ON THE SEARCH FOR GEOMETRIC ORDERS, CENTERS, AND SEPARATION

BY

MITCHELL JONES

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Doctoral Committee:

Professor Sariel Har-Peled, Chair
Professor Timothy Chan
Professor Chandra Chekuri
Professor Kasturi Varadarajan, University of Iowa

Abstract

Over the previous decade, there has been an explosion in the amount of data that needs to be stored, processed, and queried efficiently. Arguably, this is due to the large improvements in data collection methods and machine learning algorithms. A large portion of the data is naturally geometric, consisting of point sets or other simple geometric objects. Examples of operations one may want to perform on finite point sets include ordering the points for storage, sorting, and searching, computing statistical summaries of the points, or breaking the points into smaller clusters for further processing. In this thesis, we study many of the aforementioned problems from a theoretical perspective.

In part one we develop a new technique called locality-sensitive orderings. Given a finite point set $P \subseteq [0,1)^d$, we describe a collection of orderings (embeddings of P onto an interval on the real line) which have the property that for any two points in $p, q \in [0,1)^d$, there is an ordering in which all points between p and q according to the ordering are "close" to either p or q in the original space. Locality-sensitive orderings leads to surprisingly simple data structures for a variety of low-dimensional proximity based problems in computational geometry.

In the second part of this thesis, we examine various ways to define the center of a point set, and develop efficient algorithms for computing these centers in the process. We develop a new randomized algorithm for computing the approximate centerpoint of a point set. Next, we develop new exact algorithms for finding the yolk of a point set, whose motivation and definition rises from ideas in voting theory. We explore the connection between centerpoints and weak ε-nets by presenting some new alternatives to weak ε-nets.

In the third part, we investigate the notion of separation in computational geometry. In particular, we develop a new approximation algorithm for computing the minimum number of lines needed to separate all pairs of a given planar point set. Afterwards, we study the problem of active-learning a concept class which is a convex body. We develop new learning algorithms which assume the computational model has access to an efficient separation oracle for the given convex body.

To my parents, Lee and Peter, for their unconditional love and endless support.

Acknowledgments

First and foremost, I thank my advisor and mentor Sariel Har-Peled. Without him this would not have been possible. I have appreciated his guidance and constant encouragment through this degree, as well as his sharp insight for solving new problems. Almost all of my intuition about computational geometry and algorithms I have learnt from him. I'd also like to thank Timothy Chan, Chandra Chekuri, and Matus Telgarsky for their advice, mentorship, and research discussions throughout the years. Finally, I thank Kasturi Varadarajan for serving as a committee member during these unprecedented times.

In August of 2019, I visited the University of Sydney to work with Joachim Gudmundsson and his students. I thank both Joachim Gudmundsson and Julián Mestre for their continuous support and mentorship over the years as I pursued my graduate studies (I particularly thank Julián Mestre, who encouraged me to apply to the University of Illinois Urbana-Champaign). Thanks to Sampson Wong for various stimulating research discussions while I was visiting.

Working through the Ph.D. program has been no easy task, and I am forever thankful for the many friends and colleagues I have met within the research group. In particular, I'd like to thank Robert Andrews, Shant Boodaghians, Vasileios Livanos, Sahand Mozaffari, and Xilin Yu. It has always been great to discuss research, grad student life, and socialize with them. Only they know how challenging the Ph.D. process can be, and I'm glad to have met such amazing people. I'd also like to thank Hsien-Chih Chang, Kent Quanrud, and Chao Xu, who all recently graduated, for their advice and mentorship in the early years of my Ph.D. Their perspective as a student helped me prepare mentally for the later years of the program.

Moving countries was a big step for me, and it was hard to say goodbye to my friends from Australia. It has been great to keep in touch with them over the years, making sure to see them anytime I returned to the country. In particular I'd like to thank Shu Bor, Mitch Della Marta, Linus Karsai, Patti Lor, Angela Mariani, Eve Martin-Jones, Reuben Moorhouse, James Phillips, and Kayla Stevens for their companionship and support.

Lastly—but certainly not least—I'd like to thank my family, Lee, Peter, Chris, and my wife Abigail, for their constant support and love throughout the years. Finishing my Ph.D. during the pandemic has not been an easy time, part of which is because it is harder to visit home. I have been extremely lucky to have the love and support of my family at my side, I am forever thankful that they have been always there to provide the encouragement and additional motivation I needed to complete this thesis.

Contents

1 Introduction

> *Geometry is an art of making right conclusions from badly drawn pictures.*
>
> **— Niels H. Abel**

This thesis focuses on developing efficient approximation and randomized algorithms for problems with a geometric flavor. Many problems in practice deal with discrete geometric objects (such as points, lines, and disks), and it is of interest to obtain the most efficient algorithm possible. Since the inception of the field approximately fifty years ago, the field has had far reaching applications in machine learning and probability [10, 88, 120], robotics (such as motion planning [103]), and more generally problems dealing with large amounts of geometric data (for example, the development of geographic information systems). See also the discussion by de Berg et al. [18, Section 1.3]. Many of the results in these areas typically follow from exploiting the underlying geometric nature of the problem, which so often happen to contain rich mathematical structure. It is for this reason that development of geometric algorithms requires tools from many areas of computer science and mathematics.

The topics and problems studied in this thesis can be classified into three main parts:

1. Geometric orderings. Given a high dimensional point set, how can we order the points appropriately so that certain queries and operations can be performed efficiently? For example, it is natural to want to sort, store, or search through the point set.

2. Geometric centers. Here, we are interested in identifying different objects (of constant description complexity) which summarize a given point set. Ideally, we want these summaries to be robust to outliers and efficiently computable.

3. Geometric separation. Given a point set P, we investigate how lines can be used to split P into smaller clusters. Such an operation is useful for divide and conquer algorithms and has applications in machine learning. Afterwards, we propose a new computational model which is augmented with access to a separation oracle for a given convex body. In this model, we develop new learning algorithms where the concept class consists of arbitrary convex bodies.

In the remainder of this chapter, we discuss the exact problems of interest in each part. Throughout, the notation O_d hides constants that depend on the dimension d.

1.1 GEOMETRIC ORDERS

In Part I, we describe a technique that leads to new, simpler algorithms for a number of fundamental proximity problems in low-dimensional Euclidean spaces.

Given data, having an ordering over it is quite useful—it enables one to sort it, store it, and search it efficiently, among other things. Such an order is less natural for points in the plane (or in higher dimensions). One way to impose such orders is by using bijective mappings from the plane to the line (which has a natural order, and thus endows the plane with an order). Such mappings, known as space-filling curves, were discovered in 1890 by Peano [131]. (See also the book by Sagan [140] for more information on space-filling curves.) For computational purposes, the Z-order, a somewhat inferior space-filling curve, is the easiest to implement as it is easily computed by interleaving the bits of the x and y coordinates.

A natural property one desires in an ordering of the plane is that it preserves locality—points that are close together geometrically remain close in the resulting ordering. Unfortunately, no mapping/ordering can have this property universally, as the topology of the line and the plane are fundamentally different. Nevertheless, the Z-order already has some nice locality properties—it maps certain squares to intervals on the real line, and these squares form a grid that covers the unit square. Furthermore, these grids are universal, in the sense that there is a grid for any desired resolution.

To get better locality properties, one has to use more orders. It is known that if one uses three orders in the plane (which is the result of shifting the plane before applying the Z-order), then for any axis parallel square $\mathcal{C}$ inside the unit square[1], there exists a square $\mathcal{C}'$ that contains $\mathcal{C}$, such that $\mathcal{C}'$ is only slightly bigger than $\mathcal{C}$, and one of the three orders maps $\mathcal{C}'$ to an interval.

Our purpose here is to get an even stronger locality property, which requires a larger collection of orderings. Specifically, consider two points $p, p' \in [0,1]^2$. The desired property is that there are two squares $\mathcal{C}$ and $\mathcal{C}'$, and an order σ in the collection, with the following properties: (i) $p \in \mathcal{C}$ and $p' \in \mathcal{C}'$, (ii) the diameters of $\mathcal{C}$ and $\mathcal{C}'$ are only an ε-fraction of the distance between p and p', (iii) $\mathcal{C}$ and $\mathcal{C}'$ are mapped to two intervals on the real line by σ, and (iv) these two intervals are adjacent. Such an ordering σ with the desired properties is illustrated in Figure 1.1.

For algorithmic applications, this collection of orders needs to be small and easily computable. Surprisingly, we show that the desired collection of orders has size that depends only on ε (and d in general), and these orders can be easily computed. The result is in Section 2.2.

To see why having such a collection of orders is so useful, consider the problem of computing the closest pair of points in a given set of points P. Every order in the collection induces an ordering of P. Furthermore, the closest pair of points are going to be adjacent in one of these orders, and as such can be readily computed by considering all consecutive pairs of points in the ordering (the number of such pairs is linear). Furthermore, using balanced binary search trees, it is easy to maintain each ordered set under

[1] Throughout, we assume all data lies within the unit hypercube $[0,1]^d$. This can be achieved without loss of generality by scaling or rotating the data appropriately.

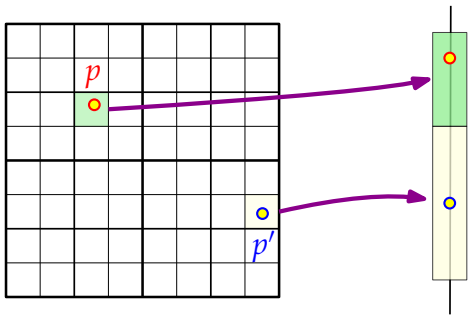


Figure 1.1: Embedding a point set on the real line.

insertions and deletions. Therefore, one can maintain the closest pair of points by storing P in such a data structure for each of the orderings. As a result, a dynamic problem that in advance might seem somewhat challenging reduces (essentially) to the mundane task of maintaining ordered sets under insertions and deletions. Additional applications of this new technique are described in Section 2.3.

1.2 GEOMETRIC CENTERS

For a given point set P in $\mathbb{R}^d$, it is natural to ask for a "summary" of P which is of small size. Such summaries could include the centroid (center of mass of P) or a collection of points which preserves some desired properties (for example, ε-nets [88] or coresets [5]). These sets of small size which capture some property of P are useful for many geometric approximation algorithms.

In Part II, our main focus is developing summaries that are a single point or simple geometric object, while also being robust to outliers (the centerpoint and the yolk of a point set). In both cases, we are interested in developing efficient algorithms to compute these quantities.

1.2.1 Centerpoints

Given a point set $P \subset \mathbb{R}^d$ and a parameter $\alpha \in (0,1)$, a point $c \in \mathbb{R}^d$ is an ***α-centerpoint*** if all closed halfspaces containing c also contain at least αn points of P. A classical implication of Helly's theorem is that a $1/(d+1)$-centerpoint always exists for any point set in $\mathbb{R}^d$ [112]. In particular, a centerpoint generalizes the definition of a median of a one dimensional data set to higher dimensions. For this reason, centerpoints are useful for divide and conquer algorithms in computational geometry.

It is currently unknown if one can compute a $\Omega(1/(d+1))$-centerpoint in polynomial time (in the dimension). The best algorithm currently known is by Chan [33], which computes a $1/(d+1)$-centerpoint in expected time $O_d(n^{d-1} + n \log n)$. For this reason, past focus has been on efficiently computing centerpoints of slightly worse quality. The first (randomized) polynomial time algorithm was presented by Clarkson et al. [49], in which they compute a $1/(4d^2)$-centerpoint in $\widetilde{O}(d^9)$ time[2], where $\widetilde{O}$ hides polylogarithmic factors

[2]There is an implicit linear dependence on n here which is required to read the input. The algorithm itself performs a single random

depending on d.

In Chapter 3, we improve the algorithm of Clarkson et al. [49] for approximating a centerpoint. Specifically, the algorithm (which is a variant of their algorithm) runs in $\widetilde{O}(d^7)$ time, and computes roughly a $1/(d+2)^2$-centerpoint. This improves both the running time, and the quality of centerpoint computed. While the improvements are small (a factor of d^2 roughly in the running time, and a factor of four in the centerpoint quality), we believe that the new algorithm is simpler. This is the first improvement of the randomized algorithm of Clarkson et al. [49] in over twenty years.

1.2.2 Exploring the connection between centerpoints and nets

In Chapter 4 we explore a different way to summarize point sets. Specifically, we study variants of weak ε-nets. From a certain perspective, ε-nets and weak ε-nets can be viewed as a summary of a given point set, responsible for capturing when a convex body is "heavy" with respect to P. Formally, given a set of points P of size n and a collection of ranges $\mathcal{R}$ (where each range is a subset of P), a set $\mathsf{S} \subseteq P$ is an ε-net for P if every range in $\mathcal{R}$ containing at least εn points of P intersects S. A celebrated result of Haussler and Welzl is that any range space of bounded VC dimension δ (Definition 4.3$_{\text{p43}}$) admits an ε-net whose size depends only on ε and δ [88, Theorem 4.1$_{\text{p44}}$]. Now, consider the range space $(P, \mathcal{C})$ where $\mathcal{C}$ is the collection of all compact convex bodies in $\mathbb{R}^d$ and $P \subset \mathbb{R}^d$ is a point set of size n. This range space has infinite VC dimension—the standard ε-net constructions do not work for this range space. The notion of ***weak ε-nets*** bypasses this issue by allowing the net S to use points outside of P. Specifically, any convex body C that contains at least εn points of P must contain a point of S.

When d is arbitrary, the best known construction of weak ε-nets is by Rubin [137, 138], who constructed weak ε-nets of size $O_d(\varepsilon^{-(d-0.5+\alpha)})$ for arbitrarily small $\alpha > 0$ (until recently, the best known bound was $O_d(\varepsilon^{-d} \log^{f(d)} \varepsilon^{-1})$, where $f(d) = O(d^2 \log d)$ by Matoušek and Wagner [114]). We refer the reader to Section 4.1 for additional background, context, and previous work.

Ideally, it would be preferable to avoid the $1/\varepsilon^{O(d)}$ term. Of course, it would be extremely surprising to have nets which completely avoid this type of dependency on the dimension. For this reason, it is interesting to consider other variants of weak ε-nets and different computational models which capture similar properties but can be more easily computed or have a smaller size. To this end, we propose three new types of nets.

First, we suggest what is called a functional net. Here, we assume that each convex body C in the range space is equipped with a ***separation oracle***. Namely, given a query point $q \in \mathbb{R}^d$, the separation oracle either reports that $q \in C$ or returns a hyperplane separating the two objects. In this model, we show that one can precompute a net (which has size polynomial in $1/\varepsilon$ and d) such that given any convex body equipped with a separation oracle, an adaptive sequence of query points can be generated (using only the net) which can

sample of P—whose size depends only on d—and all further computations are done on this random sample. Equivalently, the running time can be stated as $O(n + d^9 \text{polylog}(d))$. See also the discussion surrounding Lemma 3.10$_{\text{p37}}$.

determine if the given body contains at least εn points of P. Concretely, if the body C contains at least εn points of P, then one of the queries will be contained in C. However, the converse does not hold: C may contain less than εn points but still contain a query point (note that the standard weak ε-net definition also has this property). See Section 4.3 for further discussion on the model and result.

Along the way, we also explore the connection between centerpoints and weak ε-nets. In doing so, we define a new type of net known as an (ε, α)-center net of a point set P. Specifically, it is a subset $\mathsf{S} \subseteq \mathbb{R}^d$ such that for any convex body C containing at least εn points of P, S contains a point which is an α-centerpoint of $P \cap C$. This is a strengthening of regular weak ε-nets, as we require the net to stab each convex body C roughly in the center (with respect to the distribution of P inside C). In Section 4.4, we show the existence of an (ε, α)-center net of size $O_d(1/\varepsilon^{d^2})$, where $\alpha = \Omega_d(1/\log(1/\varepsilon))$.

Finally, we consider an extension of weak ε-nets where one allows the sample S to contain other geometric objects. We define a (k, ε)-net to be a collection of k-flats S such that if C is a convex body containing at least εn points of P, then there exists a k-flat in S intersecting C. Note that $(0, \varepsilon)$-nets are exactly weak ε-nets. In Section 4.5, we study an even simpler version of the problem, where the ground set is the hypercube $B = [0,1]^d$. In particular, for $\varepsilon \in (0,1)$ and $0 \leq k < d$, we are interested in computing the smallest set K of k-flats, such that if C is a convex body with $\mathsf{vol}(C \cap B) \geq \varepsilon$, then there is a k-flat in K which intersects C. In this specific setting, rather surprisingly, for $k \geq 1$ we obtain nets of size $O_d(1/\varepsilon^{1-k/d})$, which is *sublinear* in $1/\varepsilon$. We also establish that any such collection of k-flats must have size $\Omega_d(1/\varepsilon^{1-k/d})$, which matches our upper bound up to constants depending on d.

1.2.3 The yolk and related concepts

Suppose there is a collection of n voters in $\mathbb{R}^d$, where each dimension represents a specific ideology. In a fixed dimension, each voter maintains a value along this continuum representing their stance on a given ideology. Here, consider ideologies such as those which have a natural ordering (left/right or high/low). For example, a policy to increase/decrease the budget, a policy in criminal law to propose softer/harder sentences, and other similar social and economic policies. One can interpret $\mathbb{R}^d$ as a ***policy space***, and each point in $\mathbb{R}^d$ represents a single policy. In the Euclidean spatial model, each voter is also represented as a point in $\mathbb{R}^d$ which corresponds to their *ideal policy*. Here, we are interested in the ***proximity model***, in which voters always prefer policies which are closer to their ideal policy under the regular Euclidean norm (see [117] for an introduction to voting theory, and discussions on other spatial models). For two policies $x, y \in \mathbb{R}^d$ and a set of voters $P \subset \mathbb{R}^d$, x ***beats*** y if more voters in P prefer policy x compared to y. A plurality point is a policy which beats all other policies in $\mathbb{R}^d$. For $d = 1$, the plurality point is the median voter (when n is odd) [24]. However for $d > 1$, a plurality point is not always guaranteed to exist [139]. It is known that one can test if a plurality point exists (and if so, compute it) in $O(dn \log n)$ time [21]. Note that the plurality point is a point of Tukey depth $\lceil n/2 \rceil$—in general this is the largest possible Tukey depth any point can

have; while the centerpoint is a point that guarantees a "respectable" minority of size at least $n/(d+1)$.

Since plurality points may not always exist, one generalization of a plurality point is the yolk [115]. A hyperplane is a ***median hyperplane*** if the number of voters lying in each of the two closed halfspaces is at least $\lceil n/2 \rceil$. The ***yolk*** is the ball of smallest radius intersecting all such median hyperplanes. Note that when a plurality point exists, the yolk has radius zero (equivalently, all median hyperplanes intersect at a common point).

The yolk has received considerable attention in the literature. The first polynomial time exact algorithm for computing the yolk in $\mathbb{R}^d$ was by Tovey in $O_d\left(n^{(d+1)^2}\right)$ time—in the plane, the running time can be improved to $O(n^4)$ [150]. Following Tovey, the majority of results have focused on computing the yolk in the plane. In 2018, de Berg et al. [21] gave an $O(n^{4/3}\log^{1+\varepsilon} n)$ time algorithm (for any fixed $\varepsilon > 0$) for computing the yolk. Obtaining a faster exact algorithm remained an open problem. Gudmundsson and Wong [71, 72] presented a $(1+\varepsilon)$-approximation algorithm with $O(n\log^7 n \log^4 \varepsilon^{-1})$ running time.

In Chapter 5 we develop a randomized algorithm for computing the yolk in $O_d(n^{d-1}\log n)$ time. In particular, when $d = 2$, this gives an $O(n\log n)$ algorithm for computing the yolk *exactly*, which improves all previous results [21, 71, 72, 150]. We also explore similarly defined geometric objects (such as the Tukey and center ball of a point set), and give the first efficient algorithms for computing such objects.

1.3 GEOMETRIC SEPARATION

1.3.1 Separating point sets

For a set P of n points in $\mathbb{R}^d$, a set L of hyperplanes ***separates*** P, if for any pair of points of $p, q \in P$, there is a hyperplane in L that intersects the interior of the segment pq (which also does not contain p or q). In $\mathbb{R}^2$, L is a set of lines. The ***separability*** of P, denoted by $\mathrm{sep}(P)$, is the size of the smallest set of hyperplanes that separates P. The separability of a point set captures how grid-like the point set is. In particular, the separability of the $\sqrt{n} \times \sqrt{n}$ grid is $2\sqrt{n} - 2$, while for n points in convex position in the plane the separability is $\lceil n/2 \rceil$ (and this is the worst case assuming general position).

Motivation for studying the separability of a point set follows as separating and breaking point sets, usually into clusters, is a fundamental task in computer science, needed for divide and conquer algorithms. It is thus natural to ask what can be done if restricted to lines, and if one can do the partition in a global fashion. In particular, if we are allowed to partition the point set using the zero set of a polynomial rather than a line, then some results are known [3]. However dealing with polynomials may be less algorithmically convenient to work with. Freimer et al. [68] showed that computing the separability of a given point set in the plane is NP-complete. Other variants of the problem have also been previously studied [25, 29, 54, 64, 100, 127].

In Chapter 6 we develop an efficient approximation algorithm for computing the separability of points

in the plane. Additionally, for a set of n points P chosen uniformly at random from the unit square $[0,1]^2$, we study the random variable $\mathrm{sep}(P)$. We prove that with high probability $\mathrm{sep}(P) = O(n^{2/3})$ and $\mathrm{sep}(P) = \Omega(n^{2/3} \log\log n / \log n)$. This is perhaps surprising, as initially one may expect that the randomly sampled point set forms a grid-like structure, which can be separated with $\Theta(\sqrt{n})$ lines.

1.3.2 Classifying point sets

Suppose we are given a set P of n points in $\mathbb{R}^2$ and access to a convex body C via a ***separation oracle***. Specifically, given a query point $z \in \mathbb{R}^2$, the oracle either reports that z is inside C, or if not, it returns a separating line ℓ such that z lies on one side of ℓ, and C lies on the other. In Chapter 7 we study the problem of deciding for every point $p \in P$ whether or not p is contained in C, using as few oracle queries as possible. This question can be naturally extended to higher dimensions, where separating hyperplanes are used instead of separating lines.

In the worst case, there exist instances for which one needs to query the oracle for *every* input point. It is for this reason that our goal is to develop ***instance sensitive*** algorithms. If the input is structured well, the algorithm makes few queries. As the input deteriorates in quality, the algorithm may revert to the brute force solution. We obtain instance sensitive algorithms for 2D and 3D.

This problem can be interpreted as active-learning a convex body in relation to a set of points P that needs to be classified (as either inside or outside the body), where the queries are via a separation oracle. We are unaware of any work directly on this problem in the computational geometry community, while there are some works in the machine learning community that study related problems (usually in more practical and realistic settings, and in higher dimensions, naturally) [51, 73, 142].

1.4 REMARKS

We close this chapter with some additional remarks.

1.4.1 Computational model

Throughout the thesis, unless otherwise explicitly stated, the model of computational assumed is the unit-cost real RAM model. This model allows arbitrary real numbers to be stored and assumes arithmetic operations and comparisons between real numbers can be performed in constant time. This is the standard model in computational geometry [122, 132, 143].

1.4.2 Low-dimensional computational geometry

In this thesis, the Euclidean dimension d is assumed to be a small, fixed constant (with the single exception of Chapter 3). For this reason, in most places the constant factors involving d are ignored. This assumption is

Paper	Chapter
T. M. Chan, S. Har-Peled, and M. Jones. On locality-sensitive orderings and their applications. SIAM Journal on Computing, 49(3): 583—600, 2020. Copyright 2020 Society for Industrial and Applied Mathematics. Originally appeared at ITCS '19 [38].	Chapter 2
S. Har-Peled and M. Jones. Journey to the center of the point set. ACM Transactions on Algorithms, 17(1), Article 9, 2020. Originally appeared at SoCG '19 [80].	Chapter 3, Chapter 4
S. Har-Peled and M. Jones. Stabbing convex bodies with lines and flats. 2021. To appear at the Symposium of Computational Geometry (SoCG 2021) [82].	Chapter 4
S. Har-Peled and M. Jones. Fast algorithms for geometric consensuses. Symposium on Computational Geometry (SoCG 2020), 50:1–50:16, 2020 [79].	Chapter 5
S. Har-Peled and M. Jones. On separating points by lines. Reprinted by permission from Springer Nature: Springer Nature, Discrete & Computational Geometry, 63(3): 705–730, 2020. Copyright 2020 Springer Nature. Originally appeared at SODA '18 [81].	Chapter 6
S. Har-Peled, M. Jones, and R. Saladi. Active-learning a convex body in low dimensions. First published in Algorithma, 1–33, 2021 by Springer Nature. Reproduced with permission from Springer Nature. Originally appeared at ICALP '20. [84].	Chapter 7

Table 1.1: The papers included and their location within the thesis.

required as many algorithms naturally have hidden factors which are exponential in d (e.g., $2^{O(d)}$, $2^{O(d \log d)}$, or worse) due to the curse of dimensionality. Throughout this thesis, the notation O_d, Ω_d, and Θ_d may be used to remind the reader that the constants depending on d are hidden.

In many of the problems studied in this thesis, it remains interesting to consider the lower dimensional versions of the problem (e.g., the yolk, partitioning points, and various proximity-type problems). As one might expect, the known tools and techniques to solve problems in lower dimensions is much larger and can lead to intriguing results (some of which are developed in this thesis). For problems which sit in a higher dimensional space, typically we assume some additional structural parameters about the input instance. This allows us to develop algorithms where the running time is parametrized by the structure of the instance, which may be much smaller than the ambient dimension.

1.4.3 Roadmap

The remainder of this thesis is split into three parts: geometric orders, centers, and separation. In Chapter 8, we conclude the thesis by discussing future work and listing some open problems.

This thesis is a compilation of six papers, containing much of the same contents and results, with a slightly modified presentation. See Table 1.1 for the citations and appropriate copyright notices.

Part I

Geometric orderings

2 Locality-sensitive orderings

> *We are all faced with a series of great opportunities brilliantly disguised as impossible situations.*
>
> **— Charles Swindoll**

For any constant d and parameter $\varepsilon \in (0, 1/2]$, we show the existence of (roughly) $1/\varepsilon^d$ orderings on the unit cube $[0,1)^d$, such that for any two points $p, q \in [0,1)^d$ close together under the Euclidean metric, there is a linear ordering in which all points between p and q in the ordering are "close" to p or q. More precisely, the only points that could lie between p and q in the ordering are points with Euclidean distance at most $\varepsilon \|p - q\|$ from either p or q. These orderings are extensions of the Z-order, and they can be efficiently computed.

Functionally, the orderings can be thought of as a replacement to quadtrees and related structures (like well-separated pair decompositions). We use such orderings to obtain surprisingly simple algorithms for a number of basic problems in low-dimensional computational geometry, including (i) dynamic approximate bichromatic closest pair, (ii) dynamic spanners, (iii) dynamic approximate minimum spanning trees, (iv) static and dynamic fault-tolerant spanners, and (v) approximate nearest neighbor search.

2.1 BACKGROUND

2.1.1 Quadtrees and Z-order

Consider a point set $P \subseteq [0,1)^2$, its quadtree, and a depth-first search (DFS) traversal of this quadtree. One can order the points of P according to this traversal, resulting in some ordering $\prec$ of the underlying set $[0,1)^2$. The relation $\prec$ is the ordering along some space filling mapping.

