, 13:111–122, 1995.

[77] F. Ergun, S. Kannan, R. S. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. In *Proc. 30th Annual ACM Symposium on the Theory of Computing*, pages 259–268, 1998.

[78] J. Erickson. *Lower bounds for fundamental geometric problems*. Ph.D. thesis, Computer Science Division, University of California at Berkeley, 1996.

[79] J. Erickson. Lower bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science*, 8, 1999.

[80] J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. *Discrete and Computational Geometry*, 13:41–57, 1995.

[81] R. Fagin. Fuzzy queries in multimedia database systems. In *Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–10, 1998.

[82] E. Fischer. The art of uninformed decisions: A primer to property testing. In *The Computational Complexity Column*, volume 75. The Bulletin of the European Association for Theoretical Computer Science, 2001.

[83] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dorn, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the qbic system. *IEEE Computer*, 28:23–32, 1995.

[84] M. Fredman. How good is the information theory bound in sorting. *Theoretical Computer Science*, 1:355–361, 1976.

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

[86] M. Fredman. Lower bounds on the complexity of some optimal data structures. *SIAM Journal on Computing*, 10:1–10, 1981.

[87] A. Frieze and R. Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19:175–220, 1999.

[88] A. Gajentaan and M. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, 5:165–185, 1995.

[89] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of Np-Completeness*. W.H. Freeman & Co., 1979.

[90] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45:653–750, 1998.

[91] O. Goldreich and L. Trevisan. Three theorems regarding testing graph properties. *Random Structures and Algorithms*, 23:23–57, 2003.

[92] M. Goodrich. *Efficient parallel techniques for computational geometry*. Ph.D. thesis, Dept. Comput. Sci., Purdue Univ., 1987.

[93] M. Grotschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin, 1988.

[94] S. Har-Peled. Approximate shortest-path and geodesic diameter on convex polytopes in three dimensions. *Discrete and Computational Geometry*, 21:217–231, 1999.

[95] L. Harper. Optimal numberings and isoperimetric problems on graphs. *Journal of Combinatorial Theory*, 1:385–394, 1966.

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

[97] J. Hershberger and S. Suri. Practical methods for approximating shortest paths on a convex polytope in r^3. *Computational Geometry: Theory and Applications*, 10:31–46, 1998.

[98] P. Indyk. Sublinear-time algorithms for metric space problems. In *Proc. 31st Annual ACM Symposium on the Theory of Computing*, pages 428–434, 1999.

[99] P. Indyk. A sublinear-time approximation scheme for clustering in metric spaces. In *Proc. 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 154–159, 1999.

[100] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proc. 36th Annual ACM Symposium on the Theory of Computing*, pages 373–380, 2004.

[101] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annual ACM Symposium on the Theory of Computing*, pages 604–613, 1998.

[102] S. Kapoor. Efficient computation of geodesic shortest paths. In *Proc. 31st Annual ACM Symposium on the Theory of Computing*, pages 770–779, 1999.

[103] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proc. 29th Annual ACM Symposium on the Theory of Computing*, pages 599–608, 1997.

[104] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30:457–474, 2000.

[105] T. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proc. ACM SIGCOMM*, pages 191–202, 1998.

[106] D. Liu. A strong lower bound for approximate nearest neighbor searching in the cell probe model. *Information Processing Letters*, 92:23–29, 2004.

[107] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete and Computational Geometry*, 10:157–182, 1993.

[108] J. Matoušek. Geometric range searching. In *ACM Computing Survey*, volume 26. 1994.

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

[110] K. Mehlhorn and S. Naher. Dynamic fractional cascading. *Algorithmica*, 5:215–241, 1990.

[111] K. Mehlhorn, S. Naher, and H. Alt. A lower bound on the complexity of the union-split-find problem. *SIAM Journal on Computing*, 17:1093–1102, 1988.

[112] K. Mehlhorn, S. Naher, T. Schilz, S. Schirra, M. Seel, R. Seidel, and C. Uhrig. Checking geometric programs or verification of geometric structures. *Computational Geometry: Theory and Applications*, 12:85–103, 1999.

[113] G. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32:265–279, 1986.

[114] P. Miltersen. Lower bounds for union-split-find related problems on random access machines. In *Proc. 26th Annual ACM Symposium on the Theory of Computing*, pages 625–634, 1994.

[115] P. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57:37–49, 1998.

[116] N. Mishra, D. Oblinger, and L. Pitt. Sublinear time approximate clustering. In *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 439–447, 2001.

[117] J. Mitchell. An algorithmic approach to some problems in terrain nevigation. In *Autonomous Mobile Robots: Perception, Mapping and Nevigation*. IEEEE computer society press, Los Alamitos, CA, 1991.

[118] J. Mitchell, D. Mount, and C. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16:647–668, 1987.

[119] Y. Morimoto, T. Fukuda, S. Morishita, and T. Tokuyama. Implementation and evaluation of decision trees with range and region splitting. *Constraint*, 2:163–189, 1997.

[120] E. Mucke, I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two and three-dimensional delaunay triangulations. In *Proc. 12th Annual ACM Symposium on Computational Geometry*, pages 274–283, 1996.

[121] K. Mulmuley. Output sensitive and dynamic constructions of higher order voronoi diagrams and levels in arrangements. *Journal of Computer and Systems Sciences*, 47:437–458, 1993.

[122] S. Muthukrishnan. Data streams: Algorithms and applications. *Preprint*, 2003.

[123] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. *ECCC Report TR04-010*, 2004.

[124] A. Pogorelov. *Extrinsic geometry of convex surfaces*, volume 35. American Mathematical Society, Providence, RI, 1973.

[125] D. Ron. Property testing. In S. Rajasekaran, P. Pardalos, J. Reif, and J. Rolim, editors, *Handbook on Randomization*, volume II. 2001.

[126] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete and Computational Geometry*, 6:423–434, 1991.

[127] S. Sen. Fractional cascading revisited. *Journal of Algorithms*, 19:161–172, 1995.

[128] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM Journal on Computing*, 15:193–215, 1986.

[129] S. Suri, C. Tóth, and Y. Zhou. Range counting over multidimensional data streams. In *Proc. 20th Annual ACM Symposium on Computational Geometry*, pages 160–169, 2004.

[130] R. Tamassia and J. Vitter. Optimal cooperative search in fractional cascaded data structures. *Algorithmica*, 15, 1996.

[131] R. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and Systems Sciences*, 18:110–127, 1979.

[132] P. Vaidya. Space-time tradeoffs for orthogonal range queries. *SIAM Journal on Computing*, 18:748–758, 1989.

[133] K. Varadarajan and P. Agarwal. Approximating shortest paths on a nonconvex polyhedron. *SIAM Journal on Computing*, 30:1321–1340, 2000.

[134] A. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 18th Annual IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1977.

[135] A. Yao. A lower bound to finding convex hulls. *Journal of the ACM*, 28:780–787, 1981.

[136] A. Yao. Should tables be sorted. *Journal of the ACM*, 28:615–628, 1981.

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

[138] A. Yao. Decision tree complexity and betti numbers. In *Proc. 26th Annual ACM Symposium on the Theory of Computing*, pages 615–624, 1994.

[139] A. Yao. Algebraic decision trees and euler characteristics. *Theoretical Computer Science*, 141:133–150, 1995.

[140] A. Yao and F. Yao. A general approach to d-dimension geometric queries. In *Proc. 17th Annual ACM Symposium on the Theory of Computing*, pages 163–168, 1985.