One particular ordering of interest is the ***Z-order***. Conceptually speaking, the Z-order can be thought of as a DFS of the quadtree over $[0,1)^2$, where the children of each node in the quadtree are always visited in the same pre-defined order (see Figure 2.1). The Z-order is a total ordering over the points in $[0,1)^2$, and can be formally defined by a bijection z from the unit interval $[0,1)$ to the unit square $[0,1)^2$. Given a real number $\alpha \in [0,1)$, with the binary expansion $\alpha = 0.x_1x_2x_3\ldots$ (i.e., $\alpha = \sum_{i=1}^{\infty} x_i 2^{-i}$), the Z-order mapping

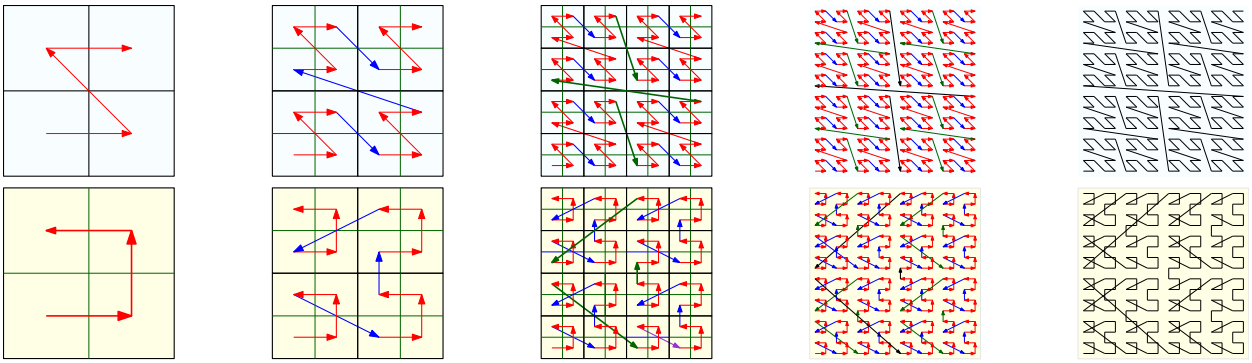


Figure 2.1: Changing the order in which a DFS visits the children of a quadtree node induces a different ordering of the underlying square (and produces different space filling curves). The top row shows the Z-order (or Σ-order), and the bottom row shows the $\supset$-order.

of α is the point $z(\alpha) = (0.x_2x_4x_6\ldots, 0.x_1x_3x_5\ldots)$. We note that the Z-order mapping z is not continuous. Nevertheless, the Z-order mapping has the advantage of being easy to define. In particular, computing the Z-order or its inverse is quite easy, if one is allowed bitwise-logical operations—in particular, the ability to compute compressed quadtrees efficiently is possible only if such operations are available [78]. The approach extends to higher constant dimensions.

The idea of using the Z-order can be traced back to the work of Morton [121], and it is widely used in databases and seems to improve performance in practice [94]. Once comparison by Z-order is available, building a compressed quadtree is no more than storing the points according to the Z-order, and this yields simple data structures for various problems. For example, Liao et al. [106] and Chan [32, 35, 37] applied the Z-order to obtain simple efficient algorithms for approximate nearest neighbor search and related problems.

2.1.2 Shifting

The Z-order (and quadtrees) does not preserve distance. That is, two points that are far away might be mapped to two close-together points, and vice versa. This problem is even apparent when using a grid, where points that are close together get separated into different grid cells. One way to get around this problem is to shift the grid (deterministically or randomly) [90]. The same approach works for quadtrees—one can shift the quadtree constructed for a point set several times such that for any pair of points in the quadtree, there will be a shift where the two points are in a cell of diameter that is $O_d(1)$ times their distance. (Recall that we use the O_d notation to hide constants that depend on d. Similarly, O_ε hides dependencies on ε.) Improving an earlier work by Bern [22], Chan [34] showed that $2\lceil d/2\rceil + 1$ deterministic shifts are enough in d dimensions. A somewhat similar shifting scheme was also suggested by Feige and Krauthgamer [63]. Random shifting of quadtrees underlines, for example, the approximation algorithm by Arora for Euclidean TSP [13].

By combining Z-order with shifting, both Chan [35] and Liao et al. [106] observed an extremely simple

data structure for $O_d(1)$-approximate nearest neighbor search in constant dimensions: just store the points in Z-order for each of the $2\lceil d/2 \rceil + 1$ shifts; given a query point q, find the successor and predecessor of q in the Z-order by binary search for each of the shifts, and return the closest point found. The data structure can be easily made dynamic to support insertions and deletions of points, and can also be adapted to find $O_d(1)$-approximate bichromatic closest pairs.

For approximate nearest neighbor (ANN) search, the $O_d(1)$ approximation factor can be reduced to $1+\varepsilon$ for any fixed $\varepsilon > 0$, though the query algorithm becomes more involved [35] and unfortunately cannot be adapted to compute $(1+\varepsilon)$-approximate bichromatic closest pairs dynamically. (In the monochromatic case, however, the approach can be adapted to find *exact* closest pairs, by considering $O_d(1)$ successors and predecessors of each point [35].)

For other proximity-related problems such as spanners and approximate minimum spanning trees (MST), this approach does not seem to work as well: for example, the static algorithms in [37], which use the Z-order, still requires explicit constructions of compressed quadtrees and are not easily dynamizable.

2.1.3 Our results

(A) **New technique: Locality-sensitive orderings.** For any given $\varepsilon > 0$, we show that there is a family of $O_d((1/\varepsilon^d)\log(1/\varepsilon))$ orderings of $[0,1)^d$ with the following property: For any $p, q \in [0,1)^d$, there is an ordering in the family such that all points lying between p and q in this ordering are within distance at most $\varepsilon \|p-q\|$ from either p or q (where $\|\cdot\|$ is the standard Euclidean norm). The order between two points can be determined efficiently using some bitwise-logical operations. See Theorem 2.1. We refer to these as ***locality-sensitive orderings***. They generalize the previous construction of $2\lceil d/2 \rceil + 1$ shifted copies of the Z-order, which guarantees the stated property only for a large specific constant (equivalent to setting $\varepsilon \approx d^{3/2}$). The new refined property ensures, for example, that a $(1+\varepsilon)$-approximate nearest neighbor of a point q can be found among the immediate predecessors and successors of q in these orderings.

Locality-sensitive orderings immediately lead to simple algorithms for a number of problems, as listed below. Many of these results are significant simplification of previous work; some of the results are new.

(B) **Approximate bichromatic closest pair.** Theorem 2.2 presents a data structure that maintains a $(1+\varepsilon)$-approximate closest bichromatic pair for two sets of points in $\mathbb{R}^d$, with an update time of $O_{d,\varepsilon}(\log n)$, for any fixed $\varepsilon > 0$ (the hidden factors depending on ε are proportional to $(1/\varepsilon^d)\log^2(1/\varepsilon)$). Previously, a general technique of Eppstein [59] can be applied in conjunction with a dynamic data structure for ANN, but the amortized update time increases by two $\log n$ factors.

(C) **Dynamic spanners.** For a parameter $t \geq 1$ and a set of points P in $\mathbb{R}^d$, a graph $G = (P, E)$ is a t-spanner

for P if for all $p, q \in P$, there is a p-q path in G of length at most $t\,\|p-q\|$. Static algorithms for spanners have been extensively studied in computational geometry. The dynamic problem appears tougher, and has also received much attention (see Table 2.1). We obtain a very simple data structure for maintaining dynamic $(1+\varepsilon)$-spanners in Euclidean space with an update (insertion and deletion) time of $O_{d,\varepsilon}(\log n)$ and having $O_{d,\varepsilon}(n)$ edges in total, for any fixed $\varepsilon > 0$. See Theorem 2.3. Although Gottlieb and Roditty [69] have previously obtained the same update time $O_{d,\varepsilon}(\log n)$, their method requires much more intricate details. (Note that Gottlieb and Roditty's method more generally applies to spaces with bounded doubling dimension, but no simpler methods have been reported in the Euclidean setting.)

(D) **Dynamic approximate minimum spanning trees.** As is well-known [30, 78], a $(1+\varepsilon)$-approximate Euclidean MST of a point set P can be computed from the MST of a $(1+\varepsilon)$-spanner of P. In our dynamic spanner (and also Gottlieb and Roditty's method [69]), each insertion/deletion of a point causes $O_{d,\varepsilon}(1)$ edge updates to the graph. Immediately, we thus obtain a dynamic data structure for maintaining a $(1+\varepsilon)$-approximate Euclidean MST, with update time (ignoring dependencies on d and ε) equal to that for the dynamic graph MST problem, which is currently $O(\log^4 n/\log\log n)$ with amortization [91].

(E) **Static and dynamic vertex-fault-tolerant spanners.** For parameters $k, t \geq 1$ and a set of points P in $\mathbb{R}^d$, a k-vertex-fault-tolerant t-spanner is a graph G which is a t-spanner and for any $P' \subseteq P$ of size at most k, the graph $G \setminus P'$ remains a t-spanner for $P \setminus P'$. Fault-tolerant spanners have been extensively studied (see Table 2.2). Locality-sensitive orderings lead to a very simple construction for k-vertex-fault-tolerant $(1+\varepsilon)$-spanners, with $O_{d,\varepsilon}(kn)$ edges, maximum degree $O_{d,\varepsilon}(k)$, and $O_{d,\varepsilon}(n\log n + kn)$ running time. See Theorem 2.4. Although this result was known before, all previous constructions (including suboptimal ones), from Levcopoulos et al.'s [104] to Solomon's work [146], as listed in Table 2.2, require intricate details. It is remarkable how effortlessly we achieve optimal $O_{d,\varepsilon}(k)$ degree, compared to the previous methods. (Note, however, that some of the more recent previous constructions more generally apply to spaces with bounded doubling dimension, and some also achieve good bounds on other parameters such as the total weight and the hop-diameter.)

Our algorithm can be easily made dynamic, with $O_{d,\varepsilon}(\log n + k)$ update time. No previous results on dynamic fault-tolerant spanners were known.

(F) **Approximate nearest neighbors.** Locality-sensitive orderings lead to a simple dynamic data structure for $(1+\varepsilon)$-approximate nearest neighbor search with $O_{d,\varepsilon}(\log n)$ time per update/query. While this result is not new [35], we emphasize that the query algorithm is the simplest so far—it is just a binary search in the orderings maintained.

reference	insertion time	deletion time
Roditty [134]	$\log n$	$n^{1/3} \log^{O(1)} n$
Gottlieb and Roditty [70]	$\log^2 n$	$\log^3 n$
Gottlieb and Roditty [69]	$\log n$	$\log n$
Theorem 2.3	$\log n$	$\log n$

Table 2.1: Previous work and our result on dynamic $(1+\varepsilon)$-spanners in $\mathbb{R}^d$. All bounds are of the form $O_{d,\varepsilon}(\,\cdot\,)$ (the hidden dependencies on ε are $1/\varepsilon^{O(d)}$).

reference	# edges	degree	running time
Levcopoulos et al. [104]	$2^{O(k)}n$	$2^{O(k)}$	$n \log n + 2^{O(k)}n$
	$k^2 n$	unbounded	$n \log n + k^2 n$
	$kn \log n$	unbounded	$kn \log n$
Lukovszki [107, 108]	kn	k^2	$n \log^{d-1} n + kn \log\log n$
Czumaj and Zhao [52]	kn	k	$kn \log^d n + k^2 n \log k$
H. Chan et al. [31]	$k^2 n$	k^2	$n \log n + k^2 n$
Kapoor and Li [96]/Solomon [146]	kn	k	$n \log n + kn$
Theorem 2.4	kn	k	$n \log n + kn$

Table 2.2: Previous work and our result on static k-vertex-fault-tolerant $(1+\varepsilon)$-spanners in $\mathbb{R}^d$. All bounds are of the form $O_{d,\varepsilon}(\,\cdot\,)$ (the hidden dependencies on ε are $1/\varepsilon^{O(d)}$).

Computational models and assumptions As stated in Chapter 1, the model of computation we have assumed is a unit-cost real RAM, supporting standard arithmetic operations and comparisons (but no floor function), augmented with bitwise-logical operations (bitwise-exclusive-or and bitwise-and), which are commonly available in programming languages (and in reality are cheaper than some arithmetic operations like multiplication).

If we assume that input coordinates are integers bounded by U and instead work in the word RAM model with $(\log U)$-bit words ($U \geq n$), then our approach can actually yield *sublogarithmic* query/update time. For example, we can achieve $O_{d,\varepsilon}(\log\log U)$ expected time for dynamic approximate bichromatic closest pair, dynamic spanners, and dynamic ANN, by replacing binary search with van Emde Boas trees [58]. Sublogarithmic algorithms were known before for dynamic ANN [35], but ours is the first sublogarithmic result for dynamic $(1+\varepsilon)$-spanners. Our results also answers the open problem of dynamic $(1+\varepsilon)$-approximate bichromatic closest pair in sublogarithmic time, originally posed by Chan and Skrepetos [39].

Throughout, we assume (without loss of generality) that ε is a power of 2; that is, $\varepsilon = 2^{-E}$ for some positive integer E.

2.2 LOCALITY-SENSITIVE ORDERINGS

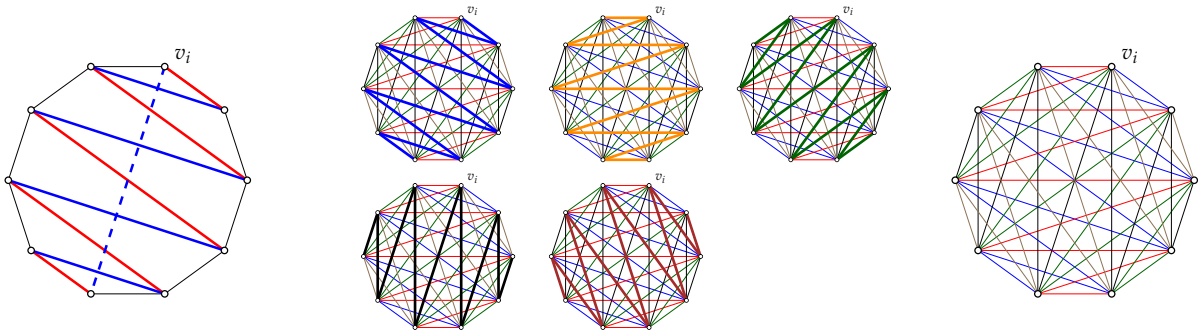


Figure 2.2: For n even, a decomposition of K_n into $n/2$ Hamiltonian paths.

2.2.1 Grids and orderings

Definition 2.1. For a set X, consider a total order (or ordering) $\prec$ on the elements of X. Two elements $x, y \in X$ are ***adjacent*** if there is no element $z \in X$, such that $x \prec z \prec y$ or $y \prec z \prec x$. For two elements $x, y \in X$ such that $x \prec y$, the ***interval*** $[x, y)$ is the set $[x, y) = \{x\} \cup \{z \in X \mid x \prec z \prec y\}$.

The following is well known, and goes back to a work by Walecki in the 19th century [7]. We include a proof for the sake of completeness. (If we don't care about the constant factor in the number of orderings, there are other straightforward alternative proofs.)

Lemma 2.1. *For n elements $\{0, \ldots, n-1\}$, there is a set $\mathfrak{O}$ of $\lceil n/2 \rceil$ orderings of the elements, such that, for all $i, j \in \{0, \ldots, n-1\}$, there exists an ordering $\sigma \in \mathfrak{O}$ in which i and j are adjacent.*

Proof: As mentioned earlier this is well known [7]. Assume n is even, and consider the clique K_n, with its vertices $v_0, \ldots, v_{n-1}$. The edges of this clique can be covered by $n/2$ Hamiltonian paths that are edge disjoint. Tracing one of these path gives rise to one ordering, and doing this for all paths results with orderings with the desired property, since edge $v_i v_j$ is adjacent in one of these paths.

To get this cover, draw K_n by using the vertices of an n-regular polygon, and draw all the edges of K_n as straight segments. For every edge $v_i v_{i+1}$ of K_n there are exactly $n/2$ parallel edges with this slope (which form a matching). Let M_i denote this matching. Similarly, for the vertex v_i, consider the segment $v_i v_{i+n/2}$ (indices are here modulo n), and the family of segments (i.e., edges) of K_n that are orthogonal to this segment. This family is also a matching M_i' of size $n/2 - 1$. Observe that $\sigma_i = M_i \cup M_i'$ forms a Hamiltonian path, as shown in Figure 2.2. Since the slopes of the segments in M_i and M_i' are unique, for $i = 0, \ldots, n/2 - 1$, it follows that $\sigma_0, \ldots, \sigma_{n/2-1}$ are an edge-disjoint cover of all the edges of K_n by $n/2$ Hamiltonian paths.

If n is odd, use the above construction for $n + 1$, and delete the redundant symbol from the computed orderings. QED.

Definition 2.2. Consider an axis-parallel cube $\mathcal{C} \subseteq \mathbb{R}^d$ with side length ℓ. Partitioning it uniformly into a $t \times t \times \cdots \times t$ grid G creates the ***t-grid*** of $\mathcal{C}$. The grid G is a set of t^d identically sized cubes with side length ℓ/t.

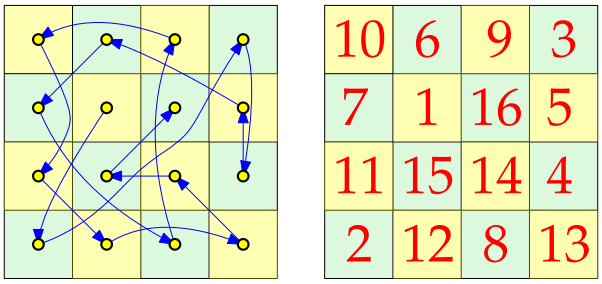


Figure 2.3: One ordering of a set of cells.

For a cube $\square \subseteq \mathbb{R}^d$, its ***diameter*** is $\mathsf{diam}(\square) = \mathsf{sidelength}(\square)\sqrt{d}$.

By Lemma 2.1 we obtain the following result.

Corollary 2.1. *For a t-grid G of an axis-parallel cube $\mathcal{C} \subseteq \mathbb{R}^d$, there is a set $\mathfrak{O}(t,d)$ of $O(t^d)$ orderings, such that for any $\square_1, \square_2 \in \mathsf{G}$, there exists an order $\sigma \in \mathfrak{O}(t,d)$ where $\square_1$ and $\square_2$ are adjacent in σ.*

2.2.2 ε-Quadtrees

Definition 2.3. An ***ε-quadtree*** $\mathcal{T}_\varepsilon$ is a quadtree-like structure, built on a cube with side length ℓ, where each cell is partitioned into a $(1/\varepsilon)$-grid. The construction then continues recursively into each grid cell of interest. As such, a node in this tree has up to $1/\varepsilon^d$ children, and a node at level $i \geq 0$ has an associated cube of side length $\ell\varepsilon^i$. When $\varepsilon = 1/2$, this is a regular quadtree.

Lemma 2.2. *Let $E > 0$ be an integer, $\varepsilon = 2^{-E}$, and $\mathcal{T}$ be a regular quadtree over $[0,2)^d$. There are ε-quadtrees $\mathcal{T}_\varepsilon^0, \ldots, \mathcal{T}_\varepsilon^{E-1}$, such that the collection of cells at each level in $\mathcal{T}$ is contained in exactly one of these ε-quadtrees.*

Proof: For $i = 0, \ldots, E-1$, construct the ε-quadtree $\mathcal{T}_\varepsilon^i$ using the cube $\left[0, 2^{E-i+1}\right)^d \supseteq [0,2)^d$ as the root. Now for $j \in \{0, \ldots, E-1\}$, observe that the collection of cells at levels $j, j+E, j+2E, \ldots,$ of $\mathcal{T}$ will also be in the quadtree $\mathcal{T}_\varepsilon^j$. Indeed, any node at level $j + \ell E$ in $\mathcal{T}$ corresponds to a cell of side length $2^{-(j+\ell E)+1}$. Now in the $(\ell+1)$th level of quadtree $\mathcal{T}_\varepsilon^j$, this same node will have side length $\varepsilon^{\ell+1}2^{E-j+1} = 2^{-(j+\ell E)+1}$. QED.

Consider an ε-quadtree $\mathcal{T}_\varepsilon$. Every node has up to $1/\varepsilon^d$ children. Consider any ordering σ of $\left\{1, \ldots, 1/\varepsilon^d\right\}$. Conceptually speaking, this induces a DFS of $\mathcal{T}_\varepsilon$ that always visits the children of a node in the order specified by σ. This induces an ordering on the points in the cube which is the root of $\mathcal{T}_\varepsilon$. Indeed, for any two points, imagine storing them in an ε-quadtree—this implies that the two points are each stored in their own leaf node, which contains no other point of interest. Now, independently of what other points are stored in the quadtree, this DFS traversal would visit these two points in the same order. This can be viewed as a space filling curve (which is not continuous) which maps a cube to an interval. This is a generalization of the Z-order. In particular, given a point set stored in $\mathcal{T}_\varepsilon$, and given σ, one can conceptually order the points according to this DFS traversal, resulting in 1-dimensional ordering of the points. We denote the resulting ordering by $(\mathcal{T}_\varepsilon, \sigma)$.

In Section 2.2.3, we show that given $(\mathcal{T}_\varepsilon, \sigma)$, the order of any two points in $[0,2)^d$ can be determined efficiently, and avoids explicitly handling this DFS traversal of $\mathcal{T}_\varepsilon$. Alternatively, the DFS on $\mathcal{T}_\varepsilon$ (according to σ) is implicitly defined by the total ordering $(\mathcal{T}_\varepsilon, \sigma)$ of points in $[0,2)^d$.

Definition 2.4. Let Π be the set of all orderings of $[0,2)^d$, induced by picking one of the $\lg(1/\varepsilon)$ trees of Lemma 2.2, together with an ordering $\sigma \in \mathfrak{O}(1/\varepsilon, d)$, as defined by Lemma 2.1. Each ordering in Π is an *ε-ordering*.

Suppose there are two points which lie in a quadtree cell that has diameter close to their distance. Formally, consider two points $p, q \in [0,1)^d$, a parameter $\varepsilon > 0$, such that p, q are both contained in a cell $\square$ of the regular quadtree $\mathcal{T}$ with $\mathsf{diam}(\square) \leq 2\|p-q\|$. Then, there is an ε-quadtree $\mathcal{T}_\varepsilon$ that has $\square$ as a node, and let $\square_p$ and $\square_q$ be the two children of $\square$ in $\mathcal{T}_\varepsilon$, containing p and q respectively. Furthermore, there is an ordering $\sigma \in \mathfrak{O}(1/\varepsilon, d)$, such that $\square_p$ and $\square_q$ are adjacent. As such, the cube $\square_p$ (resp., $\square_q$) corresponds to an interval $[x, x')$ (resp., $[x', x'')$) in the ordering $(\mathcal{T}_\varepsilon, \sigma)$, and these two intervals are adjacent. In particular, this implies that all points lying between p and q in σ have distance at most $2\varepsilon\|p-q\|$ from p or q.

If the above statement were true for all pairs of points, then this would imply the main result (Theorem 2.1). However, consider the case when there are two points close together, but no appropriately sized quadtree cell contains both p and q. In other words, two points that are close together might get separated by nodes that are much bigger in the quadtree, and this would not provide the guarantee of the main result. However, this issue can be resolved using shifting. We need the following result of Chan [34, Lemma 3.3]

Lemma 2.3. *Consider any two points $p, q \in [0,1)^d$, and let $\mathcal{T}$ be the infinite quadtree of $[0,2)^d$. For $D = 2\lceil d/2 \rceil$ and $i = 0, \ldots, D$, let $v_i = (i/(D+1), \ldots, i/(D+1))$. Then there exists an $i \in \{0, \ldots, D\}$, such that $p + v_i$ and $q + v_i$ are contained in a cell of $\mathcal{T}$ with side length $\leq 2(D+1)\|p-q\|$.*

2.2.3 Comparing two points according to an ε-ordering

We now show how to efficiently compare two points in P according to a given ε-ordering σ with a shift v_i. The shift can be added to the two points directly, and as such we can focus on comparing two points according to σ.

First, we show how to compare the msb of two numbers using only bitwise-exclusive-or and bitwise-and operations. We remark that Observation 2.1 (A) is from Chan [35].

Observation 2.1. *Let $\oplus$ denote the bitwise-exclusive-or operator. Let $\mathsf{msb}(a) := -\lfloor \lg a \rfloor$ to be the index of the most significant bit in the binary expansion of $a \in [0,2)$. Given $a, b \in [0,2)$, one can compare the* msb *of two numbers using the following:*

(A) $\mathsf{msb}(a) > \mathsf{msb}(b)$ *if and only if $a < b$ and $a < a \oplus b$.*

(B) $\mathsf{msb}(a) = \mathsf{msb}(b)$ *if and only if $a \oplus b \leq a \wedge b$, where $\wedge$ is the bitwise-and operator.*

Proof: (A) Observe that if $\mathsf{msb}(a) > \mathsf{msb}(b)$, then $2^{-\mathsf{msb}(a)} \leq a < 2^{-\mathsf{msb}(a)+1} \leq 2^{-\mathsf{msb}(b)} \leq b$. Since $\mathsf{msb}(a) > \mathsf{msb}(b)$ and $a < b$, we have $\mathsf{msb}(a \oplus b) = \mathsf{msb}(b)$. As such, we have $a < 2^{-\mathsf{msb}(a)+1} \leq 2^{-\mathsf{msb}(b)} = 2^{-\mathsf{msb}(a \oplus b)} \leq a \oplus b$.

Assume that $a < b$ and $a < a \oplus b$. Since $a < b$, it must be that $\mathsf{msb}(a) \geq \mathsf{msb}(b)$. Observe that if $\mathsf{msb}(a) = \mathsf{msb}(b)$, then $a \oplus b < a$, which is impossible. It follows that $\mathsf{msb}(a) > \mathsf{msb}(b)$, as desired.

(B) Follows by applying (A) twice (in addition to using the inequalities $a \wedge b \leq a$ and $a \wedge b \leq b$), one can show that $a \oplus b \leq a \wedge b$ if and only if $\mathsf{msb}(a) \geq \mathsf{msb}(b)$ and $\mathsf{msb}(b) \geq \mathsf{msb}(a)$. QED.

Lemma 2.4. *Let $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$ be two distinct points in $P \subseteq [0,2)^d$ and $\sigma \in \Pi$ be an ε-ordering over the cells of some ε-quadtree $\mathcal{T}_\varepsilon$ storing P. Then one can determine if $p \prec_\sigma q$ using $O(d \log(1/\varepsilon))$ bitwise-logical operations.*

Proof: Recall ε is a power of two and $E = \lg(1/\varepsilon)$. In order to compare p and q, for $i = 1, \ldots, d$, compute $a_i = p_i \oplus q_i$. Find an index i' such that $\mathsf{msb}(a_{i'}) \leq \mathsf{msb}(a_i)$ for all i. Such an index can be computed with $O(d)$ msb comparisons (using Observation 2.1 (A)). Given $p_{i'}$ and $q_{i'}$, we next determine the place in which $p_{i'}$ and $q_{i'}$ first differ in their binary representation. Note that because ε is a power of two, each digit in the base $1/\varepsilon$ expansion of $p_{i'}$ corresponds to a block of E bits in the binary expansion of $p_{i'}$. Suppose that $p_{i'}$ and $q_{i'}$ first differ inside the hth block at an index $j \in \{1, \ldots E\}$.

The algorithm now locates this index j. To do so, for $j = 1, \ldots, E$, let $b_j = 2^{E-j}/(2^E - 1) \in (0,1]$ be the number whose binary expansion has a 1 in positions $j, j+E, j+2E, \ldots$, and 0 everywhere else. For $j = 1, \ldots, E$, compute $b_j \wedge a_{i'}$ and check if $\mathsf{msb}(a_{i'}) = \mathsf{msb}(b_j \wedge a_{i'})$ (using Observation 2.1 (B)). When the algorithm finds such an index j obeying this equality, it exits the loop. We know that $p_{i'}$ and $q_{i'}$ first differ in the jth position inside the hth block (the value of h is never explicitly computed).

It remains to extract the E bits from the hth block in each coordinate $p_1, \ldots, p_d$. For $i = 1, \ldots, d$, let $B_i \in [0,1]^E$ be the bits inside the hth block of p_i. For $k = 1, \ldots, E$, set

$$B_{i,k} = \mathbb{1}\left[\mathsf{msb}(2^{j-k} a_{i'}) = \mathsf{msb}((2^{j-k} a_{i'}) \wedge p_i)\right],$$

where $\mathbb{1}[\cdot]$ is the indicator function). By repeating a similar process for all $q_1, \ldots, q_d$, we obtain the coordinates of the cells in which p and q differ. We can then consult σ to determine whether or not $p \prec_\sigma q$.

This implies that p and q can be compared using $O(d \log(1/\varepsilon))$ operations by Observation 2.1. QED.

Remark In the word RAM model for integer input, the extra $\log(1/\varepsilon)$ factor in the above time bound can be eliminated: msb can be explicitly computed in $O(1)$ time by a complicated algorithm of Fredman and Willard [67]; this allows us to directly jump to the right block of each coordinate and extract the relevant bits. (Furthermore, assembly operations performing such computations are nowadays available on most CPUs.)

2.2.4 The result

Theorem 2.1. *For $\varepsilon \in (0, 1/2]$, there is a set Π^+ of $O_d(\log(1/\varepsilon)/\varepsilon^d)$ orderings of $[0,1)^d$, such that for any two points $p, q \in [0,1)^d$ there is an ordering $\sigma \in \Pi^+$ defined over $[0,1)^d$, such that for any point u with $p \prec_\sigma u \prec_\sigma q$ it holds that either $\|p-u\| \leq \varepsilon \|p-q\|$ or $\|q-u\| \leq \varepsilon \|p-q\|$.*

Furthermore, given such an ordering σ, and two points p, q, one can compute their ordering, according to σ, using $O(d \log(1/\varepsilon))$ arithmetic and bitwise-logical operations.

Proof: Let Π^+ be the set of all orderings defined by picking an ordering from Π, as defined by Definition 2.4 using the parameter ε, together with a shift from Lemma 2.3.

Consider any two points $p, q \in [0,1)^d$. By Lemma 2.3 there is a shift for which the two points fall into a quadtree cell $\square$ with side length at most $2(D+1)\|p-q\|$. Next, there is an ε-quadtree $\mathcal{T}_\varepsilon$ that contains $\square$, and the two children that correspond to two cells $\square_p$ and $\square_q$ with side length at most $2(D+1)\varepsilon\|p-q\|$, which readily implies that the diameter of these cells is at most $2(D+1)\sqrt{d}\varepsilon\|p-q\|$. Furthermore, there is an ε-ordering in Π such that all the points of $\square_p$ are adjacent to all the points of $\square_q$ in this ordering. This implies the desired claim, after adjusting ε by a factor of $2(D+1)\sqrt{d}$ (and rounding to a power of 2). QED.

From now on, we refer to the set of orderings Π^+ in the above Theorem as locality-sensitive orderings. We remark that by the readjustment of ε in the final step of the proof, the number of locality-sensitive orderings when including the factors involving d is $O(d^{3/2})^d \cdot (1/\varepsilon^d)\log(1/\varepsilon)$.

2.2.5 Discussion

Connection to locality-sensitive hashing Let P be a set of n points in Hamming space $[0,1]^d$. Consider the decision version of the $(1+\varepsilon)$-approximate nearest neighbor problem. Specifically, for a pre-specified radius r and any given query point q, we would like to efficiently decide whether or not there exists a point $p \in P$ such that $\|q-p\|_1 \leq (1+\varepsilon)r$ or conclude that all points in P are at least distance r from q. The locality-sensitive hashing (LSH) technique [92] implies the existence of a data structure supporting this type of decision query in time $O(dn^{1/(1+\varepsilon)}\log n)$ time (which is correct with high probability) and using total space $O(dn^{1+1/(1+\varepsilon)}\log n)$. Similar results also hold in the Euclidean setting.

At a high level, LSH works as follows. Start by choosing $k := k(\varepsilon, r, n)$ indices in $[d]$ at random (with replacement). Let R denote the resulting multiset of coordinates. For each point $p \in P$, let p_R be the projection p onto these coordinates of R. We can group the points of P into buckets, where each bucket contains points with the same projection. Given a query point q, we check if any of the points in the same bucket as q is at distance at most $(1+\varepsilon)r$ from q. This construction can also be repeated a sufficient number of times in order to guarantee success with high probability.

The idea of bucketing can also be viewed as an implicit ordering on the randomly projected point set by ordering points lexicographically according to the k coordinates. In this sense, the query algorithm can be

viewed as locating q within each of the orderings, and comparing q to similar points nearby in each ordering. From this perspective, every locality-sensitive ordering can be viewed as an LSH scheme. Indeed, for a given query point q, the approximate nearest neighbor to q can be found by inspecting the elements adjacent to q in each of the locality-sensitive orderings and returning the closest point to q found (see Theorem 2.5).

Of course, the main difference between the two schemes is that for every fixed ε, the number of "orderings" in an LSH scheme is polynomial in both d and n. While for locality-sensitive orderings, the number of orderings remains exponential in d. This trade-off is to be expected, as locality-sensitive orderings guarantee a much stronger property than that of an LSH scheme.

Extension of locality-sensitive orderings to other norms in Euclidean space The L_p-norm, for $p \geq 1$, of a vector $x \in \mathbb{R}^d$ is defined as $\|x\|_p = \left(|x_1|^p + \cdots + |x_n|^p\right)^{1/p}$. The L_∞-norm, or maximum norm, is defined as $\|x\|_\infty = \max(|x_1|, \ldots, |x_n|)$.

The result of Theorem 2.1 also holds for any L_p-norm. The key change that is needed is in the proof of Lemma 2.3: For any two points $s, t \in [0,1)^d$, there exists a shift v such that $s + v$ and $t + v$ are contained in a quadtree cell of side length at most $2(D+1)\|s - t\|_p$. This extension follows easily from the proof of the Lemma. Theorem 2.1 then follows by adjusting ε by a factor of $2(D+1)d^{1/p}$ in the last step, implying that the number of orderings will be $O(d^{1+1/p})^d(1/\varepsilon^d)\log(1/\varepsilon)$. (For the L_∞-norm, ε only needs to be adjusted by a factor of $2(D+1)$.)

Extension of locality-sensitive orderings for doubling metrics A metric space with (low) doubling dimension is an abstraction of a low-dimensional Euclidean space. Formally, a metric space $(\mathcal{M}, d)$ has *doubling dimension* λ if any ball of $\mathcal{M}$ of radius r can be covered by at most 2^λ balls of half the radius (i.e., $r/2$). It is known that $\mathbb{R}^d$ has doubling dimension $O(d)$ [152]. We point out that locality-sensitive orderings still exist in this case, but they are less constructive in nature, since one needs to be provided with all the points of interest in advance.

For a point set $P \subseteq \mathcal{M}$, the analogue of a quadtree for a metric space is a net tree [85]. A net tree can be constructed as follows (the construction algorithm described here is somewhat imprecise): The root node corresponds to the point set $P \subseteq \mathcal{M}$. Compute a randomized partition of P of diameter $1/2$ (assume P has diameter one) [78, Chapter 26], and for each cluster in the partition, create an associated node and hang it on the root. The tree is computed recursively in this manner, at each level i computing a random partition of diameter 2^{-i}. The leaves of the tree are points of P.

As with quadtrees, it is possible during this randomized construction for two nearby points to be placed in different clusters and be separated further down the tree. If $\ell = d(p,q)$ for two points $p, q \in P$, then the probability that p and q lie in different clusters of diameter $r = 2^{-i}$ in the randomized partition is at most $O((\ell/r)\log n)$ [62]. In particular, for $r \approx 1/(\ell \log n)$, the probability that p and q are separated is at most a constant. If we want this property to hold with high probability for all pairs of points, one needs to construct

$O(\log n)$ (randomly constructed) net trees of P. (This corresponds to randomly shifting a quadtree $O(\log n)$ times in the Euclidean setting.)

Given such a net tree T, each node has $I = 2^{O(\lambda)}$ children. We can arbitrarily and explicitly number the children of each node by a distinct label from $\{1, \ldots, I\}$. One can define an ordering of such a tree as we did in the Euclidean case, except that the gap (in diameter) between a node and its children is $O(\varepsilon/\log n)$ instead of ε. Repeating our scheme in the Euclidean case, this implies that one would expect to require $(\varepsilon^{-1} \log n)^{O(\lambda)}$ orderings of P.

This requires having all the points of P in advance, which is a strong assumption for a dynamic data structure (as in some of the applications below). For example, Gottlieb and Roditty [69] show how to maintain dynamic spanners in a doubling metric, but only assuming that after a point has been deleted from P, the distance between the deleted point and a point currently in P can still be computed in constant time. Recently, Filtser and Le recently gave a construction of locality-sensitive orderings for metrics with doubling dimension λ of size $O(1/\varepsilon^{O(\lambda)})$ [66], which removes the dependency on n and improves on the construction sketched above.

2.3 APPLICATIONS

2.3.1 Bichromatic closest pair

Given an ordering $\sigma \in \Pi^+$, and two finite sets of points R, B in $\mathbb{R}^d$, let $\mathcal{Z} = \mathcal{Z}(\sigma, R, B)$ be the set of all pairs of points in $R \times B$ that are adjacent in the ordering of $R \cup B$ according to σ. Observe that inserting or deleting a single point from these two sets changes the contents of $\mathcal{Z}$ by a constant number of pairs. Furthermore, a point participates in at most two pairs.

Lemma 2.5. *Let R and B be two sets of points in $[0,1)^d$, and let $\varepsilon \in (0,1)$ be a parameter. Let $\sigma \in \Pi^+$ be a locality-sensitive ordering (see Theorem 2.1). Then, one can maintain the set $\mathcal{Z} = \mathcal{Z}(\sigma, R, B)$ under insertions and deletions to R and B. In addition, one can maintain the closest pair in $\mathcal{Z}$ (under the Euclidean metric). Each update takes $O(d \log n \log(1/\varepsilon))$ time, where n is the total size of R and B during the update operation.*

Proof: Maintain two balanced binary search trees T_R and T_B storing the points in R and B, respectively, according to the order σ. Insertion, deletion, predecessor query and successor query can be implemented in $O(d \log(1/\varepsilon) \log n)$ time (since any query requires $O(\log n)$ comparisons each costing $O(d \log(1/\varepsilon))$ time by Lemma 2.4). We also maintain a min-heap of the pairs in $\mathcal{Z}$ sorted according to the Euclidean distance. The minimum is the desired closest pair. Notice that a single point can participate in at most two pairs in $\mathcal{Z}$.

We now explain how to handle updates. Given a newly inserted point r (say a red point that belongs to R), we compute the (potential) pairs it participates in, by computing its successor r' in R, and its successor b' in B. If $r \prec_\sigma b' \prec_\sigma r'$ then the new pair rb' should be added to $\mathcal{Z}$. The pair before r in the ordering that might

use r is computed in a similar fashion. In addition, we recompute the predecessor and successor of r in R, and we recompute the pairs they might participate in (deleting potentially old pairs that are no longer valid).

Deletion is handled in a similar fashion—all points included in pairs with the deleted point recompute their pairs. In addition, the successor and predecessor (of the same color) need to recompute their pairs. This all requires a constant number of queries in the two trees, and thus takes the running time as stated.QED.

Theorem 2.2. *Let R and B be two sets of points in $[0,1)^d$, and let $\varepsilon \in (0,1/2]$ be a parameter. Then one can maintain a $(1+\varepsilon)$-approximation to the bichromatic closest pair in $R \times B$ under updates (i.e., insertions and deletions) in $O_d(\log n \log^2(1/\varepsilon)/\varepsilon^d)$ time per operation, where n is the total number of points in the two sets. The data structure uses $O_d(n \log(1/\varepsilon)/\varepsilon^d)$ space, and at all times maintains a pair of points $r \in R$, $b \in B$, such that $\|r-b\| \leq (1+\varepsilon)d(R,B)$, where $d(R,B) = \min_{r\in R, b\in B} \|r-b\|$.*

Proof: We maintain the data structure of Lemma 2.5 for all the locality-sensitive orderings of Theorem 2.1. All the good pairs for these data structures can be maintained together in one global min-heap. The claim is that the minimum length pair in this heap is the desired approximation.

To see that, consider the bichromatic closest pair $r \in R$ and $b \in B$. By Theorem 2.1 there is a locality-sensitive ordering σ, such that the interval I in the ordering between r and b contains points that are in distance at most $\ell = \varepsilon \|r-b\|$ from either r or b. In particular, let P_r (resp., P_b) be all the points in I in distance at most ℓ from r (resp., b). Observe that $P_r \subseteq R$, as otherwise, there would be a bichromatic pair in P_R, and since the diameter of this set is at most ℓ, this would imply that (r,b) is not the closest bichromatic pair—a contradiction. Similarly, $P_b \subseteq B$. As such, there must be two points $b' \in B$ and $r' \in R$, that are consecutive in σ, and this is one of the pairs considered by the algorithm (as it is stored in the min-heap). In particular, by the triangle inequality, we have

$$\|r'-b'\| \leq \|r'-r\| + \|r-b\| + \|b-b'\| \leq 2\ell + \|r-b\| \leq (1+2\varepsilon)\|r-b\|.$$

The theorem follows after adjusting ε by a factor of 2. QED.

Remark In the word RAM model, for integer input in $\{1,\ldots,U\}^d$, the update time can be improved to $O_d((\log\log U)\log^2(1/\varepsilon)/\varepsilon^d)$ expected, by using van Emde Boas trees [58] in place of the binary search trees (and the min-heaps as well). With standard word operations, we may not be able to explicitly map each point to an integer in one dimension following each locality-sensitive ordering, but we can still simulate van Emde Boas trees on the input as if the mapping has been applied. Each recursive call in the van Emde Boas recursion focuses on a specific block of bits of each input coordinate value (after shifting); we can extract these blocks, and perform the needed hashing operations on the concatenation of these blocks over the d coordinates of each point.

2.3.2 Dynamic spanners

Definition 2.5. For a set of n points P in $\mathbb{R}^d$ and a parameter $t \geq 1$, a ***t-spanner*** of P is an undirected graph $G = (P, E)$ such that for all $p, q \in P$,

$$\|p - q\| \leq \mathsf{d}_G(p, q) \leq t\|p - q\|,$$

where $\mathsf{d}_G(p, q)$ is the length of the shortest path from p to q in G using the edge set E.

Using a small modification of the results in the previous section, we easily obtain a dynamic $(1 + \varepsilon)$-spanner. Note that there is nothing special about how the data structure in Theorem 2.2 deals with the bichromatic point set. If the point set is monochromatic, modifying the data structure in Lemma 2.5 to account for the closest monochromatic pair of points leads to a data structure with the same bounds and maintains the $(1 + \varepsilon)$-approximate closest pair.

The construction of the spanner is very simple: Given P and $\varepsilon \in (0, 1)$, maintain orderings of the points specified by Π^+ (see Theorem 2.1). For each $\sigma \in \Pi^+$, let E_σ be the edge set consisting of edges connecting two consecutive points according to σ, with weight equal to their Euclidean distance. Thus $|E_\sigma| = n - 1$. Our spanner $G = (P, E)$ then consists of the edge set $E = \bigcup_{\sigma \in \Pi^+} E_\sigma$.

Theorem 2.3. *Let P be a set of n points in $[0, 1)^d$ and $\varepsilon \in (0, 1/2]$. One can compute a $(1 + \varepsilon)$-spanner G of P with $O_d(n \log(1/\varepsilon)/\varepsilon^d)$ edges, where every vertex has degree $O_d(\log(1/\varepsilon)/\varepsilon^d)$. Furthermore, a point can be inserted or deleted in $O_d(\log n \log^2(1/\varepsilon)/\varepsilon^d)$ time, where each insertion or deletion creates or removes at most $O_d(\log(1/\varepsilon)/\varepsilon^d)$ edges in the spanner.*

Proof: The construction is described above. The same analysis as in the proof of Theorem 2.2 implies the number of edges in G and the update time.

It remains to prove that G is a spanner. By Theorem 2.1, for any pair of points $s, t \in P$, there is a locality-sensitive ordering $\sigma \in \Pi^+$, such that the σ-interval $[s, t)$ contains only points that are in distance at most $\varepsilon \|s - t\|$ from either s or t. In particular, there must be two points in $s', t' \in P$ that are adjacent in σ, such that one of them, say s' (resp., t') is in distance at most $\varepsilon \|s - t\|$ from s (resp., t). As such, the edge $s't'$ exists in the graph being maintained.

This property is already enough to imply that this graph is a $(1 + c\varepsilon)$-spanner for a sufficiently large constant c—this follows by an induction on the distances between the points (specifically, in the above, we apply the induction hypothesis on the pairs s, s' and t, t'). We omit the easy but somewhat tedious argument—see [30] or [78, Theorem 3.12] for details. The theorem follows after adjusting ε by a factor of c. QED.

2.3.3 Static and dynamic vertex-fault-tolerant spanners

Definition 2.6. For a set of n points P in $\mathbb{R}^d$ and a parameter $t \geq 1$, a ***k-vertex-fault-tolerant t-spanner*** of P, denoted by (k,t)-VFTS, is a graph $G = (P, E)$ such that

(i) G is a t-spanner (see Definition 2.5), and

(ii) For any $P' \subseteq P$ of size at most k, the graph $G \setminus P'$ is a t-spanner for $P \setminus P'$.

A $(k, 1+\varepsilon)$-VFTS can be obtained by modifying the construction of the $(1+\varepsilon)$-spanner in Section 2.3.2. Construct a set of locality-sensitive orderings Π^+. For each $\sigma \in \Pi^+$ and each $p \in P$, connect p to its $k+1$ successors and $k+1$ predecessors according to σ with edge weights equal to the Euclidean distances. Thus each ordering maintains $O(nk)$ edges and there are $O(|\Pi^+|\, kn) = O_d(kn \log(1/\varepsilon)/\varepsilon^d)$ edges overall. We now prove that this graph G is in fact a $(k, 1+\varepsilon)$-VFTS.

Theorem 2.4. *Let P be a set of n points in $[0,1)^d$ and $\varepsilon \in (0, 1/2]$. One can compute a k-vertex-fault-tolerant $(1+\varepsilon)$-spanner G for P in time*

$$O_d\Big((n \log n \log(1/\varepsilon) + kn) \log(1/\varepsilon)/\varepsilon^d\Big).$$

The number of edges is $O_d(kn \log(1/\varepsilon)/\varepsilon^d)$ and the maximum degree is bounded by $O_d(k \log(1/\varepsilon)/\varepsilon^d)$.

Furthermore, one can maintain the k-vertex-fault-tolerant $(1+\varepsilon)$-spanner G under insertions and deletions of points in $O_d\big((\log n \log(1/\varepsilon) + k) \log(1/\varepsilon)/\varepsilon^d\big)$ time per operation.

Proof: The construction algorithm, number of edges, and maximum degree follows from the discussion above. So, consider deleting a set $P' \subseteq P$ of size at most k from G. Consider an ordering $\sigma \in \Pi^+$ with the points P' removed. By the construction of G, all the pairs of points of $P \setminus P'$ that are (now) adjacent in σ remain connected by an edge in $G \setminus P'$. The argument of Theorem 2.3 implies that the remaining graph is spanner. We conclude that $G \setminus P'$ is a $(1+\varepsilon)$-spanner for $P \setminus P'$.

As for the time taken to handle insertions and deletions, one simply maintains the orderings of the points using balanced search trees. After an insertion of a point to one of the orderings in $O(\log n \log(1/\varepsilon))$ time, $O(k)$ edges have to be added and deleted. Therefore inserting a point takes $O\big((\log n \log(1/\varepsilon) + k)\, |\Pi^+|\big) = O_d\big((\log n \log(1/\varepsilon) + k) \log(1/\varepsilon)/\varepsilon^d\big)$ time total. Deletions are handled similarly.

The total construction time follows by inserting each of the points into the dynamic data structure. QED.

2.3.4 Dynamic approximate nearest neighbors

Another application of the same data structure in Theorem 2.2 is supporting $(1+\varepsilon)$-approximate nearest neighbor queries. In this scenario, the data structure must support insertions and deletions of points and the following queries: given a point q, return a point $t \in P$ such that $\|q - t\| \leq (1+\varepsilon) \min_{p \in P} \|q - p\|$.

Theorem 2.5. *Let P be a set of n points in $[0,1)^d$. For a given $\varepsilon \in (0,1/2]$, one can build a data structure using $O_d(n \log(1/\varepsilon)/\varepsilon^d)$ space, that supports insertion and deletion in time $O_d(\log n \log^2(1/\varepsilon)/\varepsilon^d)$. Furthermore, given a query point $q \in [0,1)^d$, the data structure returns a $(1+\varepsilon)$-approximate nearest neighbor in P in $O_d(\log n \log^2(1/\varepsilon)/\varepsilon^d)$ time.*

Proof: Maintain the data structure of Lemma 2.5 for all locality-sensitive orderings of Theorem 2.1, with one difference: Since the input is monochromatic, for each locality-sensitive ordering $\sigma \in \Pi^+$, we store the points in a balanced binary search tree according to σ. The space and update time bounds easily follow by the same analysis.

Given a query point $q \in [0,1)^d$, for each of the orderings the algorithm inspects the predecessor and successor to q. The algorithm returns the closest point to q encountered. We claim that the returned point p is the desired approximate nearest neighbor.

Let $p^\star \in P$ be the nearest neighbor to q and $\ell = \|q - p^\star\|$. By Theorem 2.1, there is a locality-sensitive ordering $\sigma \in \Pi^+$ such that the σ-interval $I = [p^\star, q)$ contains points that are of distance at most $\varepsilon\ell$ from $p^\star$ or q (and this interval contains at least one point of P, namely, $p^\star$). Note that no point of P can be at distance less than $\varepsilon\ell$ to q. Thus, the point $p \in P$ adjacent to q in I is of distance at most $\varepsilon\ell$ from $p^\star$. Therefore, for such a point p, we have $\|p - q\| \leq \|p - p^\star\| + \|p^\star - q\| \leq (1+\varepsilon)\ell$.

The final query time follows from the time taken for these predecessor and successor queries, as in the proof of Lemma 2.5. QED.

Part II

Geometric centers

3 Approximating centerpoints efficiently

> *A change of perspective is worth 80 IQ points.*
>
> **— Alan Kay**

Let P be a set of n points in $\mathbb{R}^d$. For a parameter $\alpha \in (0,1)$, an α-centerpoint of P is a point $p \in \mathbb{R}^d$ such that all closed halfspaces containing p also contain at least αn points of P. Such a point may not always exist for a given α, however Rado's centerpoint theorem implies that a $1/(d+1)$-centerpoint always exists [133]. We revisit an algorithm of Clarkson et al. [49], that computes (roughly) a $1/(4d^2)$-centerpoint in $\widetilde{O}(d^9)$ randomized time, where $\widetilde{O}$ hides polylogarithmic terms. We present an improved algorithm that can compute centerpoints with quality arbitrarily close to $1/d^2$ and runs in randomized time $\widetilde{O}(d^7)$. While the improvements are (arguably) mild, it is the first refinement of the algorithm by Clarkson et al. [49] in over twenty years. The new algorithm is simpler, and the running time bound follows by a simple random walk argument, which we believe to be of independent interest. In Chapter 4 we present several new applications of the improved centerpoint algorithm and explore the connections between centerpoints and weak ε-nets.

3.1 BACKGROUND

3.1.1 Computing centerpoints

For a given point p, deciding whether p is indeed a $1/(d+1)$-centerpoint is known to be co-NP-complete [149]. Furthermore, the problem of computing a centerpoint lies in the complexity class PPAD $\cap$ PLS [118]. The fastest currently known algorithm for computing a centerpoint is by Chan [33], which computes a $1/(d+1)$-centerpoint in expected time $O(n^{d-1} + n \log n)$. For this reason, past focus has been on efficiently computing centerpoints of slightly worse quality. The first (randomized) polynomial time algorithm was presented by Clarkson et al. [49], in which they compute (roughly) a $1/(4d^2)$-centerpoint in $\widetilde{O}(d^9)$ time together with a random sampling step, where $\widetilde{O}$ hides polylogarithmic factors depending on d. Since the work of Clarkson et al. [49], there has been a variety of deterministic and randomized algorithms proposed for computing centerpoints [87, 119, 124, 136]. We summarize previously known results in Table 3.1. In many of these cited algorithms, they return not only an α-centerpoint but in addition a *proof* of the depth of

reference	quality	running time	notes
Chan [33]	$\frac{1}{d+1}$	$O(n^{d-1} + n \log n)$	Las Vegas algorithm
Miller and Sheehy [119]	$\frac{1}{2(d+1)^2}$	$O(n^{1+\log_2 d})$	deterministic
Mulzer and Werner [124]	$\frac{1}{4(d+1)^3}$	$d^{O(\log d)} n$	deterministic
Har-Peled and Zhou [87]	$\frac{1-\gamma}{2(d+1)^2}$	$d^{O(\log(d/\gamma))} n$	deterministic
Rolnick and Soberón [136]	$\frac{1-\gamma}{d(d+1)}$	$O(d/\gamma)^{O(d)} + O_w(n^4 \log \varphi^{-1})$	Monte Carlo (prob. $\geq 1-\varphi$)
Clarkson et al. [49]	$\frac{2}{3e(d+2)(d+1)}$	$O(d^9 \log d + d^8 \log^2 \varphi)$	Monte Carlo (prob. $\geq 1-\varphi$)
Our result, Theorem 3.1	$\frac{1-\gamma}{(d+2)^2}$	$O(\gamma^{-4} d^7 \log^3 d \log^3(\gamma^{-1}\varphi^{-1}))$	Monte Carlo (prob. $\geq 1-\varphi$)

Table 3.1: A summary of previous and current work for computing an approximate centerpoint for n points in $\mathbb{R}^d$. The notation O_w hides polylogarithmic dependencies on the number of bits needed to encode the input.

the point (i.e., a collection of αn simplices which contain the α-centerpoint). This proof can be easily used to verify that the returned point is indeed an α-centerpoint. In particular, the result of Clarkson et al. [49] only returns the approximate centerpoint and no such proof.

3.1.2 Random sampling vs. approximate centerpoint quality

Many randomized algorithms for computing centerpoints perform a random sampling step to obtain a sample $S \subseteq P$ and return an approximate centerpoint for S [49, 109, 149]. For example, it is not hard to argue that if $S \subseteq P$ is an ε-sample for halfspaces (see Section 4.2 for relevant definitions), then an α-centerpoint for S is an $(\alpha - \varepsilon)$-centerpoint for P [49, 109]. Perhaps unsurprisingly, this easy observation implies that as the size of the random sample increases, the quality of the approximate centerpoint improves. Our proposed algorithm does perform a random sampling step, however we use relative approximations instead of ε-samples (see Lemma 3.10).

3.1.3 Our results

Let P be a set of n points in $\mathbb{R}^d$. In addition to the improved algorithm for computing approximate centerpoints, we also suggest an application (additional applications which will be covered in Chapter 4):

(A) **Approximating centerpoints.** We revisit the algorithm of Clarkson et al. [49] for approximating a centerpoint. We present an improved algorithm, which is a variant of their algorithm which runs in

$\widetilde{O}(d^7)$ time, and computes roughly a $1/(d+2)^2$-centerpoint. This improves both the running time, and the quality of centerpoint computed. While the improvements are small (a factor of d^2 roughly in the running time, and a factor of four in the centerpoint quality), we believe that the new algorithm is simpler. The analysis is cleaner, and is of independent interest. In particular, the analysis uses a random walk argument, which is quite uncommon in computational geometry, and (we believe) is of independent interest. See Theorem 3.2.

An animation of the approximate centerpoint algorithm is available on YouTube [83] (see the YouTube channel for animations of the algorithm with different point sets).

(B) **Lower-bounding convex functions.** Given a convex function f in $\mathbb{R}^d$, such that one can compute its value and gradient at a point efficiently, we present an algorithm that computes quickly a realizable lower-bound on the value of f over P. Formally, the algorithm computes a point $q \in \mathbb{R}^d$, such that $f(q) \leq \min_{p \in P} f(p)$. The algorithm is somewhat similar in spirit to the ellipsoid algorithm. The running time of the algorithm is $\widetilde{O}(d^9)$. See Theorem 3.3.

3.2 APPROXIMATING THE CENTERPOINT VIA RADON'S URN

Definition 3.1 (Centerpoint). Given a set P of n points in $\mathbb{R}^d$, and a constant $\alpha \in (0, 1/(d+1)]$, a point $c \in \mathbb{R}^d$ is an ***α-centerpoint*** if for any closed halfspace that contains c, the halfspace also contains at least αn points of P.

It is a classical consequence of Helly's theorem that a $1/(d+1)$-centerpoint always exists [133]. If a point $c \in \mathbb{R}^d$ is a $1/(d+1)$-centerpoint for P, we omit the $1/(d+1)$ and simply say that c is a centerpoint for P.

We revisit the algorithm of Clarkson et al. [49] for approximating a centerpoint. We give a simpler variant of their algorithm, and present a different (and we believe cleaner) analysis of the algorithm. In the process we improve the running time from being roughly $\widetilde{O}(d^9)$ to $\widetilde{O}(d^7)$, and also improve the quality of centerpoint computed.

Before describing the algorithm, we need the concept of Radon points. The following result is well-known [78, 112]; we state the proof for completeness.

Lemma 3.1 (Radon's Theorem). *Given a set $R = \{q_1, \ldots, q_{d+2}\}$ of $d+2$ points in $\mathbb{R}^d$, one can partition R into two non-empty sets R_1, R_2, such that $\mathsf{conv}(R_1) \cap \mathsf{conv}(R_2) \neq \emptyset$.*

Proof: Consider the set $P = \{(q_i, 1) \mid q_i \in R\} \subset \mathbb{R}^{d+1}$. Since $|P| = d+2$, the points P are linearly dependent. As such, there are coefficients $\alpha_1, \ldots, \alpha_{d+2}$, not all of them zero, such that $\sum_{i=1}^{d+2} \alpha_i q_i = 0$ and $\sum_{i=1}^{d+2} \alpha_i = 0$. Let $I_+ = \{i \mid \alpha_i \geq 0\}$ and $I_- = \{i \mid \alpha_i < 0\}$. By definition of I_+ and I_-, observe that $\sum_{i \in I_+} \alpha_i q_i = -\sum_{i \in I_-} \alpha_i q_i$ and define $\mu = \sum_{i \in I_+} \alpha_i = -\sum_{i \in I_-} \alpha_i$. Thus, by choosing $R_1 = \{q_i \mid i \in I_+\}$ and $R_2 = R \setminus R_1$, we have that the point $p = \sum_{q_i \in R_1} (\alpha_i/\mu) q_i$ is a convex combination of the points in R_1. Similarly, $p = -\sum_{q_i \in R_2} (\alpha_i/\mu) q_i \in \mathsf{conv}(R_2)$. Hence, $p \in \mathsf{conv}(R_1) \cap \mathsf{conv}(R_2)$. QED.

Definition 3.2. Let $R \subseteq \mathbb{R}^d$ be a set of $n \geq d+2$ points and let R_1, R_2 be non-empty sets which partition R such that $\mathsf{conv}(R_1) \cap \mathsf{conv}(R_2) \neq \varnothing$. Any point $p \in \mathsf{conv}(R_1) \cap \mathsf{conv}(R_2)$ is called a ***Radon point***.

Our algorithm will need the following facts about Radon points.

Lemma 3.2. *Let R be a set of $d+2$ points in $\mathbb{R}^d$:*

(A) A Radon point for R can be computed in $O(d^3)$ time.

(B) Any Radon point for R is a $2/(d+2)$-centerpoint for R.

(C) Let p be any Radon point for R. For any halfspace h^+, if h^+ contains at most one point of R, then h^+ does not contain p.

Proof: (A) The proof of Lemma 3.1 implies that a Radon point p can be extracted from the coefficients $\alpha_1, \ldots, \alpha_{d+2}$. These coefficients can be found by solving the described homogeneous linear system consisting of $d+1$ equations in $d+2$ variables. Such a system can be solved in time $O(d^3)$ via Gaussian elimination.

(B) Let h^+ be any halfspace containing a Radon point p of R. Since $p \in \mathsf{conv}(R_1) \cap \mathsf{conv}(R_2)$, convexity implies that h^+ must also contain at least one vertex of $\mathsf{conv}(R_1)$ and $\mathsf{conv}(R_2)$. Thus, $|h^+ \cap R| \geq 2 = \frac{2}{d+2}|R|$.

(C) This statement is the contrapositive of (B). QED.

3.2.1 The algorithm

Let P be a set of n points in $\mathbb{R}^d$ for which we would like to approximate its centerpoint. To this end, let Q be initially P. In each iteration the algorithm randomly picks $d+2$ points (with repetition) from Q, computes their Radon point using Lemma 3.2 (A), randomly deletes any point of Q, and inserts the new Radon point into the point set Q. See Figure 3.1. This process is repeated for sufficiently many iterations (to be determined). At the end of this process, the algorithm returns an arbitrary point in Q as the approximate centerpoint.

Remark 3.1. The algorithm above is a variant of the algorithm of Clarkson et al. [49]. Their algorithm worked in rounds, in each round generating n new Radon points, and then replacing the point set with this new set, repeating this a sufficient number of times. Our algorithm on the other hand is a "continuous" process.

3.2.2 Intuition for the analysis

By Lemma 3.2 (B), a Radon point is a decent center for the points defining it. Visually, the above algorithm causes the points to slowly migrate towards the center region of the original point set.

The analysis of the algorithm will proceed as follows. Suppose the goal is to compute an $f(d)$-centerpoint of P. To this end, consider a halfspace h^+ that contains at most $f(d)n$ points of P. Recall that at the end of the

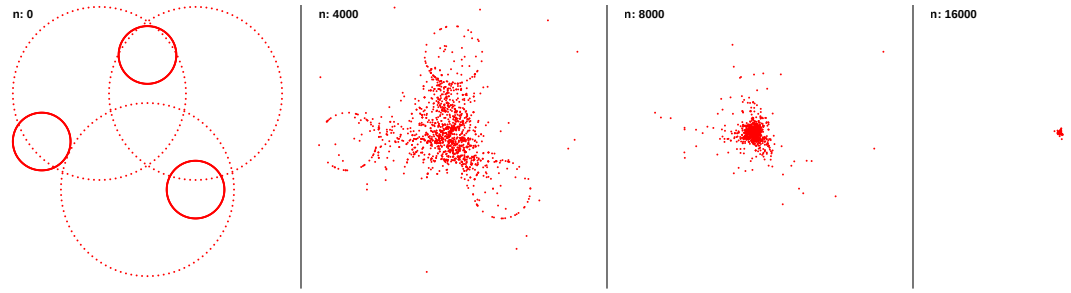


Figure 3.1: A point set consisting of points lying on three large circles and three small circles, and running the algorithm after 4000, 8000, and 16000 iterations. Several animations of the algorithm are available on YouTube [83].

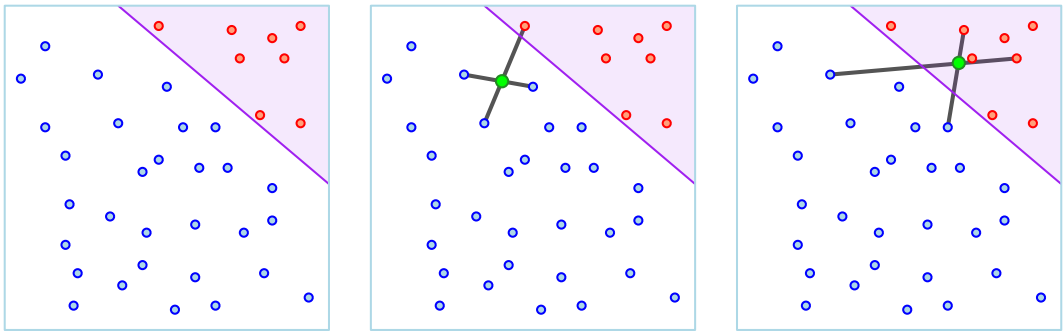


Figure 3.2: A halfspace h^+ containing $f(d)n$ points of P. Color points inside h^+ (the shaded region) as red and points outside h^+ blue. Choosing at most one red point guarantees the Radon point (green) is outside h^+.

algorithm, we return an arbitrary point in Q as the approximate centerpoint. In particular, we want to argue that after sufficiently many iterations, $\mathbb{R}^d \setminus h^+$ contains all points of Q. If we apply this argument in parallel over all such halfspaces h^+, then every point in Q at termination of the algorithm will be an $f(d)$-centerpoint of P. As such, the remainder of the analysis will focus on a fixed halfspace h^+.

At each iteration of the algorithm, we color the points of Q either red or blue. A point is colored blue if and only if it lies outside the halfspace h^+. Observe that the algorithm makes progress for h^+ when the computed Radon point falls outside h^+ and the randomly chosen point to delete was colored red (i.e., a point inside h^+). Now, the Radon point might be in h^+ only if we picked two (or more) red points in Q, by Lemma 3.2 (C). As such, we can guarantee that progress is made when at most one of the $d+2$ points chosen at random is red, and the point deleted from Q was red. See Figure 3.2.

Roughly speaking, the problem is to upper bound the number of iterations needed until all points are colored blue. We now formally state and analyze this problem in a generalized setting.

3.2.3 Radon's urn

In the ***Radon's urn*** game there are r red balls, and $b = n - r$ blue balls in an urn, and there is a parameter t. An iteration of the game goes as follows:

(A) The player picks a random ball, marks it for deletion, and returns it to the urn.

(B) The player picks a sample S of t balls (with replacement—which implies that we might have several copies of the same ball in the sample) from the urn.

(C) If at least two of the balls in the sample S are red, the player inserts a new red ball into the urn. Otherwise, the player inserts a new blue ball.

(D) The player returns the sample to the urn.

(E) Finally, the player removes the ball marked for deletion from the urn.

Note that in each stage of the game, the number of balls in the urn remains the same. We are interested in how many rounds of the game one has to play till there are no red balls in the urn with high probability. Here, the initial value of r is going to be relatively small compared to n.

Importantly, the setup described matches the setting discussed in Section 3.2.2, with $t = d + 2$ and initially $r = f(d)n$.

3.2.4 Analysis

Let $P(r)$ be the probability of picking two or more red balls into the sample, assuming that there are r red balls in the urn. We have that

$$P(r) = \sum_{i=2}^{t} \binom{t}{i} \left(\frac{r}{n}\right)^i \left(1 - \frac{r}{n}\right)^{t-i} \leq \binom{t}{2} \left(\frac{r}{n}\right)^2 \leq \frac{t^2}{2} \left(\frac{r}{n}\right)^2.$$

(For the proof of the first inequality, see [80].) Note, that $P(r) \leq 1/2$ if $n \geq tr$. Let $P_+(r)$ be the probability that the number of red balls increased in this iteration. For this to happen, at least two red balls had to be in the sample, and the deleted ball must be blue. Let $P_-(r)$ be the probability that the number of red balls decreases—the player needs to pick strictly less than two red balls in the sample, and delete a red ball. This implies

$$P_+(r) = P(r)(1 - r/n) \leq P(r) \qquad \text{and} \qquad P_-(r) = (1 - P(r))(r/n).$$

The probability for a change in the number of red balls at this iteration is

$$P_\pm(r) = P_+(r) + P_-(r) = P(r)(1 - r/n) + (1 - P(r))(r/n) = (1 - 2r/n)P(r) + r/n.$$

Lemma 3.3. *Let $\xi \in (0, 1/4)$ be a parameter. If $r \leq R$, then*

$$P_+(r) \leq \frac{1/2 - \xi}{1/2 + \xi} P_-(r), \text{ where } R = (1 - 2\xi)\frac{n}{t^2}.$$

Proof: We solve $P(r) \leq \frac{1-2\xi}{1+2\xi} P_-(r) \leq (1-2\xi)P_-(r)$, which is equivalent to

$$P(r) \leq \big(1-P(r)\big)\frac{r}{n}(1-2\xi) \iff \frac{n}{(1-2\xi)r}P(r) \leq 1-P(r) \iff P(r) \leq \frac{(1-2\xi)r}{(1-2\xi)r+n}.$$

Now the last inequality holds if $\frac{t^2r^2}{2n^2} \leq \frac{(1-2\xi)r}{2n}$ since

$$P(r) \leq \frac{t^2r^2}{2n^2} \leq \frac{(1-2\xi)r}{2n} \leq \frac{(1-2\xi)r}{(1-2\xi)r+n}.$$

Namely, the desired condition holds when $r \leq (1-2\xi)n/t^2$. QED.

We conclude this section by making some remarks regarding the specific problem we need to solve and the roles of the parameters introduced above.

The question Let $\vartheta \in (0,1)$ and $r_0 = (1-\vartheta)R$ be the number of red balls in the urn at the start of the game (note that both r_0 and R are functions of n and t). Let $\varphi > 0$ be a parameter. The key question is the following: How large does n need to be so that if we start with r_0 red balls (and $n - r_0$ blue balls), the game ends with all balls being blue with probability $\geq 1-\varphi$?

The game as a random walk An iteration of the game where the number of red balls changes is an ***effective*** iteration. Considering only the effective iterations, this can be interpreted as a random walk starting at $X_0 = (1-\vartheta)R$ and at every iteration either decreasing the value by one with probability at least $1/2+\xi$, and increasing the value with probability at most $1/2-\xi$, by Lemma 3.3. This walk ends when either it reaches 0 or R. If the walks reaches R, then we consider the process to have failed (see the next remark). Otherwise if the walk reaches 0, then there are no red balls in the urn as desired, and the process succeeds.

The ratio R/n The value R/n is an upper bound on the ratio of red balls that the urn can have and still, with good probability, end with zero red balls at the end of the game. If this ratio is violated anytime during the game, then the urn might end up consisting of only red balls. We want to start the game with an urn initially having close to R red balls, but still end up with an entirely blue urn with sufficiently high probability.

The parameter ξ The parameter ξ controls the bias in the random walk. As ξ goes to zero, the random walk becomes less unbalanced as it moves left with probability $1/2+\xi$ and move right with probability $1/2-\xi$. Naturally, if the walk has a near-equal chance to move left or right, then we show that the number of iterations until success will depend polynomially on $1/\xi$. In the context of our main application to centerpoints, this parameter is key to obtaining centerpoints with approximation quality arbitrarily close to $1/(d+2)^2$.

3.2.5 Analyzing the related walk

Consider the random walk that starts at $Y_0 = (1-\vartheta)R$. At the ith iteration, $Y_i = Y_{i-1} - 1$ with probability $1/2 + \xi$, and $Y_i = Y_{i-1} + 1$ with probability $1/2 - \xi$. Let $\mathcal{Y} = Y_1, Y_2, \ldots$ be the resulting random walk which can be shown to stochastically dominate the walk $X_0, X_1, \ldots$. Formally, we mean that for any integer k and each i, $\mathbf{Pr}[X_i > k] \leq \mathbf{Pr}[Y_i > k]$. This fact can be proved using the coupling technique, see [56, Section 7.4]. Hence, it suffices to analyze the behavior of $\mathcal{Y}$.

For large values of ξ, the walk is strongly biased towards going to 0, and as such it does not hang around too long before moving on, as testified by the following Lemma.

Lemma 3.4. *Let I be any integer number, and let $\varphi > 0$ be a parameter. The number of times the random walk $\mathcal{Y}$ visits I is at most $\frac{1}{4\xi^2} \ln \frac{1}{4\xi^2\varphi}$ times, and this holds with probability $\geq 1 - \varphi$.*

Proof: Let τ be the first index such that $Y_\tau = I$. The probability that $Y_{\tau+2i} = I$ is

$$p_{\tau+i} = \binom{2i}{i} \left(\frac{1}{2} - \xi\right)^i \left(\frac{1}{2} + \xi\right)^i \leq 2^{2i} \left(\frac{1}{4} - \xi^2\right)^i = \left(1 - 4\xi^2\right)^i.$$

The probability that the walk would visit I again after time $\tau + u$ is bounded by $\nu(u) = \sum_{i=u}^{\infty} p_{\tau+i} \leq p_{\tau+u} / \left(4\xi^2\right)$. We want to choose u so that $\nu(u) \leq \varphi$. In particular,

$$\nu(u) \leq (1 - 4\xi^2)^u / (4\xi^2) \leq \exp\Big(-4\xi^2 u\Big) / (4\xi^2) \leq \varphi$$

for $u \geq \frac{1}{4\xi^2} \ln \frac{1}{4\xi^2\varphi}$. Thus the walk might visit I during the first u iterations, but with probability $\geq 1 - \varphi$ it never visits it again. We conclude, that with probability $\geq 1 - \varphi$, the walk visits the value I at most u times. QED.

We next bound the probability that the walk fails.

Lemma 3.5. *Let $\varphi > 0$ be a parameter. If $R \geq \frac{1}{2\xi\vartheta} \ln \frac{1}{4\xi^2\varphi}$ then the probability that the random walk $\mathcal{Y}$ ever visits R (and thus fails) is bounded by φ.*

Proof: Since the walk starts at $(1-\vartheta)R$, the first time it can arrive to R is at time ϑR. In particular, the probability that $Y_{\vartheta R+2i} = R$ is

$$\begin{aligned} p'_i &= \binom{\vartheta R + 2i}{\vartheta R + i} \left(\frac{1}{2} - \xi\right)^{\vartheta R+i} \left(\frac{1}{2} + \xi\right)^i \leq 2^{\vartheta R+2i} \left(\frac{1}{2} - \xi\right)^{\vartheta R+i} \left(\frac{1}{2} + \xi\right)^i \\ &= (1 - 2\xi)^{\vartheta R+i} (1 + 2\xi)^i = (1 - 2\xi)^{\vartheta R} \left(1 - 4\xi^2\right)^i. \end{aligned}$$

It follows that the probability of failure is bounded by

$$p = \sum_{i=0}^{\infty} p'_i \leq (1 - 2\xi)^{\vartheta R} / \left(4\xi^2\right) \leq \exp(-2\xi\vartheta R) / (4\xi^2).$$

In particular, we require that $p \leq \exp(-2\xi\vartheta R)/\left(4\xi^2\right) \leq \varphi$, which holds for $R \geq \frac{1}{2\xi\vartheta} \ln \frac{1}{4\xi^2\varphi}$. QED.

3.2.6 Back to the urn

The next Lemma we prove will require the existence of concentration bounds for sums of geometric variables. Such concentration bounds can be derived from Chernoff-type inequalities [93].

Lemma 3.6 ([93, Theorem 2.3]). *Let $X_1, \ldots, X_m$ be m independent random variables with geometric distribution with parameter $1/2$. For any $\lambda \geq 1$, we have $\mathbf{Pr}[\sum_{i=1}^{m} X_i \geq \lambda \cdot 2m] \leq \lambda^{-1} 2^{-2m(\lambda - 1 - \ln \lambda)}$. For $\lambda = 4$, this becomes $\mathbf{Pr}[\sum_{i=1}^{m} X_i \geq 8m] \leq 2^{-2m(4-1-\ln 4)}/4 \leq 2^{-3m} \leq e^{-3m/2}$, for $m \geq 10$.*

The number of red balls in the urn is stochastically dominated by the random walk above. The challenge is that the number of iterations one has to play before an effective iteration happens (which corresponds to one step of the above walk) depends on the number of red balls, and behaves like the coupon collector problem. Specifically, if there are $r \leq R$ red balls in the urn, then the probability for an effective step is $P_{\pm}(r) \geq (1 - P(r))(r/n) \geq r/2n$, as $P(r) \leq 1/2$.[1] This implies that in expectation, one has to wait at most $2n/r$ iterations before an effective iteration happens.

Lemma 3.7. *Let $\varphi > 0$ and $\xi \in (0, 1/4)$ be parameters. For any value $r \leq R$, the urn spends at most $s(r) = O\big((n/r)\xi^{-2} \ln(\xi^{-1}\varphi^{-1})\big)$ regular iterations, throughout the game, having exactly r balls in it, with probability $\geq 1 - \varphi$.*

Proof: Consider an iteration in which the urn has r red balls. Let Δ be the random variable who value is equal to the number of iterations played until the amount of red balls change. By the above discussion, we have that $\mathbb{E}[\Delta] \leq 2n/r$. In particular, Markov's inequality implies that $\mathbf{Pr}[\Delta \geq 4n/r] \leq \mathbb{E}[\Delta]/(4n/r) \leq 1/2$.

Call a collection of $4n/r$ consecutive iterations having exactly r red balls a ***block*** and define $X = \lceil \Delta/(4n/r) \rceil$ to be the random variable which is equal to the number of blocks until there is a change in the number of red balls. A block is a ***success*** if one of its iterations changes the amount of red balls. Otherwise the block is a ***failure***. As the chance of success is at least $1/2$ for each block, and X measures the number of trials until success, X forms a geometric distribution with parameter $1/2$.

Lemma 3.4 implies that the number of effective iterations with r red balls is at most $m = \ln(1/(4\xi^2\varphi))/4\xi^2$. For each of the m visits, we define an associated random variable X_i with the same geometric distribution described above. One can verify that $\mu = \mathbb{E}[\sum_i X_i] \leq 2m$. We now prove that $\sum_i X_i$ is not much larger than μ with good probability. Indeed, applying Lemma 3.6 directly implies that

$$\mathbf{Pr}\left[\sum_{i=1}^{m} X_i \geq 8m\right] \leq \exp(-3m/2) = \exp\Big(3/(8\xi^2)\ln(4\xi^2\varphi)\Big) = (4\xi^2\varphi)^{3/(8\xi^2)} \leq \varphi,$$

[1] Recall that $P(r) \leq 1/2$ when $r \leq n/t$. The latter inequality always holds during the random process, since $r \leq R = (1-2\xi)n/t^2 \leq n/t$, by Lemma 3.3.

where the last inequality follows since $\xi < 1/4$.

We conclude that the number of iterations the urn spends having exactly r balls in it is bounded by $(2n/r)\sum_i X_i = O((n/r)\xi^{-2}\ln(\xi^{-1}\varphi^{-1}))$ with probability at least $1-\varphi$. QED.

Lemma 3.8. *Let $\varphi > 0$ and $\vartheta \in (0,1)$ be parameters, and assume that $n = \Omega\left(\frac{t^2}{\xi\vartheta}\ln\frac{1}{\xi\varphi}\right)$. The total number of regular iterations one has to play till the urn contains only blue balls, is $O\big(n\xi^{-2}\log n\log(n\xi^{-1}\varphi^{-1})\big)$, and this bound holds with probability $\geq 1-\varphi$.*

Proof: The bound on the number of steps follows readily by summing up the bound of Lemma 3.7, for $r = 1,\ldots,R$. Specifically, we apply this Lemma with failure probability $\varphi/2n$, to bound the sum $\sum_{r=1}^{R} s(r) = O\big(n\xi^{-2}\log n\log(n\xi^{-1}\varphi^{-1})\big)$. The probability of failure is at most $R\varphi/2n \leq \varphi/2$.

The other reason for a failure is that the urn reaches a state where it has R red balls. In particular, using Lemma 3.5 to bound this probability by $\varphi/2$, requires that $R \geq \frac{1}{2\xi\vartheta}\ln\frac{1}{2\xi^2\varphi}$. By the value specified in Lemma 3.3, this requires $(1-2\xi)\frac{n}{t^2} \geq \frac{1}{2\xi\vartheta}\ln\frac{1}{2\xi^2\varphi}$, which holds for the value of n stated. QED.

3.2.7 Back to approximating the centerpoint

Equipped with Lemma 3.8, we now return our focus to computing an $f(d)$-centerpoint for a given point set P. We will apply Lemma 3.8 to each halfspace h^+ that contains exactly $f(d)n$ points of P. To apply the Radon's urn analysis above, we require that the number of initial red balls is $f(d)n = (1-\vartheta)R$, where $\vartheta \in (0,1)$. By Lemma 3.3, $R = (2/3)n/t^2$ (by choosing, say, $\xi = 1/6$). Recall that $t = d+2$, which implies that

$$(1-\vartheta)\frac{2n}{3(d+2)^2} = f(d)n \iff f(d) = \frac{2(1-\vartheta)}{3(d+2)^2} \geq \frac{1-\vartheta}{2(d+2)^2} \tag{3.1}$$

We now apply the Radon's urn analysis to argue that after a sufficient number of iterations, all of the points of Q are outside h^+.

Playing many Radon's urn games in parallel Consider all halfspaces that might be of interest. To this end, consider any hyperplane passing through d points of P, and translate it so that it contains on one of its sides exactly $f(d)n$ points (naturally, there are two such translations). Each such hyperplane thus defines two natural halfspaces. Let H be the resulting set of halfspaces. Observe that $|H| \leq 2\binom{n}{d} \leq 2(ne/d)^d$. If Q does not contain any point in any of the halfspaces of H then all its points are $f(d)$-centerpoints. In particular, one can think about this as playing $|H|$ parallel Radon's urn games. We want the algorithm to succeed with probability $\geq 1-\varphi$. Setting the probability of failure for each halfplane of H to be $\varphi/|H|$, and by Lemma 3.8, we have that all of these halfspaces are empty after playing

$$O\Big(n\log n\log(n|H|\varphi^{-1})\Big) = O\big(dn\log n\log(n/\varphi)\big)$$

iterations, with probability of success being $1 - |H|(\varphi/|H|) = 1 - \varphi$ by the union bound. Using Lemma 3.8 requires that $n = \Omega\big(t^2\vartheta^{-1}\ln(|H|/\varphi)\big) = \Omega\big(\vartheta^{-1}d^3\ln n + \vartheta^{-1}d^2\ln\varphi^{-1}\big)$ which holds for $n = \Omega(\vartheta^{-1}d^3\ln d + \vartheta^{-1}d^2\ln\varphi^{-1})$.

By Lemma 3.2 (A), computing a Radon point for $d+2$ points in $\mathbb{R}^d$ can be done in $O(d^3)$ time, and hence we obtain the following result. A proof of the Lemma involving the parameter ξ is located in [80].

Lemma 3.9 ([80]). *Let $\varphi > 0$ and $\vartheta \in (0,1)$ be parameters, and let P be a set of $n = \Omega(\vartheta^{-1}d^3\ln d + \vartheta^{-1}d^2\ln\varphi^{-1})$ points in $\mathbb{R}^d$. Let $\alpha = \frac{1-\vartheta}{2(d+2)^2}$. Then, one can compute a α-centerpoint of P via a randomized algorithm. The running time of the randomized algorithm is $O\big(d^3\cdot dn\log n\log(n/\varphi)\big) = O(d^4 n\log n\log(n/\varphi))$, and it succeeds with probability $\geq 1-\varphi$.*

Removing the dependency on n We now remove the dependency on n in the running time of Lemma 3.9 by performing a random sampling step beforehand via the use of relative approximations. See Section 4.2.1$_{\text{p43}}$ for the relevant definitions, specifically Definition 4.4$_{\text{p43}}$ and Theorem 4.3$_{\text{p44}}$. To start, we need the following observation.

Lemma 3.10. *Let P be a set of n points in $\mathbb{R}^d$. Suppose that $S \subseteq P$ is a relative $(p,\hat{\varepsilon})$-approximation for halfspaces. Let $c \in \mathbb{R}^d$ be an α-centerpoint for S. If $\alpha \geq p$, then c is a $(1-\hat{\varepsilon})p$-centerpoint for P.*

Proof: Let h^+ be any halfspace containing c. As c is an α-centerpoint for S and S is a relative $(p,\hat{\varepsilon})$-approximation for P, we have that

$$\overline{m}\big(h^+\big) \geq \overline{s}\big(h^+\big) - \hat{\varepsilon}p \geq \alpha - \hat{\varepsilon}p \geq (1-\hat{\varepsilon})p.$$

As such, c is a $(1-\hat{\varepsilon})p$-centerpoint for P. QED.

Theorem 3.1. *Given a set P of n points in $\mathbb{R}^d$, a parameter φ, and a constant $\vartheta \in (0,1)$, one can compute a $\frac{1-\vartheta}{2(d+2)^2}$-centerpoint of P. The running time of the algorithm is $O\Big(\vartheta^{-3}d^7\log^3 d\log^3\varphi^{-1}\Big)$ together with a random sampling step, and it succeeds with probability $\geq 1-\varphi$.*

Proof: The idea is to pick a random sample S from P that is a relative $(\rho,\vartheta/8)$-approximation for halfspaces, where $\rho = 1/(10d^2)$. This range space has VC dimension $d+1$, and by Theorem 4.3, a sample of size

$$\mu = O\big(\rho^{-1}\vartheta^{-2}(d\log\rho^{-1} + \log\varphi^{-1})\big) = O\big(d^2\vartheta^{-2}(d\log d + \log\varphi^{-1})\big)$$

is a relative $(\rho,\vartheta/8)$-approximation.

Running the algorithm of Lemma 3.9 on S with $\vartheta/8$ yields a τ-centerpoint c of S, where $\tau = \frac{1-\vartheta/8}{2(d+2)^2}$. Note that $\tau \geq \rho$ for $d \geq 2$, and so Lemma 3.10 implies that c is an α-centerpoint for P, where

$$\alpha = (1-\vartheta/8)\tau = \frac{(1-\vartheta/8)^2}{2(d+2)^2} \geq \frac{1-\vartheta}{2(d+2)^2}.$$

By Lemma 3.9, the running time of the resulting algorithm is

$$O(d^4\mu \log \mu \log(\mu/\varphi)) = O\big(\vartheta^{-2} d^7 \log^2 \vartheta^{-1} \log^3 d \log^3 \varphi^{-1}\big) = O\big(\vartheta^{-3} d^7 \log^3 d \log^3 \varphi^{-1}\big). \qquad \text{QED.}$$

Repeating the above calculations with the parameter ξ allows us to compute centerpoints of quality arbitrarily close to $1/(d+2)^2$. In order to preserve that the random walk succeeds with probability at least $1-\varphi$, the sample size n must depend on the parameter ξ.

Theorem 3.2 ([80]). *Given a set P of n points in $\mathbb{R}^d$, a parameter φ, and a constant $\gamma \in (0,1)$, one can compute a $\frac{1-\gamma}{(d+2)^2}$-centerpoint of P. The running time of the algorithm is*

$$O\Big(\gamma^{-4} d^7 \log^3 d \, \log^3\Big(\gamma^{-1}\varphi^{-1}\Big)\Big),$$

together with a random sampling step, and it succeeds with probability $\geq 1-\varphi$.

Remark 3.2. The above compares favorably to the result of [49, Corollary 3]—they get a running time of $O(d^9 \log d + d^8 \log^2 \varphi^{-1})$, which is slower by roughly a factor of d^2, and computes a $\frac{1}{4.08(d+2)(d+1)}$-centerpoint of P—the quality of the centerpoint is roughly worse by a factor of four.

3.3 APPLICATION: LOWER-BOUNDING A CONVEX FUNCTION

In this section we discuss a straightforward application of the improved centerpoint algorithm. This particular application (and future applications) revolve around the idea of *oracle access*. Suppose we are interested in lower bounding a convex function given an oracle to compute it's gradient.

Definition 3.3. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex function. For a number $c \in \mathbb{R}$, define the ***level set of f*** as $\mathcal{L}_f(c) = \Big\{p \in \mathbb{R}^d \mid f(p) \leq c\Big\}$. If f is a convex function, then $\mathcal{L}_f(c)$ is a convex set for all $c \in \mathbb{R}$.

Definition 3.4. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex (and possibly non-differentiable) function. For a point $p \in \mathbb{R}^d$, a vector $v \in \mathbb{R}^d$ is a ***subgradient*** of f at p if for all $q \in \mathbb{R}^d$, $f(q) \geq f(p) + \langle v, q-p\rangle$. The ***subdifferential*** of f at $p \in \mathbb{R}^d$, denoted by $\partial f(p)$, is the set of all subgradients $v \in \mathbb{R}^d$ of f at p.

It is well known that when the domain of f is $\mathbb{R}^d$ and f is a convex function, then $\partial f(p)$ is a non-empty set of all $p \in \mathbb{R}^d$ (for example, see [65, Chapter 3]).

Theorem 3.3. *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex (possibly non-differentiable) function and P a set of n points in $\mathbb{R}^d$. Assume that one has access to an oracle which given $p \in \mathbb{R}^d$ returns an arbitrary element in the subdifferential $\partial f(p)$. With $O(d^2 \log n)$ queries to the oracle, one can compute a point $q \in \mathbb{R}^d$ (not necessarily in P) such that $f(q) \leq \min_{p \in P} f(p)$.*

Proof: Let $P_1 = P$, and $P_i \subseteq P$ denote the set of remaining points at the beginning of the ith iteration. In iteration i, for some constant $c > 0$, compute a (c/d^2)-centerpoint c_i of P_i using Theorem 3.1 in time $O(d^7 \log^3 d)$ with success probability $1/2$. Define $C_i = \mathcal{L}_f(f(c_i))$. We now use the oracle to obtain subgradient vector $v \in \partial f(c_i)$. Given v, let h_i be the d-dimensional hyperplane orthogonal to v which passes through c_i. By construction, h_i is tangent to C_i at c_i. Let h_i^+ be the halfspace formed from h_i which contains the interior of C_i. If $|h_i^- \cap P_i| \geq c\,|P_i|\,/d^2$, then such an iteration is successful and we set $P_{i+1} = P_i \setminus (h_i^- \cap P_i)$ and continue to iteration $i+1$. Otherwise the iteration has failed and we repeat the ith iteration. This procedure is repeated until we reach an iteration τ in which $|P_\tau|$ is of constant size. At this stage, the algorithm returns the point achieving the minimum of $\min_{1 \leq i \leq \tau} f(c_i)$ and $\min_{p \in P_\tau} f(p)$. Because f is convex, the algorithm returns a point q such that $f(q) \leq f(p)$ for all $p \in P$.

As for the number of queries, note that in each iteration the expected number of centerpoint calculations (and thus queries) until a successful iteration is $O(1)$. It remains to bound the number of successful iterations. In each successful iteration, a c/d^2-fraction of points are discarded. Therefore there are at most τ iterations, for which τ is the smallest number with $(1 - c/d^2)^\tau n$ smaller than some constant. This implies $\tau = O(d^2 \log n)$. QED.

4 Functional nets, center nets, and other variants

> *Reductio ad absurdum, which Euclid loved so much, is one of a mathematician's finest weapons. It is a far finer gambit than any chess play: a chess player may offer the sacrifice of a pawn or even a piece, but a mathematician offers the game.*
>
> **— Godfrey H. Hardy**
> (A Mathematician's Apology)

In this chapter we explore many different types of nets, including nets for more complex ranges, such as arbitrary convex bodies. In particular, we use the improved centerpoint algorithms derived in Chapter 3 to efficiently construct these nets, and explore the connection between centerpoints and weak ε-nets. We assume the reader is already familiar with the concept of range spaces, VC dimension, and related results. A brief summary is provided in Section 4.2.

4.1 BACKGROUND

Range spaces and ε-nets A ***range space*** is a pair $\mathsf{S} = (\mathcal{U}, \mathcal{R})$, where $\mathcal{U}$ is the ***ground set*** (finite or infinite) and $\mathcal{R}$ is a (finite or infinite) family of subsets of $\mathcal{U}$. The elements of $\mathcal{R}$ are ***ranges***.

Suppose that $\mathcal{U}$ is a finite set. For a parameter $\varepsilon \in (0,1)$, a subset $\mathsf{S} \subseteq \mathcal{U}$ is an ***ε-net*** for the range space S, if for every range $\mathsf{r} \in \mathcal{R}$ with $|\mathsf{r} \cap \mathcal{U}| \geq \varepsilon |\mathcal{U}|$ has $\mathsf{r} \cap \mathsf{S} \neq \varnothing$. The ε-net theorem of Haussler and Welzl [88] implies the existence of ε-nets of size $O(\delta \varepsilon^{-1} \log \varepsilon^{-1})$, where δ is the VC dimension of the range space S. The use of ε-nets is widespread in computational geometry [78, 112]. We formalize these definitions and make them concrete in Section 4.2.

Weak ε-nets Consider the range space $(P, \mathcal{C})$, where $\mathcal{C}$ is the collection of all compact convex bodies in $\mathbb{R}^d$ and $P \subset \mathbb{R}^d$ is a point set of size n. This range space has infinite VC dimension—the standard ε-net constructions do not work for this range space. The notion of ***weak ε-nets*** bypasses this issue by allowing the net S to use points outside of P. Specifically, any convex body C that contains at least εn points of P must contain a point of S. The first construction of weak ε-nets in the plane was due to Bárány et al. [16], where they construct nets of size $O(1/\varepsilon^{1026})$. For all $d \geq 1$, Alon et al. [6] were the first to construct weak

ε-nets in $\mathbb{R}^d$ whose size was bounded in terms of $1/\varepsilon$ and d. In 2004, Matoušek and Wagner [114] gave a simpler construction of weak ε-nets of size $O_d(\varepsilon^{-d}\log^{f(d)}\varepsilon^{-1})$, where $f(d) = O(d^2\log d)$. Recently, Rubin [137, 138] gave an improved bound, showing existence of weak ε-nets of size $O_d(\varepsilon^{-(d-0.5+\alpha)})$ for arbitrarily small $\alpha > 0$. For more detailed history of the problem, see the introduction of Rubin [137, 138]. As for a lower bound, Bukh et al. [28] gave constructions of point sets for which any weak ε-net must have size $\Omega_d(\varepsilon^{-1}\log^{d-1}\varepsilon^{-1})$. Closing this gap remains a major open problem. See [126] for a recent survey of ε-nets and related concepts. See also the recent work by Rok and Smorodinsky [135] and references therein.

Basis of weak ε-nets Mustafa and Ray [125] showed that one can pick a random sample S of size $c_d\varepsilon^{-1}\log\varepsilon^{-1}$ from P, and then compute a weak ε-net for P directly from S, showing that the size of the support needed to compute a weak ε-net is (roughly) the size of a regular ε-net. Unfortunately, the constant in their sample $c_d = O\big(d^d(\log d)^{cd^3\log d}\big)$ is doubly exponential in the dimension. This constant c_d is related to the $((d+1)^2, d+1)$-Hadwiger-Debrunner number (the best known upper bounds on (p,q)-Hadwiger-Debrunner numbers can be found in [98, 99]).

In particular, all current results about weak ε-nets suffer from the "curse of dimensionality" and have constants that are at least doubly exponential in the dimension.

(k,ε)-nets and uniform measure A natural extension of weak ε-nets is to allow the net S to contain other geometric objects. Given a collection of n points $P \subset \mathbb{R}^d$ and a parameter $0 \le k < d$, we define a (weak) (k,ε)-net to be a collection of k-flats S such that if C is a convex body containing at least εn points of P, then there exists a k-flat in S intersecting C. Note that $(0,\varepsilon)$-nets are exactly weak ε-nets.

In general, one would expect that as k increases, the size of the (k,ε)-net shrinks. For example, a $(1,\varepsilon)$-net for a collection of points in $\mathbb{R}^3$ can be constructed by projecting the points down onto the xy-plane and applying Rubin's construction in the plane to obtain a weak ε-net S of size $O(\varepsilon^{-(3/2+\alpha)})$ [137]. Lifting S up back into three dimensions results in a $(1,\varepsilon)$-net of the same size, which is smaller than the best known weak ε-net size in $\mathbb{R}^3$ [114, 137, 138]. However, one might expect that a $(1,\varepsilon)$-net of even smaller size is possible in $\mathbb{R}^3$, as this construction uses a set of parallel lines (i.e., one would expect the lines in an optimal net to be arbitrarily oriented).

As part of this chapter, we study an even simpler version of the problem, where the ground set is the hypercube $B = [0,1]^d$. In particular, for $\varepsilon \in (0,1)$ and $0 \le k < d$, we are interested in computing the smallest set K of k-flats, such that if C is a convex body with $\mathsf{vol}(C \cap B) \ge \varepsilon$, then there is a k-flat in K which intersects C. For sake of exposition, throughout the rest of the paper we refer to this set K as a ***(k,ε)-net for volume measure***. We note that $[0,1]^d$ can be replaced with any arbitrary compact convex body in the definition (the size of the (k,ε)-net increases by a factor depending on d).

4.1.1 Our results

Let P be a set of n points in $\mathbb{R}^d$. We suggest some alternatives to weak ε-nets, and obtain some related results:

(A) **Functional nets.** Let $C \subseteq \mathbb{R}^d$ be a convex body. Suppose we are only given access to C via a separation oracle: given a query point z, the oracle either returns that z is in C, or alternatively, the oracle returns a hyperplane separating z and C. We show that a random sample of size

$$O\Big(\varepsilon^{-1} d^3 \log d \log^3 \varepsilon^{-1} + \varepsilon^{-1} \log \varphi^{-1}\Big) = \widetilde{O}\Big(d^3/\varepsilon\Big),$$

with probability $\geq 1 - \varphi$, can be used to decide if a query convex body C contains less than an ε-fraction of the points of P (in which case we say that C is *ε-light*). Formally, the algorithm, using only the sample, performs $O(d^2 \log \varepsilon^{-1})$ oracle queries—if any of the query points generated stabs C, then C is considered as (potentially) containing more than εn points. Alternatively, if all the queries missed C, then C contains less than εn points of P. The query points can be computed in polynomial time, and we emphasize that the dependency in the running time and sample size are polynomial in ε and d. See Theorem 4.4. Within the context of weak ε-nets, given that current constructions of weak ε-nets have size of the form $1/\varepsilon^{O(d)}$, this result can be viewed as mitigating the curse of dimensionality—the sample size avoids an exponential dependency on d.

(B) **Center nets.** Using the above, one can also construct a weak ε-net directly from such a sample—this improves over the result of Mustafa and Ray [125] as far as the dependency on the dimension is concerned, and is described in Lemma 4.7.

Surprisingly, by using ideas from Theorem 4.4 one can get a stronger form of a weak ε-net, which we refer to as an *(ε, α)-center net*. Here $\alpha = \Omega(1/(d \log \varepsilon^{-1}))$ and one can compute a set $\mathcal{W}$ of size (roughly) $\widetilde{O}_d\big(\varepsilon^{-O(d^2)}\big)$, such that if a convex body C containing $\geq \varepsilon n$ points of P, then $\mathcal{W}$ contains a point z which is an α-centerpoint (Definition 3.1$_{\text{p29}}$) of $C \cap P$. Namely, the net contains a point that stabs C in the "middle" of the point set $C \cap P$. See Theorem 4.5.

(C) **Explicit constructions of (k, ε)-nets.** In Lemma 4.10, we show that any (k, ε)-net for volume measure must have size $\Omega_d(1/\varepsilon^{1-k/d})$. Perhaps surprisingly, we give a relatively simple construction of (k, ε)-nets for volume measure of size $O_d(1/\varepsilon^{1-k/d})$ for $k \geq 1$ (Theorem 4.6). For $k = 0$, we obtain nets of size $O_d((1/\varepsilon)\log^{d-1}(1/\varepsilon))$ (Theorem 4.8). Importantly, both constructions are deterministic and explicit (see the discussion below).

As far as the authors are aware, this particular problem we study has not been addressed before. The only related result known is the existence of explicit constructions of $(0, \varepsilon)$-nets for volume measure for axis parallel boxes in $\mathbb{R}^d$, and is briefly mentioned in [28]. In this case, one can construct a $(0, \varepsilon)$-net for volume measure of size $O_d(1/\varepsilon)$ using Van der Corput sets in two dimensions, and

Halton-Hammersely sets in higher dimensions.

Deterministic vs. explicit constructions of ε-nets For the regular concept of ε-nets, there are known deterministic constructions. They work by repeatedly halving the input point set, using deterministic discrepancy constructions, until the set is of the desired size [41, 111]. On the one hand, for our setting (i.e., the measure is uniform volume on the unit hypercube) it is not clear what the generated ε-net is without running this construction algorithm outright. On the other hand, we develop a construction of weak ε-nets—for uniform volume measure over the hypercube for ellipsoids—which are much simpler and are explicit; one can easily compute the ith point in this net using polylogarithmic space.

4.2 PRELIMINARIES

4.2.1 Ranges spaces, VC dimension, samples and nets

The following is a quick survey of (standard) known results about ε-nets, ε-samples, and relative approximations [78].

Definition 4.1. A ***range space*** S is a pair $(\widehat{\mathsf{X}}, \mathcal{R})$, where $\widehat{\mathsf{X}}$ is a ***ground set*** (finite or infinite) and $\mathcal{R}$ is a (finite or infinite) family of subsets of $\widehat{\mathsf{X}}$. The elements of $\widehat{\mathsf{X}}$ are ***points*** and the elements of $\mathcal{R}$ are ***ranges***.

For technical reasons, it will be easier to consider a finite subset $\mathsf{X} \subset \widehat{\mathsf{X}}$ as the underlining ground set.

Definition 4.2. Let $\mathsf{S} = (\widehat{\mathsf{X}}, \mathcal{R})$ be a range space, and let X be a finite (fixed) subset of $\widehat{\mathsf{X}}$. For a range $\mathsf{r} \in \mathcal{R}$, its ***measure*** is the quantity $\overline{m}(\mathsf{r}) = |\mathsf{r} \cap \mathsf{X}|/|\mathsf{X}|$. For a subset $S \subseteq \mathsf{X}$, its ***estimate*** of $\overline{m}(\mathsf{r})$, for $\mathsf{r} \in \mathcal{R}$, is the quantity $\overline{s}(\mathsf{r}) = |\mathsf{r} \cap S|/|S|$.

Definition 4.3. Let $\mathsf{S} = (\widehat{\mathsf{X}}, \mathcal{R})$ be a range space. For $Y \subseteq \widehat{\mathsf{X}}$, let $\mathcal{R}_{|Y} = \left\{ \mathsf{r} \cap Y \mid \mathsf{r} \in \mathcal{R} \right\}$ denote the ***projection*** of $\mathcal{R}$ on Y. The range space S projected to Y is $\mathsf{S}_{|Y} = \left(Y, \mathcal{R}_{|Y}\right)$. If $\mathcal{R}_{|Y}$ contains all subsets of Y (i.e., if Y is finite, we have $\left|\mathcal{R}_{|Y}\right| = 2^{|Y|}$), then Y is ***shattered*** by $\mathcal{R}$ (or equivalently Y is shattered by S).

The VC ***dimension*** of S, denoted by $\dim_{\mathsf{VC}}(\mathsf{S})$, is the maximum cardinality of a shattered subset of $\widehat{\mathsf{X}}$. If there are arbitrarily large shattered subsets, then $\dim_{\mathsf{VC}}(S) = \infty$.

Definition 4.4. Let $\mathsf{S} = (\widehat{\mathsf{X}}, \mathcal{R})$ be a range space, and let X be a finite subset of $\widehat{\mathsf{X}}$. For $0 \leq \varepsilon \leq 1$, a subset $S \subseteq \mathsf{X}$ is an ***ε-sample*** for X if for any range $\mathsf{r} \in \mathcal{R}$, we have $|\overline{m}(\mathsf{r}) - \overline{s}(\mathsf{r})| \leq \varepsilon$, see Definition 4.2. Similarly, a set $S \subseteq \mathsf{X}$ is an ***ε-net*** for X if for any range $\mathsf{r} \in \mathcal{R}$, if $\overline{m}(\mathsf{r}) \geq \varepsilon$ (i.e., $|\mathsf{r} \cap \mathsf{X}| \geq \varepsilon\,|\mathsf{X}|$), then r contains at least one point of S (i.e., $\mathsf{r} \cap S \neq \emptyset$).

A generalization of both concepts is *relative approximation*. Let $p, \hat{\varepsilon} > 0$ be two fixed constants. A ***relative $(p, \hat{\varepsilon})$-approximation*** is a subset $S \subseteq \mathsf{X}$ that satisfies $(1 - \hat{\varepsilon})\overline{m}(\mathsf{r}) \leq \overline{s}(\mathsf{r}) \leq (1 + \hat{\varepsilon})\overline{m}(\mathsf{r})$, for any $\mathsf{r} \in \mathcal{R}$ such that $\overline{m}(\mathsf{r}) \geq p$. If $\overline{m}(\mathsf{r}) < p$ then the requirement is that $|\overline{s}(\mathsf{r}) - \overline{m}(\mathsf{r})| \leq \hat{\varepsilon} p$.

Theorem 4.1 (ε-net theorem, [88]). *Let $(\widehat{\mathsf{X}}, \mathcal{R})$ be a range space of* VC *dimension ξ, let X be a finite subset of $\widehat{\mathsf{X}}$, and suppose that $0 < \varepsilon \leq 1$ and $\varphi < 1$. Let N be a set obtained by m random independent draws from X, where $m \geq \max\left(\frac{4}{\varepsilon}\lg\frac{4}{\varphi}, \frac{8\xi}{\varepsilon}\lg\frac{16}{\varepsilon}\right)$. Then N is an ε-net for X with probability at least $1-\varphi$.*

The following is a slight strengthening of the result of Vapnik and Chervonenkis [151]—see [78, Theorem 7.13].

Theorem 4.2 (ε-sample theorem). *Let $\varphi, \varepsilon > 0$ be parameters and let $\left(\widehat{\mathsf{X}}, \mathcal{R}\right)$ be a range space with* VC *dimension ξ. Let $\mathsf{X} \subset \widehat{\mathsf{X}}$ be a finite subset. A sample of size $O\left(\varepsilon^{-2}\left(\xi + \log\varphi^{-1}\right)\right)$ from X is an ε-sample for $\mathsf{S} = (\mathsf{X}, \mathcal{R})$ with probability $\geq 1-\varphi$.*

Theorem 4.3 ([86, 105]). *A sample S of size $O\left(\hat{\varepsilon}^{-2}p^{-1}\left(\xi\log p^{-1} + \log\varphi^{-1}\right)\right)$ from a range space with* VC *dimension ξ, is a relative $(p, \hat{\varepsilon})$-approximation with probability $\geq 1-\varphi$.*

The following is a standard statement on the VC dimension of a range space formed by mixing several range spaces together (see [78]).

Lemma 4.1. *Let $\mathsf{S}_1 = (\widehat{\mathsf{X}}_1, \mathcal{R}_1), \ldots, \mathsf{S}_k = (\widehat{\mathsf{X}}, \mathcal{R}_k)$ be k range spaces, where all of them have the same* VC *dimension ξ. Consider the new set of ranges $\widehat{\mathcal{R}} = \{r_1 \cap \ldots \cap r_k \mid r_1 \in R_1, \ldots, r_k \in R_k\}$. Then the range space $\widehat{\mathsf{S}} = (\widehat{\mathsf{X}}, \widehat{\mathcal{R}})$ has* VC *dimension $O(\xi k \log k)$.*

4.2.2 Weak ε-nets

A convex body $C \subseteq \mathbb{R}^d$ is ***ε-heavy*** (or just *heavy*) if $\overline{m}(C) \geq \varepsilon$ (i.e., $|C \cap P| \geq \varepsilon |P|$). Otherwise, C is ***ε-light***.

Definition 4.5 (Weak ε-net). Let P be a set of n points in $\mathbb{R}^d$. A finite set $S \subset \mathbb{R}^d$ is a ***weak ε-net*** for P if for any convex set C with $\overline{m}(C) \geq \varepsilon$, we have $S \cap C \neq \varnothing$.

Note, that like (regular) ε-nets, weak ε-nets have one-sided error—if C is heavy then the net must stab it, but if C is light then the net may or may not stab it.

4.3 FUNCTIONAL NETS: A WEAK NET IN THE ORACLE MODEL

4.3.1 The model, construction, and query process

Model Given a convex body $C \subseteq \mathbb{R}^d$, we assume oracle access to it. This is a standard model in optimization. Specifically, given a query point z, the oracle either returns that $z \in C$, or alternatively it returns a (separating) hyperplane h, such that C lies completely on one side of h, and z lies on the other side.

Our purpose here is to precompute a small subset $S \subseteq P$, such that given any convex body C (with oracle access to it), one can decide if C is ε-light. Specifically, the query algorithm (using only S, and not the whole point set P) generates an (adaptive) sequence of query points $z_1, z_2, \ldots$, such that if any of these query points

are in C, then the algorithm considers C to be heavy. Otherwise, if all the query points miss C, then the algorithm outputs (correctly) that C is light (i.e., $\overline{m}(C) < \varepsilon$).

Construction Given P, the set S is a random sample from P of size

$$\mu = O\Big(\varepsilon^{-1} d^3 \log d \log^3 \varepsilon^{-1} + \varepsilon^{-1} \log \varphi^{-1}\Big) = \widetilde{O}\Big(d^3/\varepsilon\Big), \tag{4.1}$$

where $\varphi > 0$ is a prespecified parameter.

Query process Given a convex body C (with oracle access to it), the algorithm starts with $S_0 = S$. In the ith iteration, the algorithm computes a $\Omega(1/d^2)$-centerpoint z_i of S_i using the algorithm of Theorem 3.1, with failure probability at most $1/4$. If the oracle returns that $z_i \in C$, then the algorithm returns z_i as a proof of why C is considered to be heavy. Otherwise, the oracle returns a separating hyperplane h_i, such that the open halfspace h_i^- contains z_i. Let $S_i' = S_{i-1} \setminus h_i^-$. If $|S_i'| \leq (1-\gamma)\,|S_{i-1}|$, where $\gamma = 1/16d^2$ then we set $S_i = S_i'$ (such an iteration is called ***successful***). Otherwise, we set $S_i = S_{i-1}$. The algorithm stops when $|S_i| \leq \varepsilon\,|S|\,/8$.

4.3.2 Correctness

Let I be the set of indices of all the successful iterations, and consider the convex set $C_I = \cap_{i \in I} h_i^+$. The set C_I is an outer approximation to C. In particular, for an index j, let $C_j = \cap_{i \in I, i \leq j} h_i^+$ be this outer approximation in the end of the jth iteration. We have that $S_j = S \cap C_j$.

Lemma 4.2. *There are at most $\tau = O(d^2 \log \varepsilon^{-1})$ successful iterations. For any j, the convex polyhedron C_j is defined by the intersection of at most τ closed halfspaces.*

Proof: We start with $\mu = |S_0|$ points in S_0. Every successful iteration reduces the number of points in the net S_{j-1} by a factor of γ. Furthermore, the algorithm stops as soon as $|S_j| \leq \varepsilon\,|S_0|\,/8$. This implies there are at most τ iterations, for the minimal τ such that $(1-\gamma)^\tau \leq \varepsilon/8$, where $\gamma = 1/(16d^2)$. That is $\tau = O(d^2 \log \varepsilon^{-1})$.

The second claim is immediate—every successful iteration adds one halfspace to the intersection that forms C_I. QED.

Let $\mathcal{H}^\tau$ be the set of all of convex polyhedra in $\mathbb{R}^d$ that are formed by the intersection of τ closed halfspaces.

Remark 4.1. The VC dimension of $(\mathbb{R}^d, \mathcal{H}^\tau)$ is

$$D = O(d\tau \log \tau) = O\Big(d\Big(d^2 \log \varepsilon^{-1}\Big) \log\Big(d^2 \log \varepsilon^{-1}\Big)\Big) = O(d^3 (\log d) \log^2 \varepsilon^{-1}).$$

This follows readily, as the VC dimension of the range space of points in $\mathbb{R}^d$ and halfspaces is $d+1$, and by the bound of Lemma 4.1 for the intersection of τ such ranges.

Lemma 4.3. *The set S is a relative $(\varepsilon/8, 1/4)$-approximation for $(P, \mathcal{H}^\tau)$, with probability $1-\varphi$.*

Proof: Using Theorem 4.3 with $p = \varepsilon/8$, $\hat{\varepsilon} = 1/4$, and $\xi = D$, implies that a random sample of P of size

$$\begin{aligned} O\Big(\hat{\varepsilon}^{-2} p^{-1} (\xi \log p^{-1} + \log \varphi^{-1})\Big) &= O\Big(\varepsilon^{-1} (D \log \varepsilon^{-1} + \log \varphi^{-1})\Big) \\ &= O\big(\varepsilon^{-1} (d^3 \log d \log^3 \varepsilon^{-1} + \log \varphi^{-1})\big) \end{aligned}$$

is the desired relative $(p, \hat{\varepsilon})$-approximation with probability $\geq 1-\varphi$. And this is indeed the size of S, see Eq. (4.1). QED.

Lemma 4.4. *Given a convex query body C equipped with a separation oracle, the expected number of oracle queries performed by the algorithm is $O(d^2 \log \varepsilon^{-1})$ with expected running time $O\Big(d^9 \varepsilon^{-1} \log^{O(1)}(d\varepsilon^{-1}\varphi^{-1})\Big)$.*

Proof: If the computed point is in the ith iteration is indeed a centerpoint of S_{i-1}, then the algorithm would either stop in this iteration, or the iteration would be successful. Since the probability of the computed point to be the desired centerpoint is at least $\geq 3/4$, it follows that the algorithm makes (in expectation) $\tau/(3/4)$ iterations till success. The ith iteration requires $O(|S_i| + d^7 \log^3 d)$ time, since we use the algorithm of Theorem 3.1 to compute the approximate centerpoint. Summing this over all the iterations τ (bounded in Lemma 4.2), we get expected running time

$$\begin{aligned} O(d^2\mu + \tau d^7 \log^3 d) &= O\Big(d^2 \Big(\varepsilon^{-1} d^3 \log d \log^3 \varepsilon^{-1} + \varepsilon^{-1} \log \varphi^{-1}\Big) + d^9 \log^3 d \log \varepsilon^{-1}\Big) \qquad (4.2) \\ &= O\Big(d^2 \varepsilon^{-1} \log \varphi^{-1} + d^5 \varepsilon^{-1} \log d \log^3 \varepsilon^{-1} + d^9 \log^3 d \log \varepsilon^{-1}\Big). \end{aligned}$$

QED.

Lemma 4.5. *Assuming that S is the desired relative approximation, then for any query body C, if the algorithm declares that it is ε-light, then $|C \cap P| < \varepsilon n$.*

Proof: Let I be the set of successful iterations by the algorithm, and recall that C_I is the outer approximation of C and is the intersection of at most τ halfspaces. Now the algorithm stops in an iteration i when $|S_i| \leq (\varepsilon/8)\,|S|$. Consequently, $\overline{s}(C) \leq \overline{s}(C_I) = |S_i| \,/\, |S| \leq \varepsilon/8$. There are two cases:

(i) If $\overline{m}(C_I) < \varepsilon/8$, then $\overline{m}(C) \leq \overline{m}(C_I) < \varepsilon$ as claimed.

(ii) Otherwise $\overline{m}(C_I) \geq \varepsilon/8$. But then, since S is a relative $(\varepsilon/8, 1/4)$-approximation for the points P and ranges $\mathcal{H}^\tau$ and $C_I \in \mathcal{H}^\tau$, we have that $\overline{m}(C) \leq \overline{m}(C_I) \leq \frac{1}{1-1/4}\overline{s}(C_I) \leq (4/3)(\varepsilon/8) < \varepsilon$.

In either case, the algorithm is correct. QED.

The above implies the following.

Theorem 4.4. *Let P be a set of points in $\mathbb{R}^d$, and let $\varepsilon, \varphi > 0$ be parameters. Let S be a random sample of P of size*

$$\mu = O\Big(\varepsilon^{-1} d^3 \log d \log^3 \varepsilon^{-1} + \varepsilon^{-1} \log \varphi^{-1}\Big) = \widetilde{O}(\varepsilon^{-1} d^3).$$

Then, for a given query convex body C equipped with a separation access, the algorithm described above, which uses only S, computes a sequence of query points $q_1, \ldots, q_m$, such that either:

(i) one of the points $q_i \in C$, and the algorithm outputs q_i as a "proof" that C is ε-heavy, or

(ii) the algorithm outputs that $|C \cap P| < \varepsilon n$.

The query algorithm has the following performance guarantees:

(A) The expected number of oracle queries is $\mathbb{E}[m] = O(d^2 \log \varepsilon^{-1})$.

(B) The algorithm itself (ignoring the oracle queries) runs in $\widetilde{O}\big(d^9 \varepsilon^{-1}\big)$ time (see Eq. (4.2) for exact bound).

The output of the algorithm is correct, for all convex bodies, with probability $\geq 1 - \varphi$.

Remark 4.2. One may hope to bound the probability of the algorithm reporting a false positive. However this is inherently not possible for any weak ε-net construction. Indeed, the algorithm can fail to distinguish between a polygon that contains at least εn of the points of P and a polygon that contains *none* of the points of P. Consider n points P lying on a circle in $\mathbb{R}^2$. Choose εn of these points on the circle, and let C be the convex hull of these points. Clearly C contains at least εn points of P. Now, take each vertex in C and "slice" it off, forming a new polygon C' that contains no points from P. However, C' is still a large polygon and as such may contain a centerpoint during the execution of the above algorithm. Therefore our algorithm may report that C' contains a large fraction of the points, even though C' is contains no points of P, and so it fails to distinguish between C and C'.

Remark 4.3. Clarkson et al. [49] provide also a randomized algorithm that finds a $\big(\frac{1}{d+1} - \gamma\big)$-centerpoint with probability $1 - \delta$ in time $O\big(\big[d\gamma^{-2} \log(d\gamma^{-1})\big]^{d+O(1)} \log \delta^{-1}\big)$. We could use this algorithm instead of Theorem 3.1 in the query process. Since we are computing a better quality centerpoint, the number of iterations τ and sample size μ would be smaller by a factor of d. Specifically, $\tau = O(d \log \varepsilon^{-1})$ and from Lemma 4.1, the VC dimension of the range space $\mathsf{S} = (P, \mathcal{H}^\tau)$ becomes $D = O(d^2 \log d \log^2 \varepsilon^{-1})$. Following the proof of Lemma 4.3, we can construct a sample S which is relative $(\varepsilon/8, 1/4)$-approximation for S with probability $1 - \varphi$ of size

$$\mu = O\Big(\varepsilon^{-1}(D \log \varepsilon^{-1} + \log \varphi^{-1})\Big) = O\Big(\varepsilon^{-1}(d^2 \log d \log^3 \varepsilon^{-1} + \log \varphi^{-1})\Big). \tag{4.3}$$

4.4 CONSTRUCTING CENTER NETS

We next introduce a strengthening of the concept of a weak ε-net. Namely, we require that there is a point p in the net which stabs an ε-heavy convex body C, and that p is also a good centerpoint for $C \cap P$.

Definition 4.6. For a set P of n points in $\mathbb{R}^d$, and parameters $\varepsilon, \alpha \in (0,1)$, a subset $\mathcal{W} \subseteq \mathbb{R}^d$ is an (ε, α)-***center net*** if for any convex shape C, such that $|P \cap C| \geq \varepsilon n$, we have that there is an α-centerpoint of $P \cap C$ in $\mathcal{W}$.

In this section we prove existence of an (ε, α)-center net $\mathcal{W}$ of size roughly $O_d(\varepsilon^{-d^2})$, where

$$\alpha = \frac{\mathsf{c}_1}{(d+1)\log \varepsilon^{-1}},$$

and $\mathsf{c}_1 \in (0,1)$ is some fixed constant to be specified shortly. Note that the quality of the centerpoint is worse by a factor of $\log \varepsilon^{-1}$ than the best one can hope for.

4.4.1 The construction

The construction of the center net will be based on an algorithm for constructing a weak ε-net for P. In particular, the construction algorithm will use the following two results.

Lemma 4.6 ([114]). *Given a set P of n points in $\mathbb{R}^d$, one can compute a set Q of $O\left(n^{d^2}\right)$ points, such that for any subset $P' \subseteq P$, there is a $1/(d+1)$-centerpoint of P' in Q.*

Proof: This is well known, and we include a proof for the sake of completeness.

Let H be the set of all hyperplanes which pass through d points of P. The original proof of the centerpoint theorem implies that a vertex of the arrangement $\mathcal{A}(H)$ is a $1/(d+1)$-centerpoint of P. Let $\mathsf{V}(P)$ denote the set of vertices of $\mathcal{A}(H)$. Observe that $\mathsf{V}(P') \subseteq \mathsf{V}(P)$, for all $P' \subseteq P$, thus implying that $\mathsf{V}(P)$ contains all desired centerpoints. As for the size bound, observe that

$$\alpha = |H| \leq \binom{n}{d} \leq \left(\frac{ne}{d}\right)^d$$

and

$$|\mathsf{V}(P)| \leq \binom{\alpha}{d} \leq \left(\frac{e\left(\frac{ne}{d}\right)^d}{d}\right)^d = n^{d^2}\left(\frac{e}{d}\right)^{d^2+d} = O\left(n^{d^2}\right). \qquad \text{QED.}$$

Lemma 4.7. *Let P be a set of n points in $\mathbb{R}^d$. Let S be a random sample from P of size $\mu = \widetilde{O}(\varepsilon^{-1}d^2)$, see Eq. (4.3) for the exact bound. Then, one can compute a set of points $\mathcal{W}$ from S, of size*

$$O(\mu^{d^2}) = O\left(\left(\varepsilon^{-1}(d^2 \log d \log^3 \varepsilon^{-1} + \log \varphi^{-1})\right)^{d^2}\right)$$

which is a weak ε-net for P with probability $\geq 1 - \varphi$.

Proof: Imagine running the algorithm of Theorem 4.4 with the better quality centerpoint algorithm, as sketched in Remark 4.3. This requires computing a sample S of size as specified in Eq. (4.3). Let Q be the

universal set of centerpoints, as computed by Lemma 4.6, for the set S. We claim that Q is a weak ε-net. Indeed, assuming the sample S is good, in the sense that the algorithm of Theorem 4.4 works (which happens with probability $\geq 1 - \varphi$), then running this algorithm on any convex query body C (using S), generates a sequence of points, such that one of them stabs C, if C is ε-heavy. However, the stabbing points computed by the algorithm of Theorem 4.4 are centerpoints of some subset of S.

It follows that all the stabbing points that might be computed by the algorithm of Theorem 4.4, over all possible ε-heavy query bodies C are contained in the set Q. Consequently, if C is ε-heavy, then C must contain one of the points of Q. QED.

Remark 4.4. A similar construction of a weak ε-net, to the one in Lemma 4.7, from a small sample was described by Mustafa and Ray [125]. Their sample has exponential dependency on the dimension, so the resulting weak ε-net has somewhat worse dependency on the dimension than our construction. In any case, these constructions are inferior as far as the dependency on ε, compared to the work of Matoušek and Wagner [114] and Rubin [137, 138].

The idea will be to repeat the construction of the net of Lemma 4.7, with somewhat worse constants. Specifically, take a sample S of size $\mu = \widetilde{O}(\varepsilon^{-1} d^2)$ from P, see Eq. (4.3) for the exact bound. Next, we construct the set $\mathcal{W}$ for S, using the result of Lemma 4.6. Return $\mathcal{W}$ as the desired (ε, α)-center net.

4.4.2 Correctness

The proof is algorithmic. Fix any convex ε-heavy body C, and let $S_1 = S$ be the ***active set*** and let $P_1 = C \cap P$ be the ***residual set*** in the beginning of the first iteration.

We now continue in a similar fashion to the algorithm of Theorem 4.4. In the ith iteration, the algorithm computes the $1/(d+1)$-centerpoint z_i of S_i (running times do not matter here, so one can afford computing the best possible centerpoint). If z_i is a 2α-centerpoint for P_i, then z_i is intuitively a good centerpoint for P, and the algorithm returns z_i as the desired center point. Observe that by construction, $z_i \in \mathcal{W}$ as desired.

If not, then there exists a closed halfspace h_i^+ containing z_i and at most $2\alpha \, |P_i|$ points of P_i. Let

$$P_{i+1} = P_i \setminus h_i^+ \qquad \text{and} \qquad S_{i+1} = S_i \setminus h_i^+.$$

The algorithm now continues to the next iteration.

Analysis The key insight is that the active set S_i shrinks much faster than the residual set P_i. However, by construction, S_i provides a good upper bound to the size of P_i. Now once the upper bound provided by S_i on the size of P_i is too small, this would imply that the algorithm must have stopped earlier, and found a good centerpoint.

Lemma 4.8. *Let $\tau = \lceil 1 + 3(d+1) + (d+1)\log \varepsilon^{-1} \rceil$, and $\alpha = 1/(4\tau)$. Assuming that S is a relative $(\varepsilon/8, 1/4)$-approximation for the range space $\mathsf{S} = (P, \mathcal{H}^\tau)$, the above algorithm stops after at most τ iterations.*

Proof: As before, we can interpret the algorithm as constructing a convex polyhedra. Indeed, let $D_{i+1} = \bigcup_{j=1}^{i} h_j^-$, and observe that $P_{i+1} \subseteq P \cap D_{i+1}$, and $S_{i+1} = S \cap D_{i+1}$.

For an iteration $i < \tau$, we have

$$n_{i+1} = |P_{i+1}| \geq (1-2\alpha)\,|P_i| \geq (1-2\alpha)^i\,|P_1| \geq (1-2\alpha i)n_1 \geq (1-2\alpha\tau)n_1 \geq (\varepsilon/2)n,$$

using $(1-x)^i \geq 1 - ix$, which holds for any positive $x \in [0,1]$.

On the other hand, the active set shrinks faster in each such iteration, since z_i is a $1/(d+1)$-centerpoint of S_i. Setting $s_i = |S_i|$, we have that

$$s_{i+1} \leq \left(1 - \frac{1}{d+1}\right) s_i \leq \left(1 - \frac{1}{d+1}\right)^i s_1 \leq \exp\left(-\frac{i}{d+1}\right) s_1.$$

We have that $s_\tau \leq \varepsilon s_1/e^3 \leq (\varepsilon/20)s_1$. It follows that,

$$\frac{\varepsilon}{2} \leq \frac{|P_\tau|}{n} \leq \frac{|P \cap D_\tau|}{n} = \overline{m}(D_\tau) \leq \max\left(\frac{1}{1-1/4}\bar{s}(D_\tau), \frac{\varepsilon}{8}\cdot\frac{1}{4} + \bar{s}(D_\tau)\right) \leq \frac{7}{3}\bar{s}(D_\tau) + \frac{\varepsilon}{32}.$$

The penultimate inequality follows since S is a relative $(\varepsilon/8, 1/4)$-approximation to P for ranges like D_τ. However we do not know which case applies (i.e., depending on whether or not $\overline{m}(D_\tau) \geq \varepsilon/8$) and therefore need to take the maximum over both cases. Finally,

$$\frac{\varepsilon}{2} \leq \frac{7}{3}\bar{s}(D_\tau) + \frac{\varepsilon}{32} = \frac{7}{3}\frac{|S_\tau|}{|S|} + \frac{\varepsilon}{32} \leq \frac{7}{3}\frac{(\varepsilon/20)s_1}{s_1} + \frac{\varepsilon}{32} < \frac{\varepsilon}{5},$$

which is impossible. We conclude the algorithm must have stopped at an earlier iteration. QED.

Lemma 4.9. *The above algorithm outputs a α-centerpoint of $P \cap C$.*

Proof: Assume the algorithm stopped in the ith iteration. But then z_i is a 2α-centerpoint of P_i. Since $n_i \geq n_\tau \geq n_1/2$, it follows that any closed halfspace that contains z_i, contains at least $2\alpha n_i \geq \alpha n_1$ points of P_i, and thus of P_1. We conclude that z_i is a α-centerpoint of P as desired. QED.

Arguing as in Remark 4.3 implies the following.

Corollary 4.1. *For the above algorithm to succeed with probability $\geq 1 - \varphi$, the sample S needs to be a sample of the size specified by Eq. (4.3).*

Theorem 4.5. *Let P be a set of n points in $\mathbb{R}^d$, and $\varepsilon > 0$ be a parameter. For $\gamma = \log(1/\varepsilon)$, there exists a $(\varepsilon, \Omega(1/(d\gamma)))$-center net $\mathcal{W}$ (which is also a weak ε-net) of P (see Definition 4.6). The size of the net $\mathcal{W}$ is $O(\mu^{d^2}) \approx O_d(\varepsilon^{-d^2})$, where $\mu = \widetilde{O}(\varepsilon^{-1}d^2)$, see Eq. (4.3) for the exact bound.*

Proof: The theorem follows readily from the above, by setting $\varphi = 1/2$. QED.

4.5 CONSTRUCTING NETS FROM LINES AND FLATS

Definition 4.7. The affine hull of a point set $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^d$ is the set

$$\left\{ \sum_i \alpha_i p_i \mid \forall i \, \alpha_i \in \mathbb{R} \text{ and } \sum_i \alpha_i = 1 \right\}.$$

For $0 \leq k < d$, a ***k-flat*** is the affine hull of a set of $k+1$ (affinely independent) points.

Definition 4.8. For parameters $\varepsilon \in (0,1)$ and $k \in \{0, 1, \ldots, d-1\}$, a set K of k-flats is a ***(k,ε)-net for volume measure*** if for any convex body $C \subseteq \mathbb{R}^d$ with $\mathrm{vol}\left(C \cap [0,1]^d\right) \geq \varepsilon$, there exists a flat $\varphi \in K$ such that $\varphi \cap C \neq \emptyset$.

4.5.1 Lower bound

Lemma 4.10. *For a parameter $\varepsilon \in (0,1)$, any (k,ε)-net for volume measure must have size $\Omega_d(1/\varepsilon^{1-k/d})$.*

Proof: Let K be a (k,ε)-net for volume measure. For each k-flat $\varphi \in K$, let $H(\varphi, r)$ be the locus of points in $[0,1]^d$ within distance at most r from φ (for $k = 1$ in three dimensions, this is the intersection of $[0,1]^d$ and the cylinder with radius r centered at the line φ). Note that a ball b with center c and radius r intersects a k-flat φ if and only if $c \in H(\varphi, r)$.

Fix $r = (\varepsilon/\mu)^{1/d}$, where μ is a constant to be determined shortly. We claim that by choosing μ appropriately, if K is a (k,ε)-net for volume measure, then the collection of objects $\{H(\varphi, r) \mid \varphi \in K\}$ covers $[0,1]^d$. Indeed, suppose not. Then there exists a point $p \in [0,1]^d$ not covered by any of the objects $H(\varphi, r)$. This implies that a ball b centered at p with radius r does not intersect any k-flat of K, and its volume is $c_d r^d = c_d \varepsilon/\mu$, where c_d is a constant that depends on d. Choose $\mu = c_d$ so that b has volume at least ε, but does intersect any k-flat of K. A contradiction to the required net property.

Hence, by the choice of r, any (k,ε)-net for volume measure must satisfy the condition in which $\{H(\varphi, r) \mid \varphi \in K\}$ covers $[0,1]^d$. For any k-flat $\varphi \in K$, we have $\beta = \mathrm{vol}(H(\varphi, r)) = O_d(r^{d-k}) = O_d(\varepsilon^{1-k/d})$. Thus, to cover $[0,1]^d$, we have that $|K| \geq 1/\beta = \Omega_d(1/\varepsilon^{1-k/d})$. QED.

4.5.2 Constructing (k,ε)-nets for volume measure for $k > 0$

In this section we give a self-contained proof of a deterministic, explicit construction of (k,ε)-nets for volume measure of size $O_d(1/\varepsilon^{1-k/d})$ for $k \geq 1$ which matches the lower bound of Lemma 4.10 up to constant factors. The construction will be done recursively on the dimension d.

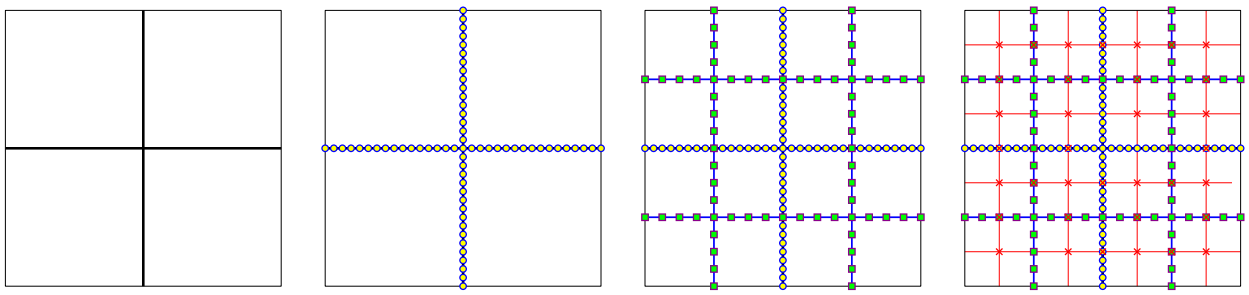


Figure 4.1: The multi-level grid, and its associated lines.

Base case: $k = d - 1$ Here a $(d-1, \varepsilon)$-net for volume measure of size $d/\varepsilon^{1/d} = O_d(1/\varepsilon^{1-k/d})$ follows readily by overlaying a d-dimensional grid of size length $\varepsilon^{1/d}$ and letting the net consist of the hyperplanes forming the grid. As such, we assume $k < d - 1$.

Construction The construction is based on quadtrees. Starting with the entire cube $[0,1]^d$, we construct d orthogonal hyperplanes which *split* the cube into 2^d cubes of side length $1/2$. We refer to such hyperplanes as ***splitting hyperplanes***. This splitting process is continued recursively inside each cell, for $i = 0, \ldots, \tau$, where

$$\tau = \left\lceil \frac{1}{d} \lg \frac{1}{\varepsilon} \right\rceil + 3 \lceil \lg(3d) \rceil + 1 \tag{4.4}$$

(and $\lg = \log_2$), so that cubes at the ith level of the construction has side length $1/2^i$. The number of such cubes at the ith level is 2^{di}. Naturally, these cubes together form a grid with side length $1/2^i$. See Figure 4.1 for an illustration of the construction in two dimensions.

For each splitting hyperplane h at level $i \geq 1$, which splits cells of side length $1/2^{i-1}$ into cells of side length $1/2^i$, we recursively construct a (k, ε_i)-net for volume measure on h (which lies in $d-1$ dimensions), where

$$\varepsilon_i = \frac{2^i \varepsilon}{4d}. \tag{4.5}$$

We collect all k-flats on all splitting hyperplanes at all levels into our (k, ε)-net for volume measure K.

Analysis

Lemma 4.11. *The constructed (k, ε)-net for volume measure has size $O_d(1/\varepsilon^{1-k/d})$.*

Proof: Let $T(\varepsilon, d)$ denote the minimum size of a (k, ε)-net for volume measure over $[0,1]^d$. The proof is by induction on d. When $d = k + 1$, we have $T(\varepsilon, k+1) \leq (k+1)/\varepsilon^{1/(k+1)}$, by the base case described above. So assume $d \geq k + 2$ and $T(\delta, d') \leq \beta(d')/\delta^{1-k/d'}$ for all $d' < d$, where $\beta(d')$ is a constant to be determined. By the inductive hypothesis, the above construction produces a (k, ε)-net for volume measure of size

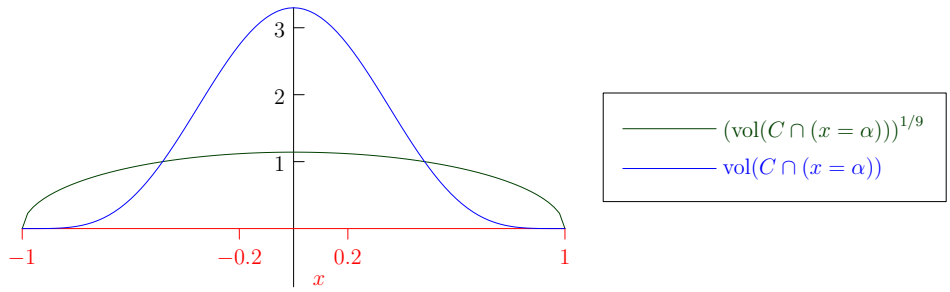


Figure 4.2: The slice volume, and its 1/9th power, for the unit radius ball C in 10 dimensions. This is an example of the concavity implied by the Brunn-Minkowski inequality, which in turn implies that the slice function is unimodal.

$$|K| \leq d \sum_{i=1}^{\tau} 2^{i-1} T(\varepsilon_i, d-1) \leq d \sum_{i=1}^{\tau} \frac{2^{i-1}\beta(d-1)}{\varepsilon_i^{1-k/(d-1)}} \leq \frac{4d^2\beta(d-1)}{\varepsilon^{1-k/d}} \sum_{i=1}^{\tau} \frac{2^{i-1}}{2^{i-ik/(d-1)}}$$
$$\leq \frac{2d^2\beta(d-1)}{\varepsilon^{1-k/d}} \sum_{i=1}^{\tau} 2^{ik/(d-1)} \leq \frac{4d^2\beta(d-1)}{\varepsilon^{1-k/(d-1)}} \cdot 2^{\tau k/(d-1)} \leq \frac{16d^2\beta(d-1)}{\varepsilon^{1-k/d}}.$$

The last inequality follows since $\tau \leq \frac{1}{d}\lg\frac{1}{\varepsilon} + 2$. In particular, we obtain the recurrence $\beta(d) = 16d^2\beta(d-1)$, which solves to $\beta(d) = d^{O(d)}$. As such, the size of K is $O_d(1/\varepsilon^{1-k/d})$. QED.

The Brunn-Minkowski inequality and unimodal functions The C be a convex body in $\mathbb{R}^d$. For a parameter $\alpha \in \mathbb{R}$, let $f(\alpha)$ denote the $(d-1)$-dimensional volume of C intersected with the hyperplane $x = \alpha$. The Brunn-Minkowski inequality [78, 112] implies that the function $g(\alpha) = f(\alpha)^{1/(d-1)}$ is concave. In particular, g is ***unimodal***. Namely, there exists a $\beta \in \mathbb{R}$ such that g is non-decreasing on $(-\infty, \beta]$ and non-increasing on $[\beta, \infty)$. As such, the function f itself is unimodal. See Figure 4.2.

Lemma 4.12. *The set K is a (k, ε)-net for volume measure.*

Proof: Let C be a convex body contained in $[0,1]^d$ with volume at least ε. Assume, for the sake of contradiction, that C is not stabbed by any of the k-flats of K.

Let $h(\alpha)$ be the hyperplane orthogonal to the first axis which intersects the first axis at $\alpha \in \mathbb{R}$. Define the function

$$f(\alpha) = \mathsf{vol}\big(C \cap h(\alpha)\big).$$

By the Brunn-Minkowski inequality, the function $g(\alpha) = f(\alpha)^{1/(d-1)}$ is concave and unimodal. Define the point $x^* \in [0,1]$ so that $x^\star = \arg\max_\alpha f(\alpha)$.

Let $V(\Delta) = f(x^\star + \Delta)$, and let $v(\Delta) = (V(\Delta))^{1/(d-1)}$. The function v, being a translation of g, is concave and unimodal. Let $r_i \geq 0$ be the maximum number such that $V(r_i) = \varepsilon_i$, for $i = 1, \ldots, \tau$. Observe that if $r_i \geq 1/2^i$, then there is hyperplane orthogonal to the first axis that has a recursive construction of a net on it, for ε_i. This by induction would imply that the net intersects C. We thus assume from this point on that

$$r_i < \frac{1}{2^i},$$

for all i. Observe that $r_1 \geq r_2 \geq \cdots \geq r_\tau$, as $\varepsilon_1 < \varepsilon_2 < \cdots < \varepsilon_\tau$ (more specifically, $\varepsilon_i = 2\varepsilon_{i-1}$ for all i).

The concavity of $v(\cdot)$, see Figure 4.3, implies that

$$\frac{v(r_{i+2}) - v(r_{i+1})}{r_{i+2} - r_{i+1}} \geq \frac{v(r_{i+1}) - v(r_i)}{r_{i+1} - r_i} \qquad \Longrightarrow \qquad \frac{r_{i+1} - r_i}{r_{i+2} - r_{i+1}} \leq \frac{v(r_{i+1}) - v(r_i)}{v(r_{i+2}) - v(r_{i+1})},$$

as $r_{i+1} - r_i < 0$ and $v(r_{i+2}) - v(r_{i+1}) > 0$. Since $V(r_{i+1}) = \varepsilon_{i+1} = 2\varepsilon_i = 2V(r_i)$, we have that $v(r_{i+1}) =$

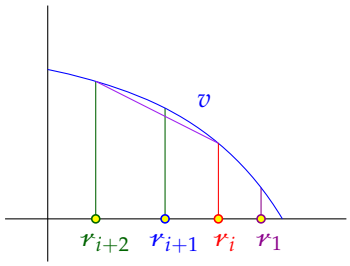


Figure 4.3: By the choice of $r_\tau \leq \ldots \leq r_1$, we have $v(r_\tau) \geq \ldots \geq v(r_1)$.

$2^{1/(d-1)} v(r_i)$. For $i < \tau$, let $\ell_i = r_i - r_{i+1}$. Plugging this into the above, observe

$$\frac{\ell_i}{\ell_{i+1}} = \frac{r_i - r_{i+1}}{r_{i+1} - r_{i+2}} \leq \frac{v(r_{i+1}) - v(r_i)}{v(r_{i+2}) - v(r_{i+1})} = \frac{(2^{1/(d-1)} - 1)v(r_i)}{2^{1/(d-1)}(2^{1/(d-1)} - 1)v(r_i)} = \frac{1}{2^{1/(d-1)}}.$$

Since $\ell_{\tau-1} \leq r_{\tau-1} \leq 1/2^{\tau-1}$, we have

$$\begin{aligned} r_1 &= r_\tau + \sum_{i=1}^{\tau-1} \ell_i \leq r_\tau + \ell_{\tau-1}\left(1 + \frac{1}{2^{1/(d-1)}} + \frac{1}{2^{2/(d-1)}} + \cdots\right) \\ &\leq r_\tau + 2d\ell_{\tau-1} \leq (2d+1) r_{\tau-1} < \frac{2d+1}{2^{\tau-1}} < \frac{\varepsilon^{1/d}}{4d^2}, \end{aligned}$$

by the value of τ, see Eq. (4.4).

Let I_1 be the maximum interval, where the value of $V(x) \geq \varepsilon_1$, for any $x \in I_1$. By the above, we have that if the net does not intersect C, then $\|I_1\| \leq 2r_1 \leq 2\varepsilon^{1/d}/(4d^2)$.

We define $I_2, \ldots, I_d$ in a similar fashion on the other axes, and the same argumentation would imply that $\|I_j\| \leq 2\varepsilon^{1/d}/(4d^2)$, for all j. Furthermore, any plane orthogonal to the axes that avoids the box $B = I_1 \times I_2 \cdots \times I_d$ has an intersection with C of volume at most ε_1. We conclude that the total value of C is at most

$$\mathsf{vol}(C) \leq \mathsf{vol}(B) + \sum_{j=1}^{d} \int_{y \in [0,1] \setminus I_j} \mathsf{vol}\Big(C \cap (x_j = y)\Big)\, \mathrm{d}y \leq \prod_{j=1}^{d} \|I_j\| + d\varepsilon_1 \ll \varepsilon,$$

which is a contradiction to $\mathsf{vol}(C) \geq \varepsilon$. QED.

Theorem 4.6. *Given $\varepsilon \in (0,1)$ and $k \in \{1, \ldots, d-1\}$, the above is a deterministic and explicit construction of a (k, ε)-net for volume measure over $[0,1]^d$ of size $O_d(1/\varepsilon^{1-k/d})$.*

4.5.3 Constructing $(0, \varepsilon)$-nets for volume measure

Ellipsoids are enough We now give constructions for $(0, \varepsilon)$-nets for volume measure. The following result shows that it suffices to build such nets when the convex bodies are restricted to be ellipsoids.

Lemma 4.13. *Suppose there exists a $(0, \varepsilon)$-net for volume measure over $[0,1]^d$ for ellipsoids of size $T(\varepsilon, \tau)$, for $\tau = 1, \ldots, d$. Then one can construct a $(0, \varepsilon)$-net for volume measure over $[0,1]^d$ of size $T(\varepsilon/d^d, d)$.*

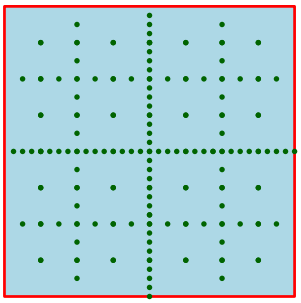


Figure 4.4: The net constructed.

Proof: Consider any convex body C, such that $\mathsf{vol}\Big(C \cap [0,1]^d\Big) \geq \varepsilon$. Let E be the ellipsoid of largest volume contained inside $C \cap [0,1]^d$. By John's ellipsoid theorem, we have that $E \subseteq C \subseteq dE$. In particular,

$$\mathsf{vol}(E) = \mathsf{vol}(dE)/d^d \geq \frac{\mathsf{vol}(C)}{d^d} \geq \frac{\varepsilon}{d^d}.$$

As such, any $(0, \varepsilon/d^d)$-net for volume measure when the convex bodies are restricted to be ellipsoids is a $(0,\varepsilon)$-net for volume measure in the general setting. QED.

Hence, we focus on building $(0,\varepsilon)$-nets for volume measure (equivalently, these are also ε-nets for volume measure) for ellipsoids. Note that it is easy to obtain an ε-net of size $O_d(\varepsilon^{-1}\log\varepsilon^{-1})$ by random sampling [88]. Here, we give a deterministic, explicit construction of such a net.

We first describe the net construction in two dimensions and then extend the proof and ideas to arbitrary dimension.

Let E be an ellipse contained in the unit square $[0,1]^2$ with $\mathsf{area}(E) \geq \varepsilon$. The following construction is inspired by a construction of Pach and Tardos [129].

Construction Let $M = 3 + \lceil \lg \varepsilon^{-1} \rceil$. For $j = 1, \ldots, M-1$, consider the rectangle

$$R_j = [0, 1/2^{M-j}] \times [0, 1/2^j].$$

Consider the natural tiling of $[0,1]^2$ by the rectangle R_i, and let P_i be the set of vertices of the resulting grid G_i in the interior of the unit square. Let $S = \cup_i P_i$. See Figure 4.4.

Correctness We need the following easy observation, whose proof is included for the sake of completeness.

Claim 4.1. *Let c be the center of an ellipse E, and let h be the longest horizontal segment contained in E. The segment h passes through c.*

Proof: By the central symmetry of E, if h does not pass through c, then it has a symmetric reflection h' through c, which is a horizontal segment of the same length. Let ℓ be the horizontal line through c, and

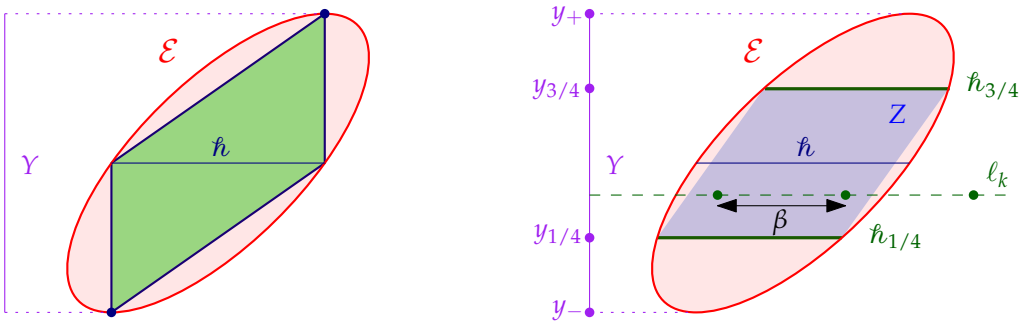


Figure 4.5: The setup for proof of correctness.

observe that $|\ell \cap E| \geq |\hbar|$ by convexity. By the smoothness of E, it follows that $|\ell \cap E| > |\hbar|$, which is a contradiction. QED.

Lemma 4.14. *The set S constructed above is an ε-net for volume measure over $[0,1]^2$ for ellipses. Furthermore, $|S| = O(\varepsilon^{-1} \log \varepsilon^{-1})$.*

Proof: Observe that for any i, we have $\mathsf{area}(R_i) = 2^{-(M-j)-j} = 2^{-M} \geq \varepsilon/8$. As such, $|P_i| = O(1/\varepsilon)$, and $|S| = O(M/\varepsilon) = O(\varepsilon^{-1} \log \varepsilon^{-1})$.

Let $E \subseteq [0,1]^2$ be any ellipse with $\mathsf{area}(E) \geq \varepsilon$. Let Y denote the projection of E onto the y-axis. Observe that $|Y| \geq \varepsilon$. Let $\hbar$ be the longest horizontal segment contained in E (which passes through the center of E by Claim 4.1). The two extreme y-axis points in E, and the segment $\hbar$ forms a quadrilateral in E of area $|\hbar|\,|Y|/2$, see Figure 4.5. Let $Y = [y_-, y_+]$, and for $\alpha \in Y$, let $g(\alpha) = |\{y = \alpha\} \cap E|$. We have that

$$|\hbar|\,|Y|/2 \leq \mathsf{area}(E) = \int_{\alpha=y_-}^{y_+} g(\alpha)\,\mathrm{d}\alpha \leq |\hbar|\,|Y|\,.$$

Since $\mathsf{area}(E) \geq \varepsilon$, we conclude that $|\hbar| \geq \varepsilon/|Y|$.

We set $y_{1/4} = (3/4)y_- + (1/4)y_+$ and $y_{3/4} = (1/4)y_- + (3/4)y_+$. Consider the two horizontal segments $\hbar_{1/4} = \{y = y_{1/4}\} \cap E$ and $\hbar_{3/4} = \{y = y_{3/4}\} \cap E$. These two segments are of the same length and are parallel. Furthermore, $\gamma = |\hbar_{1/4}| = |\hbar_{3/4}| \geq |\hbar|/2$, see Figure 4.5. Consider the parallelogram Z formed by the convex hull of $\hbar_{1/4}$ and $\hbar_{3/4}$. Observe, that for any $\alpha \in [y_{1/4}, y_{3/4}]$, we have that $|\{y = \alpha\} \cap Z| = \gamma$. As such, $\mathsf{area}(Z) = \gamma \cdot |Y|/2 \geq |\hbar|/2 \cdot |Y|/2 \geq \varepsilon/4$. Let k be the minimum integer such that $1/2^{k+1} \leq |Y|/2$. Since $|Y| \geq \varepsilon$, it follows that $k < M - 2$.

This implies that the grid G_{k+1} has a horizontal line ℓ_k that intersects Z. Furthermore, we have

$$|\ell_k \cap E| \geq |\ell_k \cap Z| = \gamma \geq \frac{|\hbar|}{2} \geq \frac{\varepsilon}{2|Y|} \geq \varepsilon 2^k \geq \frac{8 \cdot 2^k}{2^M} = \frac{1}{2^{M-k-3}} > \frac{1}{2^{M-(k+1)}} = \beta,$$

since $M = 3 + \lceil \lg \varepsilon^{-1} \rceil$. Namely, the spacing of the points of G_{k+1} on the line ℓ_k (i.e., β) is shorter than the

interval $\ell_k \cap E$. It follows that a point of $P_{k+1} \subseteq S$ lies in E, and thus establishing the claim. QED.

We now extend the previous construction to higher dimensions. The construction is recursive. Namely, we assume that for all $d' < d$ we can construct an ε-net for volume measure over $[0,1]^{d'}$ for ellipsoids of size $(\beta(d')/\varepsilon)\lg^{d'-1}(1/\varepsilon)$, where $\beta(d')$ is a constant depending on the dimension d' (to be determined shortly). Lemma 4.14 proves the claim when $d = 2$.

Construction Label the d axes $x_1, \ldots, x_d$. Let $\tau = \lceil (1/d)\lg(1/\varepsilon) \rceil$ and define the function $\Delta(i) = 2^i \varepsilon^{1/d}$. We repeat the following construction for each axis x_ℓ, where $\ell = 1, \ldots, d$. For each $i = 0, \ldots, \tau$, let $M_i = \lceil \lg(1/\Delta(i)) \rceil$. For each i, and for each $j = 0, \ldots, M_i$, form $2^j + 1$ evenly spaced hyperplanes which are orthogonal to the axis x_ℓ (thus consecutive hyperplanes are separated by distance 2^{-j}). For each hyperplane h, we recursively construct a $(0, \varepsilon/\Delta(i+2))$-net $P_{\ell,i,j}$ for $[0,1]^{d-1}$ on $h \cap [0,1]^d$. Let $P_\ell = \cup_{i=1}^{\tau} \cup_{j=1}^{M_i} P_{\ell,i,j}$. Finally, we claim the point set $P = \cup_{\ell=1}^{d} P_\ell$ is the desired $(0,\varepsilon)$-net.

Label the d axes $x_1, \ldots, x_d$. Let $\tau = \lceil (1/d)\lg(1/\varepsilon) \rceil$ and define the function $\Delta(i) = 2^i \varepsilon^{1/d}$. We repeat the following construction for each axis x_ℓ, where $\ell = 1, \ldots, d$. For each $i = 0, \ldots, \tau$, let $M_i = \lceil \lg(1/\Delta(i)) \rceil$. For each i, and for each $j = 0, \ldots, M_i$, form $2^j + 1$ evenly spaced hyperplanes which are orthogonal to the axis x_ℓ (thus consecutive hyperplanes are separated by distance 2^{-j}). For each hyperplane h, we recursively construct a $\varepsilon/\Delta(i+2)$-net $P_{\ell,i,j}$ for $[0,1]^{d-1}$ on $h \cap [0,1]^d$. Let $P_\ell = \cup_{i=1}^{\tau} \cup_{j=1}^{M_i} P_{\ell,i,j}$. Finally, we claim the point set $P = \cup_{\ell=1}^{d} P_\ell$ is the desired ε-net for volume measure.

Theorem 4.7. *For $\varepsilon \in (0, 2^{-2d}]$, there exists a ε-net for volume measure over $[0,1]^d$ for ellipsoids, of size*

$$2^{O(d^2)} \varepsilon^{-1} \lg^{d-1} \varepsilon^{-1}.$$

Proof: We first bound the size of the resulting net. Since $\varepsilon \leq 2^{-2d}$, by a direct calculation,

$$\begin{aligned}
|P| &\leq \sum_{\ell=1}^{d} |P_\ell| \leq d \sum_{i=0}^{\tau} \sum_{j=0}^{M_i} (2^j + 1) \cdot \beta(d-1) \cdot \left(\frac{\Delta(i+2)}{\varepsilon} \lg^{d-2}\left(\frac{\Delta(i+2)}{\varepsilon} \right) \right) \\
&\leq \frac{2d \cdot \beta(d-1)}{\varepsilon} \sum_{i=0}^{\tau} 2^{M_i+1} \cdot 2^2 \Delta(i) \lg^{d-2}\left(\frac{\Delta(i+2)}{\varepsilon} \right) \\
&\leq \frac{2^5 d \cdot \beta(d-1)}{\varepsilon} \sum_{i=0}^{\tau} \lg^{d-2}\left(\frac{2^{i+2}}{\varepsilon^{1-1/d}} \right) \\
&\leq \frac{2^5 d \cdot \beta(d-1)}{\varepsilon} \sum_{i=0}^{\tau} \left((i+2) + \lg\left(\frac{1}{\varepsilon^{1-1/d}} \right) \right)^{d-2}.
\end{aligned}$$

Since $i + 2 \leq \tau + 2 \leq \lg(1/\varepsilon)$ for $\varepsilon \leq 2^{-2d}$, we have

$$|P| \leq \frac{2^5 d \cdot \beta(d-1)}{\varepsilon} \left[(\tau+1) \cdot 2^{d-2} \lg^{d-2}\left(\frac{1}{\varepsilon} \right) \right] \leq \frac{2^5 d \cdot \beta(d-1)}{\varepsilon} \left[\frac{4}{d} \lg \frac{1}{\varepsilon} \cdot 2^{d-2} \lg^{d-2} \frac{1}{\varepsilon} \right].$$

As such, $|P| \leq \frac{2^{d+5} \cdot \beta(d-1)}{\varepsilon} \lg^{d-1}\left(\frac{1}{\varepsilon}\right)$. In particular, we obtain the recurrence $\beta(d) = 2^{d+5}\beta(d-1)$, which solves to $\beta(d) = 2^{O(d^2)}$. Hence, $|P| = 2^{O(d^2)}\varepsilon^{-1} \lg^{d-1} \varepsilon^{-1}$.

We now argue correctness. Let E be an ellipsoid of volume at least ε. Let B be the smallest enclosing axis-aligned box for E. Suppose that the longest edge of B is along the ℓth axis. In particular, along this ℓth axis B has side length $s \geq \varepsilon^{1/d}$, for otherwise $\mathsf{vol}(E) \leq \mathsf{vol}(B) \leq s^d < \varepsilon$. We claim that E intersects a point in the set P_ℓ.

Let $L = [\ell_-, \ell_+]$ be the projection of E onto the ℓth axis, with $s = |L|$. For $x \in L$, define $H(x)$ to be the hyperplane orthogonal to the ℓth axis which intersects the ℓth axis at x. Finally, let K be the hyperplane through the center of E which is orthogonal to the ℓth axis and set $F = E \cap K$. We claim that $\mathsf{vol}(F) \geq \varepsilon/s$. To prove the claim, suppose towards contradiction that $\mathsf{vol}(E \cap K) < \varepsilon/s$. Then,

$$\mathsf{vol}(E) = \int_{\ell_-}^{\ell_+} \mathsf{vol}(E \cap H(x))\,\mathrm{d}x < \frac{\varepsilon}{s}\int_{\ell_-}^{\ell_+} 1\,\mathrm{d}x = \frac{\varepsilon}{s}\,|L| = \varepsilon,$$

a contradiction.

Choose an integer $i \geq 0$ such that $s \in [\Delta(i), \Delta(i+1))$. Let $z_{1/4} = (3/4)\ell_- + (1/4)\ell_+$ and $z_{3/4} = (1/4)\ell_- + (3/4)\ell_+$. Observe that for all $x \in [z_{1/4}, z_{3/4}]$, $\mathsf{vol}(E \cap H(x)) \geq \varepsilon/(2s) \geq \varepsilon/\Delta(i+2)$. Next, let j be the minimum integer such that $1/2^{j+1} \leq s/2$. Note that such an integer exists, as we can choose $j = \lceil \lg(1/s) \rceil$. Since $s \geq \Delta(i)$, $j \leq \lceil \lg(1/\Delta(i)) \rceil \leq M_i$. Thus, for our choices of i and j, we have found a hyperplane h which intersects E with $\mathsf{vol}(E \cap h) \geq \varepsilon/\Delta(i+2)$. By our recursive construction, there is a point in the net $P_{\ell,i,j}$ which intersects $E \cap h$ and thus E. QED.

Theorem 4.8. *There is a deterministic, explicit construction of $(0,\varepsilon)$-nets for volume measure over $[0,1]^d$ of size*

$$O_d\left(\frac{1}{\varepsilon}\log^{d-1}\frac{1}{\varepsilon}\right).$$

Proof: Follows by plugging in the bound for Theorem 4.7 into Lemma 4.13. QED.

5 The yolk and related geometric consensuses

> *When you have exhausted all possibilities, remember this: you haven't.*
>
> **— Thomas Edison**

Let P be a set of n points in $\mathbb{R}^d$ in general position. A median hyperplane (roughly) splits the point set P in half. The *yolk* of P is the ball of smallest radius intersecting all median hyperplanes of P. The *egg* of P is the ball of smallest radius intersecting all hyperplanes which contain exactly d points of P.

We present exact algorithms for computing the yolk and the egg of a point set, both running in expected time $O_d(n^{d-1} \log n)$. The running time of the new algorithm is a polynomial time improvement over existing algorithms. We also present algorithms for several related problems, such as computing the Tukey and center balls of a point set, among others.

5.1 BACKGROUND

5.1.1 Voting games and the yolk

Suppose there is a collection of n voters in $\mathbb{R}^d$, where each dimension represents a specific ideology. In a fixed dimension, each voter maintains a value along this continuum representing their stance on a given ideology. One can interpret $\mathbb{R}^d$ as a *policy space*, and each point in $\mathbb{R}^d$ represents a single policy. In the Euclidean spatial model, a voter $p \in \mathbb{R}^d$ always prefers policies which are closer to p under the Euclidean norm. For two policies $x, y \in \mathbb{R}^d$ and a set of voters $P \subset \mathbb{R}^d$, x *beats* y if more voters in P prefer policy x compared to y. A plurality point is a policy which beats all other policies in $\mathbb{R}^d$. For $d = 1$, the plurality point is the median voter (when n is odd) [24]. However for $d > 1$, a plurality point is not always guaranteed to exist [139]. It is known that one can test if a plurality point exists (and if so, compute it) in $O(dn \log n)$ time [21]. Note that the plurality point is a point of Tukey depth $\lceil n/2 \rceil$—in general this is the largest possible Tukey depth any point can have; while the centerpoint (Definition 3.1$_{\text{p29}}$) is a point that guarantees a "respectable" minority of size at least $n/(d+1)$.

Since plurality points may not always exist, one generalization of a plurality point is the yolk [115]. A hyperplane is a ***median hyperplane*** if the number of voters lying in each of the two closed halfspaces is at

least $\lceil n/2 \rceil$. The ***yolk*** is the ball of smallest radius intersecting all such median hyperplanes. Note that when a plurality point exists, the yolk has radius zero (equivalently, all median hyperplanes intersect at a common point).

We also consider the following restricted problem. A hyperplane is ***extremal*** if and only if it passes through d points, under the assumption that the points are in general position. The ***extremal yolk*** is the ball of smallest radius intersecting all extremal median hyperplanes. Importantly, the yolk and the extremal yolk are different problems—the radius of the yolk and extremal yolk can differ [148].

5.1.2 The egg of a point set

A problem related to computing the yolk is the following: For a set of n points P in $\mathbb{R}^d$, compute the smallest radius ball intersecting all extremal hyperplanes of P (i.e., all hyperplanes passing through d points of P). Such a ball is the ***egg*** of P. See Figure 5.1 for an illustration of the yolk and egg of a point set.

5.1.3 Linear programs with many implicit constraints

The problem of computing the egg can be written as a linear program (LP) with $\Theta_d(n^d)$ constraints, defined implicitly by the point set P. One can apply Seidel's algorithm [141] (or any other linear time LP solver in constant dimension) to obtain an $O_d(n^d)$ expected time algorithm for computing the egg (or the yolk, with a bit more work). However, as each d-tuple of points forms a constraint, it is natural to ask if one can obtain a faster algorithm in this setting. Specifically, we are interested in the following problem: Let I be an instance of a d-dimensional LP specified via a set of n entities P, where each k-tuple of P induces a linear constraint in I, for some (constant) integer k. The problem is to efficiently solve I, assuming access to some additional subroutines.

5.1.4 Previous work

The yolk Let P be a set of n points in $\mathbb{R}^d$. Both the yolk and extremal yolk have been studied in the literature. The first polynomial time exact algorithm for computing the yolk in $\mathbb{R}^d$ was by Tovey in $O_d\left(n^{(d+1)^2}\right)$ time—in the plane, the running time can be improved to $O(n^4)$ [150]. Following Tovey, the majority of results have focused on computing the yolk in the plane. In 2018, de Berg et al. [21] gave an $O(n^{4/3}\log^{1+\varepsilon} n)$ time algorithm (for any fixed $\varepsilon > 0$) for computing the yolk. Obtaining a faster exact algorithm remained an open problem. Gudmundsson and Wong [71, 72] presented a $(1+\varepsilon)$-approximation algorithm with $O(n\log^7 n \log^4 \varepsilon^{-1})$ running time. An unpublished result of de Berg et al. [19] achieves a randomized $(1+\varepsilon)$-approximation algorithm for the extremal yolk running in expected time $O(n\varepsilon^{-3}\log^3 n)$.

The egg The egg of a point set in $\mathbb{R}^d$ can be computed by solving a linear program with $\Theta_d(n^d)$ constraints. The egg is a natural extension to computing the yolk, and thus obtaining faster exact algorithms is of interest.

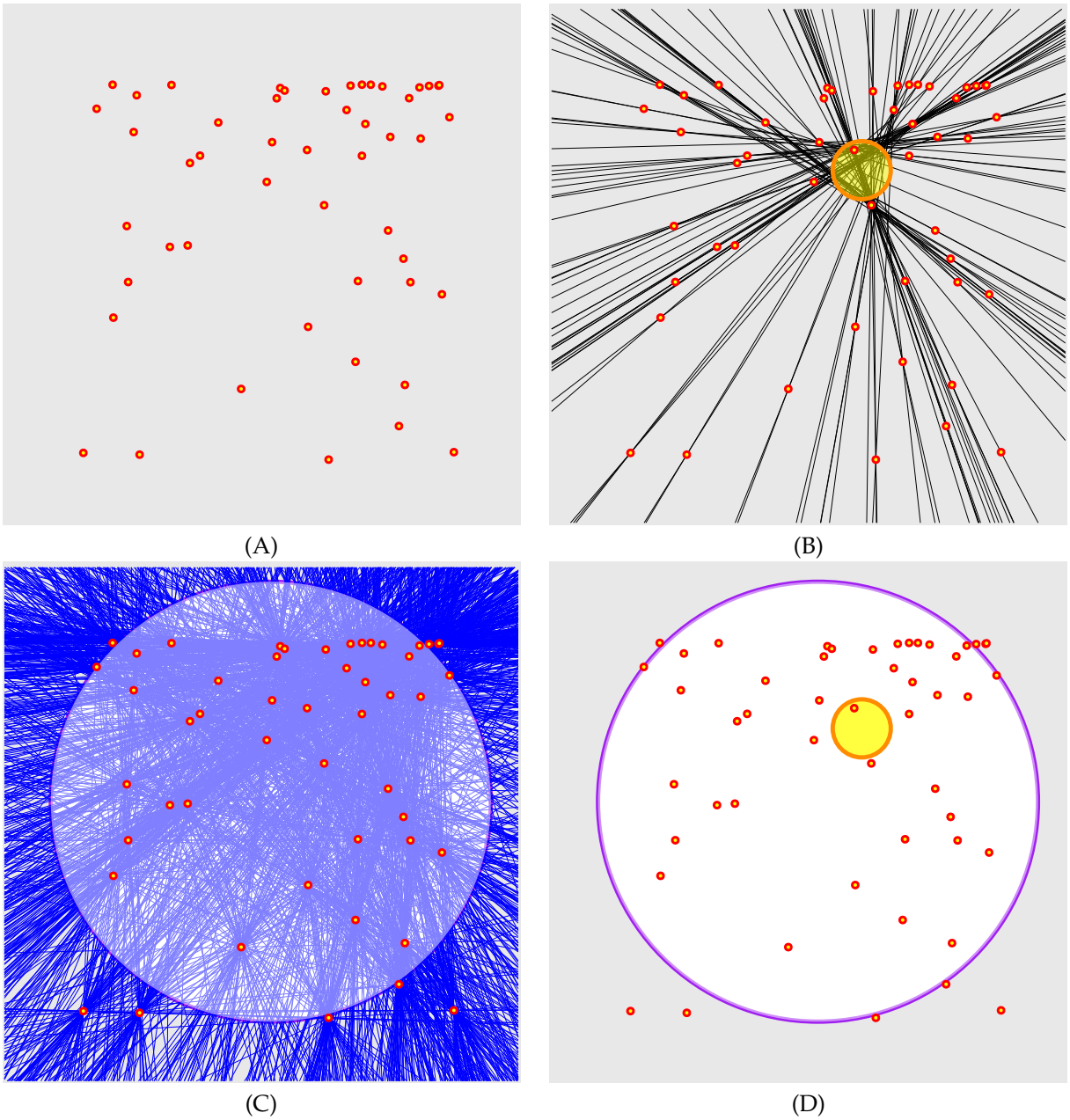


Figure 5.1: (A) Points. (B) Median lines and the extremal yolk. (C) All lines and the egg. (D) Points with the extremal yolk and the egg.

$d = 2$	$(1+\varepsilon)$-apx.	Exact	Our results (Exact)
Extremal yolk	$O(n\varepsilon^{-3}\log^3 n)$ [19]	$O(n^{4/3}\log^{1+\epsilon} n)$ [21]	$O(n\log n)$ Theorem 5.1
Yolk	$O(n\log^7 n\log^4 \varepsilon^{-1})$ [71]	$O(n^{4/3}\log^{1+\epsilon} n)$ Variant of [21]	$O(n\log n)$ Theorem 5.2
$d = 3$			
Yolk	?	$O(n^3)$ Known techniques	$O(n^2)$ Remark 5.1
$d > 3$			
Extremal yolk	?	$O_d(n^d)$ Known techniques	$O_d(n^{d-1}\log n)$ Theorem 5.1
Yolk	?	$O_d(n^d)$ Known techniques	$O_d(n^{d-1}\log n)$ Theorem 5.2

Table 5.1: Some previous work on the yolk and our results. Existing algorithms are deterministic, while the running time of our algorithms holds in expectation.

The authors are not aware of any previous work on this specific problem. Bhattacharya et al. [23] gave an algorithm which computes the smallest radius ball intersecting a set of m hyperplanes in $O(m)$ time, when $d = O(1)$, by formulating the problem as an LP (see also Lemma 5.4). However we emphasize that in our problem the set of hyperplanes are implicitly defined by the point set P, and is of size $\Theta(n^d)$ in $\mathbb{R}^d$.

Implicit LPs In 2004, Chan [33] developed a framework for solving LPs with many implicit constraints (the motivation was to obtain an efficient algorithm for computing the Tukey depth of a point set). Informally, suppose that each input set P of entities maps to a set $\mathcal{H}(P)$ of implicit constraints. For n entities P and a candidate solution, suppose one can decide if the candidate solution violates any constraints of $\mathcal{H}(P)$ in $D(n)$ time. Additionally, assume that from P, one can construct $r = O(1)$ sets $P_1, \ldots, P_r$, each of size at most n/c (for some constant $c > 1$) with $\mathcal{H}(P) = \bigcup_{i=1}^{r} \mathcal{H}(P_i)$. If this partition step can be performed in $D(n)$ time, then both assumptions imply that the resulting LP can be solved in $O(D(n))$ expected time.

5.1.5 Our results

In this chapter we revisit Chan's algorithm for solving LPs with many implicitly defined constraints [33]. The technique leads to efficient algorithms for the following problems. Throughout, let $P \subset \mathbb{R}^d$ be a set of n points in general position:

(A) The yolk (and extremal yolk) of P can be computed *exactly* in $O_d(n^{d-1}\log n)$ expected time. Hence in the plane, the yolk can be computed *exactly* in $O(n\log n)$ expected time. This improves all existing algorithms (both exact and approximate) [19, 21, 71, 72, 150] for computing the yolk in the plane, and our algorithm easily generalizes to higher dimensions. See Table 5.1 for a summary of our results and previous work.

(B) By a straight-forward modification of the above algorithm, see Lemma 5.7, implies that the egg of P can be computed in $O_d(n^{d-1}\log n)$ expected time. The authors are not aware of any previous work on this specific problem.

(C) Let $H_k(P)$ be the collection of all open halfspaces which contain at least $n-k$ points of P. Consider the convex polygon $\mathcal{T}_k = \cap_{h\in H_k(P)} h$. Observe that $\mathcal{T}_0$ is the convex hull of P, with $\mathcal{T}_0 \supseteq \mathcal{T}_1 \supseteq \cdots$. The centerpoint theorem implies that $\mathcal{T}_{n/(d+1)}$ is non-empty (and contains the centerpoint). The Tukey depth of a point q in the minimal k such that $q \in \mathcal{T}_k \setminus \mathcal{T}_{k+1}$.

When $\mathcal{T}_k$ is non-empty, the ***center ball*** of P is the ball of largest radius contained inside $\mathcal{T}_k$. For $\mathcal{T}_k$ empty, we define the ***Tukey ball*** of P as the smallest radius ball intersecting all halfspaces of $H_k(P)$.

In Section 5.5 we show that the Tukey ball and center ball can both be computed in $\widetilde{O}_d\big(k^{d-1}\big[1+(n/k)^{\lfloor d/2\rfloor}\big]\big)$ expected time (see Lemma 5.12 and Lemma 5.14, respectively). In particular when k is a (small) constant, a point of Tukey depth k can be computed in time $\widetilde{O}_d(n^{\lfloor d/2\rfloor})$. This improves Chan's $O_d(n^{d-1}\log n)$ expected time algorithm for deciding if there is a point of Tukey depth at least k [33].

(D) For a set $Q \subseteq \mathbb{R}^d$, let $\mathsf{conv}(Q)$ denote the convex hull of Q. For a given integer k let $\mathcal{C}(P,k) = \big\{\mathsf{conv}(Q) \mid Q \in \binom{P}{k}\big\}$, where $\binom{P}{k}$ is the set of all k-tuples of points of P. We define the ***k-ball*** of P as the smallest radius ball intersecting all convex bodies in $\mathcal{C}(P,k)$.

While one may be tempted to apply the techniques discussed so far for implicit LPs, there is a faster algorithm using $(\leq k)$-sets. When k is constant, in Lemma 5.15 we present an algorithm for computing the k-ball in $O_d(n^{\lfloor d/2\rfloor} + n\log n)$ expected time. As such, the smallest ball intersecting all triangles induced by triples of a set of n points in $\mathbb{R}^3$ can be computed in $O(n\log n)$ expected time.

In Section 5.7, we present another application of Chan's technique for solving implicit LP-type problems.

(E) Given a set L of n lines in the plane, the ***crossing distance*** between two points $p,q \in \mathbb{R}^2$ is the number of lines of L intersecting the segment pq. Given a point $q \in \mathbb{R}^2$ not lying on any lines of L, the disk of smallest radius containing all vertices of $\mathcal{A}(L)$ within crossing distance at most k from q can be computed in $O(n\log n)$ expected time. See Lemma 5.16.

5.2 PRELIMINARIES

Notation. In this chapter, $\widetilde{O}$ hides factors of the form $\log^c n$, where c may depend on the dimension d.

5.2.1 LP-type problems

An LP-type problem, introduced by Sharir and Welzl [145], is a generalization of a linear program. Let $\mathcal{H}$ be a set of constrains and f be an objective function. For any $\mathcal{B} \subseteq \mathcal{H}$, let $f(\mathcal{B})$ denote the value of the optimal

solution for the constraints of $\mathcal{B}$. The goal is to compute $f(\mathcal{H})$. If the problem is infeasible, let $f(\mathcal{H}) = \infty$. Similarly, define $f(\mathcal{H}) = -\infty$ if the problem is unbounded.

Definition 5.1. Let $\mathcal{H}$ be a set of constraints, and let $f : 2^{\mathcal{H}} \to \mathbb{R} \cup \{\infty, -\infty\}$ be an objective function. The tuple $(\mathcal{H}, f)$ forms an LP-*type problem* if the following properties hold:

(A) MONOTONICITY. For any $\mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{H}$, we have $f(\mathcal{B}) \leq f(\mathcal{C})$.

(B) LOCALITY. For any $\mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{H}$ with $f(\mathcal{C}) = f(\mathcal{B}) > -\infty$, and for all $s \in \mathcal{H}$, $f(\mathcal{C}) < f(\mathcal{C}+s) \iff f(\mathcal{B}) < f(\mathcal{B}+s)$, where $\mathcal{B}+s = \mathcal{B} \cup \{s\}$.

A *basis* of $\mathcal{H}$ is an inclusion-wise minimal subset $\mathcal{b} \subseteq \mathcal{H}$ with $f(\mathcal{b}) = f(\mathcal{H})$. The *combinatorial dimension* δ is the maximum size of any feasible basis of any subset of $\mathcal{H}$. Throughout, we consider δ to be constant. For a basis $\mathcal{b} \subseteq \mathcal{H}$, we say that $h \in \mathcal{H}$ *violates* the current solution induced by $\mathcal{b}$ if $f(\mathcal{b}+h) > f(\mathcal{b})$. LP-type problems with n constraints can be solved in randomized time $O(n)$, hiding constants depending (exponentially) on δ [48], where the bound on the running time holds with high probability.

5.2.2 Implicit LPs using Chan's algorithm

Our algorithms will need the following result of Chan [33] on solving LPs with implicitly defined constraints.

Lemma 5.1 ([33]). *Let $(\mathcal{H}, f)$ be an* LP*-type problem of constant combinatorial dimension δ, and let c_δ be a constant that depends only on δ. Let $\psi, c > 1$ be fixed constants, such that $c_\delta \log^\delta \psi < c$. For an input space Π, suppose that there is a function $g : \Pi \to 2^{\mathcal{H}}$ which maps inputs to constraints. Furthermore, assume that for any input $P \in \Pi$ of size n, we have:*

(I) When $n = O(1)$, a basis for $g(P)$ can be computed in constant time.

(II) For a basis $\mathcal{b}$, one can decide if $\mathcal{b}$ satisfies $g(P)$ in $D(n)$ time.

(III) In $D(n)$ time, one can construct sets $P_1, \ldots, P_\psi \in \Pi$, each of size at most n/c, such that $g(P) = \bigcup_{i=1}^{\psi} g(P_i)$.

Then a basis for $g(P)$ can be computed in $O(D(n))$ expected time (hiding constants depending on δ), assuming that $D(n/k) = O(D(n)/k)$, for all positive integers $k \leq n$.

5.2.3 Duality, levels, and zones

The following definitions and facts are well known in computational geometry. See also de Berg et al. [18] for additional context.

Definition 5.2 (Duality). The *dual hyperplane* of a point $p = (p_1, \ldots, p_d) \in \mathbb{R}^d$ is the hyperplane $p^\star$ defined by the equation $x_d = -p_d + \sum_{i=1}^{d-1} x_i p_i$. The *dual point* of a hyperplane h defined by $x_d = a_d + \sum_{i=1}^{d-1} a_i x_i$ is the point $h^\star = (a_1, a_2, \ldots, a_{d-1}, -a_d)$.

Fact 5.1. *Let p be a point and let h be a hyperplane. Then p lies above h if and only if the hyperplane $p^\star$ lies below the point $h^\star$.*

Given a set of objects T (e.g., points in $\mathbb{R}^d$), we let $T^\star = \{x^\star \mid x \in T\}$ denote the dual set of objects.

Definition 5.3 (Levels). For a collection of hyperplanes H in $\mathbb{R}^d$, the ***level*** of a point $p \in \mathbb{R}^d$ is the number of hyperplanes of H lying on or below p. The ***k-level*** of H is the union of points in $\mathbb{R}^d$ which have level equal to k. The ***$(\leq k)$-level*** of H is the union of points in $\mathbb{R}^d$ which have level at most k.

By Fact 5.1, if h is a hyperplane which contains k points of P lying on or above it, then the dual point $h^\star$ is a member of the k-level of $P^\star$.

For a set of hyperplanes H, we let $\mathcal{A}(H)$ denote the arrangement of H and $\mathsf{V}(\mathcal{A}(H))$ denote the vertices of the arrangement of H.

Definition 5.4 (Zone of a surface). For a collection of hyperplanes H in $\mathbb{R}^d$, the complexity of a cell ψ in the arrangement $\mathcal{A}(H)$ is the number of faces (of all dimensions) which are contained in the closure of ψ. For a $(d-1)$-dimensional surface γ, the ***zone*** $\mathcal{Z}(\gamma, H)$ of γ is the subset of cells of $\mathcal{A}(H)$ which intersect γ. The ***complexity of a zone*** is the sum of the complexities of the cells in $\mathcal{Z}(\gamma, H)$.

The complexity of a zone of a hyperplane is known to be $\Theta(n^{d-1})$ [57]; for general algebraic surfaces it is larger by a logarithmic factor. Furthermore, the cells in the zone of a surface can be computed efficiently using lazy randomized incremental construction [20].

Lemma 5.2 ([12, 20]). *Let H be a set of n hyperplanes in $\mathbb{R}^d$ and let γ be a $(d-1)$-dimensional algebraic surface of degree δ. The complexity of the zone $\mathcal{Z}(\gamma, H)$ is $O_{d,\delta}(n^{d-1}\log n)$ (the hidden constants depend on d and δ). The collection of cells in $\mathcal{Z}(\gamma, H)$ can be computed in $O_{d,\delta}(n^{d-1}\log n)$ expected time.*

5.3 COMPUTING THE EXTREMAL YOLK

Definition 5.5. Let $P \subset \mathbb{R}^d$ be a set of n points in general position. A median hyperplane is a hyperplane such that each of its two closed halfspaces contain at least $\lceil n/2 \rceil$ points of P. A hyperplane is extremal if it passes through d points of P. The ***extremal yolk*** is the ball of smallest radius interesting all extremal median hyperplanes of P.

We give an $O_d(n^{d-1}\log n)$ expected time exact algorithm computing the extremal yolk. To do so, we focus on the more general problem.

Problem 5.1. Let $E_k(P)$ be the collection of extremal hyperplanes which contain exactly k points of P on or above it. Here, k is not necessarily constant. The goal is to compute the smallest radius ball intersecting all hyperplanes of $E_k(P)$.

We observe that computing the extremal yolk can be reduced to the above problem.

Lemma 5.3. *The problem of computing the extremal yolk can be reduced to Problem 5.1.*

Proof: Suppose that n is even, and define the set $S_{\text{even}} = \{n/2, n/2+1, \ldots, n/2+d\}$. A case analysis shows that any extremal median hyperplane h must have exactly m points of P above or on h, where $m \in S_{\text{even}}$. Thus, computing the extremal yolk reduces to computing smallest radius ball intersecting all hyperplanes in the set $\bigcup_{m \in S_{\text{even}}} E_m(P)$.

When n is odd, a similar case analysis shows that any extremal median hyperplane must have exactly m points above or on it, where $m \in S_{\text{odd}} = \{\lceil n/2 \rceil, \lceil n/2 \rceil + 1, \ldots, \lceil n/2 \rceil + d - 1\}$. Analogously, computing the extremal yolk with n odd reduces to computing the smallest radius ball intersecting all hyperplanes in the set $\bigcup_{m \in S_{\text{odd}}} E_m(P)$. QED.

To solve Problem 5.1, we apply Chan's result for solving implicit LP-type problems [33], stated in Lemma 5.1. We first prove that Problem 5.1 is an LP-type problem when the constraints are explicitly given (the following Lemma was also observed by Bhattacharya et al. [23]).

Lemma 5.4. *Problem 5.1 when the constraints (i.e., hyperplanes) are explicitly given, is an* LP*-type problem and has combinatorial dimension* $\delta = d + 1$.

Proof: We prove something stronger, namely that the problem can be written as a linear program, implying it is an LP-type problem. Let $\mathcal{H}$ be the set of n hyperplanes. For each hyperplane $h \in \mathcal{H}$, let $\langle a_h, x \rangle + b_h = 0$ be the equation describing h, where $a_h \in \mathbb{R}^d$, $\|a_h\| = 1$, and $b_h \in \mathbb{R}$. Because of the requirement that $\|a_h\| = 1$, for a given point $p \in \mathbb{R}^d$, the distance from p to a hyperplane h is $|\langle a_h, p \rangle + b_h|$.

The linear program has $d+1$ variables and $2n$ constraints. The $d+1$ variables represent the center $p \in \mathbb{R}^d$ and radius $r \geq 0$ of the egg. The resulting LP is

$$\begin{aligned}
\min \quad & r & \\
\text{subject to} \quad & r \geq \langle a_h, p \rangle + b_h & \forall h \in \mathcal{H} \\
& r \geq -\left(\langle a_h, p \rangle + b_h\right) & \forall h \in \mathcal{H} \\
& p \in \mathbb{R}^d. &
\end{aligned}$$

As for the combinatorial dimension, observe that any basic feasible solution for the above linear program will be tight for at most $d+1$ of the above $2n$ constraints. Namely, these $d+1$ planes are tangent to the optimal radius ball, and as such form a basis $\mathcal{b} \subseteq \mathcal{H}$. QED.

To apply Lemma 5.1 we need to: (i) design an appropriate input space, (ii) develop a decider, and (iii) construct a constant number of subproblems which cover the constraint space.

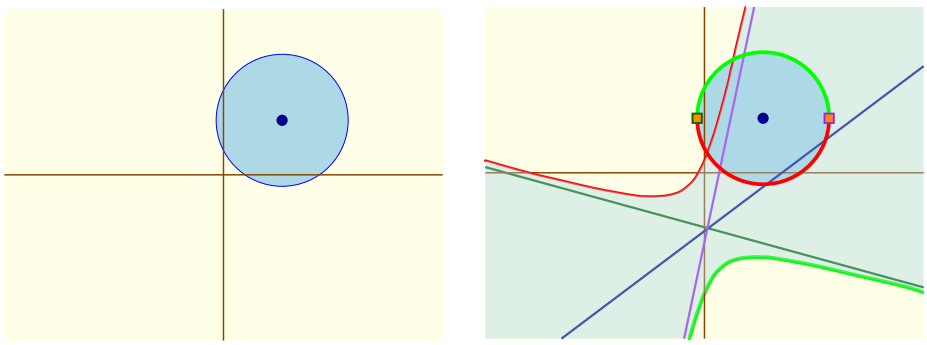


Figure 5.2: A disk and its dual.

5.3.1 Building the decider

The algorithm will work in the dual space. In the dual, the interior of a ball b corresponds to a closed region $\mathsf{b}^\star$ which lies between two branches of a hyperboloid, see Figure 5.2.

Lemma 5.5. *The dual of the set of points in a ball is the set of hyperplanes whose union forms the region enclosed between two branches of a hyperboloid.*

Proof: In $\mathbb{R}^d$ the hyperplane h defined by $x_d = \beta + \sum_{i=1}^{d-1} \alpha_i x_i$, or more compactly $\langle x, (-\alpha, 1)\rangle = \beta$, intersects a disk b centered at $p = (p_1, \ldots, p_d)$ with radius $r \iff$ the distance of h from p is at most r. That is, h intersects b if

$$\begin{aligned}
\frac{|\langle p, (-\alpha, 1)\rangle - \beta|}{\|(-\alpha, 1)\|} \leq r &\iff (\langle p, (-\alpha, 1)\rangle - \beta)^2 \leq r^2 \|(-\alpha, 1)\|^2 \\
&\iff \left(p_d - \beta - \sum_{i=1}^{d-1} \alpha_i p_i\right)^2 \leq r^2(\|\alpha\|^2 + 1). \\
&\iff \frac{\left(p_d - \beta - \sum_{i=1}^{d-1} \alpha_i p_i\right)^2}{r^2} - \|\alpha\|^2 \leq 1.
\end{aligned}$$

The boundary of the above inequality is a hyperboloid in the variables $p_d - \beta - \sum_{i=1}^{d-1} \alpha_i p_i$ and $\alpha_1, \ldots, \alpha_{d-1}$. This corresponds to an affine image of a hyperboloid in the dual space $\alpha \times -\beta$. QED.

Throughout, we let $\mathsf{b}^\star$ denote the region between the two branches of the hyperboloid dual to a ball b.

Algorithm Given a candidate solution (i.e., a ball b in the primal) and a collection of points $Q \subseteq P$. Our goal is to construct a decider which detects if there is a hyperplane of $E_k(P)$, passing through d points of Q, which avoids the interior of the ball b. In the dual setting, the problem is to decide if there is a vertex of $\mathcal{A}(Q^\star)$ which is a member of the k-level, and is inside the region $\mathbb{R}^d \setminus \mathsf{b}^\star$.

The input The input to the algorithm is a simplex Δ, the set of hyperplanes

$$H = P^\star \cap \Delta = \{h \in P^\star \mid h \cap \Delta \neq \emptyset\}$$

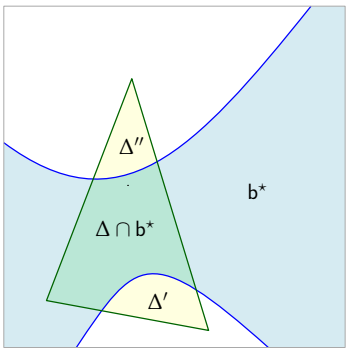


Figure 5.3: The region $\Delta \cap (\mathbb{R}^d \setminus \mathsf{b}^\star)$ consists of (at most) two disjoint convex regions, Δ' and Δ''.

(i.e., all hyperplanes of $P^\star$ that intersect Δ), a candidate solution $\mathsf{b}^\star$, and a parameter u which is the number of hyperplanes of $P^\star$ lying completely below Δ.

The task Decide if there is a vertex of $\mathcal{A}(P^\star)$ of the k-level in $\Delta \cap (\mathbb{R}^d \setminus \mathsf{b}^\star)$. That is, there is a vertex of level k that is outside $\mathsf{b}^\star$ but inside Δ.

The decision procedure Consider the set $\Delta \cap (\mathbb{R}^d \setminus \mathsf{b}^\star)$, where Δ is a simplex, and notice that the set is the union of at most two convex regions. Indeed, the set $\mathbb{R}^d \setminus \mathsf{b}^\star$ consists of two disjoint connected components, where each component is a convex body. Intersecting a simplex Δ with each component of $\mathbb{R}^d \setminus \mathsf{b}^\star$ produces two (disjoint) convex bodies Δ' and Δ'' (it is possible that Δ' or Δ'' are empty). See Figure 5.3. Let Δ' be one of these two regions of interest. The algorithm will process Δ'' in exactly the same way.

If Δ' is empty, then no constraints are violated. Otherwise, we need to check for any violated constraints inside Δ'. Let $\partial\Delta'$ denote the boundary of Δ'. Define $H' \subseteq H$ to be the subset of hyperplanes intersecting Δ'. Observe that it suffices to check if there is a vertex v in the arrangement $\mathcal{A}(H')$ such that: (i) v has level k in $P^\star$, (ii) v is a member of some cell in the zone $\mathcal{Z}(\partial\Delta', H')$, and (iii) v is contained in Δ'.

The algorithm computes $\mathcal{Z}(\partial\Delta', H')$. Next, it chooses a vertex v of the arrangement $\mathcal{A}(H')$ which lies inside Δ' and computes its level in H' (adding u to the count). The algorithm then walks around the vertices of the zone *inside* Δ', computing the level of each vertex along the walk. Note that the level between any two adjacent vertices in the arrangement differ by at most a constant (depending on d). If at any point we find a vertex of the desired level (such a vertex also lies inside Δ'), we report the corresponding median hyperplane which violates the given ball b. See Figure 5.4 for an illustration.

Analysis The running time of the algorithm is proportional to the complexity of the zone $\mathcal{Z}(\partial\Delta', H')$. Because the boundary of Δ' is constructed from $d+1$ hyperplanes and the boundary of the hyperboloid, Lemma 5.2 implies that the zone complexity is no more than $O_d(|H|^{d-1} \log |H|)$. As such, our decision procedure runs in time $D(n) = O_d(n^{d-1} \log n)$.

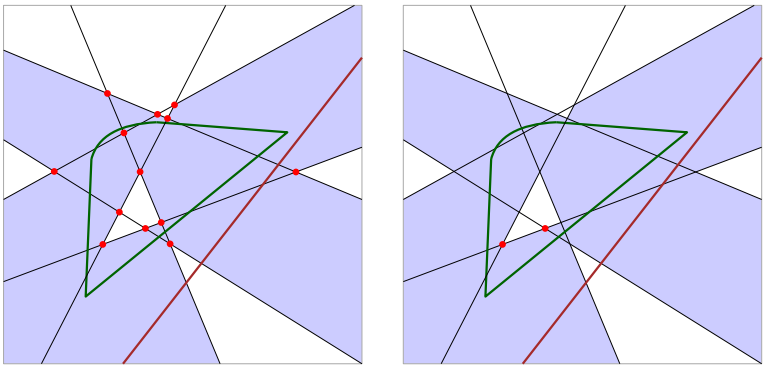


Figure 5.4: Left: A convex region Δ'. Let H' be the set of lines intersecting Δ', with one line lying completely below Δ' ($\mathsf{u} = 1$). The shaded regions are the cells of $\mathcal{A}(H')$ intersecting $\partial\Delta'$. The vertices of the cells in the zone $\mathcal{Z}(\partial\Delta', H')$ are highlighted. Right: The vertices of $\mathcal{Z}(\partial\Delta', H')$ which are part of the 3-level and contained inside Δ'.

5.3.2 Constructing subproblems

To decompose a given input into smaller subproblems, we need the notion of cuttings.

Definition 5.6 (Cuttings). Given n hyperplanes in $\mathbb{R}^d$, a ***$(1/c)$-cutting*** is a collection of interior disjoint simplices covering $\mathbb{R}^d$, such that each simplex intersects at most n/c hyperplanes. A $(1/c)$-cutting of size $O_d(c^d)$ can be constructed in $O_d(nc^{d-1})$ time [45].

Given a simplex Δ and the set of hyperplanes $H = P^\star \cap \Delta$, we compute a $(1/c)$-cutting of H into $O_d(c^d)$ simplices, and clip this cutting inside Δ. For each cell in this new cutting, we compute the set of hyperplanes which intersect it, and the number of hyperplanes lying completely below the cell naively in $O(|H|)$ time. Repeating this process for the $O_d(c^d)$ cells implies that this decomposition procedure can be completed in $O(|H|)$ time (ignoring dependencies on d), as $(1/c)$-cuttings can be constructed deterministically in time $O_d(n)$ for constant c [45].

The above shows that we can decompose a given input of size n into $\psi = O_d(c^d)$ subproblems, each of size at most n/c. Furthermore, this decomposition preserves all implicit constraints of interest (vertices of $\mathcal{A}(H)$). Choosing c to be a sufficiently large constant (possibly depending on d), to meet the requirements of Lemma 5.1, finishes the construction.

5.3.3 Putting it all together

The above discussions together with Lemma 5.1 and $D(n) = O_d(n^{d-1}\log n)$ implies the following.

Lemma 5.6. *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. For a given integer k, one can compute in $O_d(n^{d-1}\log n)$ expected time the smallest radius ball intersecting all of the hyperplanes of $E_k(P)$*

Corollary 5.1. *Let $P \subset \mathbb{R}^d$ be a set of n points in general position, and let $S \subseteq [\![n]\!]$, where $[\![n]\!] = \{1, \ldots, n\}$. One can compute in $O_d(n^{d-1}\log n)$ expected time the smallest radius ball intersecting all of the hyperplanes of $\bigcup_{k\in S} E_k(P)$*

Proof: The algorithm is a slight modification of Lemma 5.6. During the decision procedure, for each vertex in the zone, we check if it is a member of the k-level for some $k \in S$. If S is of non-constant size, membership in S can be checked in constant time using hashing. QED.

5.3.4 Computing the extremal yolk and the egg

Theorem 5.1. *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. One can compute the extremal yolk of P in $O_d(n^{d-1} \log n)$ expected time.*

Proof: The result follows by applying Corollary 5.1 with the appropriate choice of S. When n is even, Lemma 5.3 tells us to choose $S = \{n/2, n/2+1, \ldots, n/2+d\}$. When n is odd, we set $S = \{\lceil n/2 \rceil, \lceil n/2 \rceil + 1, \ldots, \lceil n/2 \rceil + d - 1\}$. QED.

Lemma 5.7. *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. One can compute the egg of P in $O_d(n^{d-1} \log n)$ expected time.*

Proof: Follows by Corollary 5.1 with $S = [\![n]\!]$. (Alternatively, by directly modifying the decision procedure to check if any vertex of the zone $\mathcal{Z}(\Delta', H')$ lies inside Δ'.) QED.

5.3.5 An algorithm sensitive to k

Recall that to compute the extremal yolk, we reduced the problem to computing the smallest ball intersecting all hyperplanes which contain a fixed number of points of P above or on them (see Lemma 5.3). In particular, we developed an algorithm for Problem 5.1 and applied it when k is proportional to n. It is natural to ask for an algorithm for Problem 5.1 which is faster when k is small. The algorithm will work for all values of k. However when k is large, the running time deteriorates to the running time of the algorithm of Lemma 5.6.

To develop an algorithm sensitive to k, we use the result of Lemma 5.6 as a black-box and introduce the notion of shallow cuttings.

Definition 5.7 (Shallow cuttings). Let H be a set of n hyperplanes in $\mathbb{R}^d$. A ***k-shallow cutting*** is a collection of simplices such that: (i) the union of the simplices covers the $(\leq k)$-level of H (see Definition 5.3), and (ii) each simplex intersects at most k hyperplanes of H.

Matoušek was the first to prove existence of k-shallow cuttings of size $O_d((n/k)^{\lfloor d/2 \rfloor})$ [113]. When $d = 2, 3$, a k-shallow cutting of size $O(n/k)$ can be constructed in $O(n \log n)$ time [40]. A similar result holds in higher dimensions (the proof is sketched in [79]).

Lemma 5.8 ([79]). *Let H be a set of n hyperplanes in $\mathbb{R}^d$. A k-shallow cutting of size $O_d((n/k)^{\lfloor d/2 \rfloor})$ can be constructed in $O_d(k(n/k)^{\lfloor d/2 \rfloor} + n \log n)$ expected time. For each simplex Δ in the cutting, the algorithm returns the set of hyperplanes intersecting Δ and the number of hyperplanes lying below Δ.*

Let $P \subset \mathbb{R}^d$ be a set of n points and let $H = P^\star$ be the set of dual hyperplanes. The algorithm itself is a randomized incremental algorithm, mimicking Seidel's algorithm for solving LPs [141]. First, compute a k-shallow cutting for the set of hyperplanes H using Lemma 5.8. Let $\Delta_1, \ldots, \Delta_\ell$, where $\ell = O_d((n/k)^{\lfloor d/2 \rfloor})$, be the collection of simplices in the cutting. For each simplex Δ_i, we have the subset $H \cap \Delta_i$ and the number of hyperplanes lying completely below H (which is at most k). For each cell Δ_i, let $g(\Delta_i)$ be the set of vertices of $\mathcal{A}(H)$ which have level k and are contained in Δ_i.

The algorithm The input to the algorithm is a set of simplices and an initial ball b_0. Such a ball is uniquely defined by a subset of $d+1$ constraints, and this is a basis for the LP-type problem.

Begin by randomly permuting the simplices $\Delta_1, \ldots, \Delta_\ell$. At all times, the algorithm maintains a ball b_i of smallest radius which meets all the constraints defined by $\cup_{j=1}^{i} g(\Delta_i)$ in the dual. In the ith iteration, the algorithm performs a *violation test*: it decides if any constraint of $g(\Delta_i)$ is violated by $\mathsf{b}_{i-1}^\star$. If so, the algorithm executes a *basis computation*, in which it computes the ball b_i' of smallest radius which obeys the constraints of $g(\Delta_i)$ in the dual and the $d+1$ constraints defining b_{i-1}. The algorithm then computes a ball b_i by invoking itself recursively on the subset of cells $\Delta_1, \ldots, \Delta_i$ with b_i' as the initial basis.

Lemma 5.9. *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. For a given integer k, one can compute in $\widetilde{O}_d\left(k^{d-1}\left(1 + (n/k)^{\lfloor d/2 \rfloor}\right)\right)$ expected time the smallest radius ball intersecting all of the hyperplanes of $E_k(P)$.*

Proof: The algorithm is described above. As for the analysis, it is similar to any randomized incremental algorithm for LP-type problems. The key difference is that we are not adding a single constraint incrementally, but rather a collection of constraints in each iteration. Fortunately, this does not change the analysis of the algorithm (for further details, see the proof of Lemma 5.1 in [33] or [79]).

It is known that in expectation, the algorithm performs $O_d((n/k)^{\lfloor d/2 \rfloor})$ violation tests and $O_d(\log^{d+1}(n/k))$ basis computations [145]. Since each simplex Δ_i intersects $O(k)$ hyperplanes of H, each of these subroutines can be implemented in $O_d(k^{d-1} \log k)$ time using Lemma 5.6. Finally, we account for the time needed to construct the shallow cutting—by Lemma 5.8 this can be done in $O_d(k(n/k)^{\lfloor d/2 \rfloor} + n \log n)$ expected time. QED.

5.4 COMPUTING THE (CONTINUOUS) YOLK

Definition 5.8. Let $P \subset \mathbb{R}^d$ be a set of n points in general position. The ***continuous yolk*** of P is the ball of smallest radius intersecting all median hyperplanes of P.

In contrast to Definition 5.5, we emphasize that the (continuous) yolk must intersect all median hyperplanes defined by P (not just extremal median hyperplanes).

As before, the algorithm works in the dual space. For an integer k, let $H_k(P)$ be the collection of halfspaces containing exactly k points of P on or above it. Equivalently, $P^\star$ is the collection of hyperplanes defined

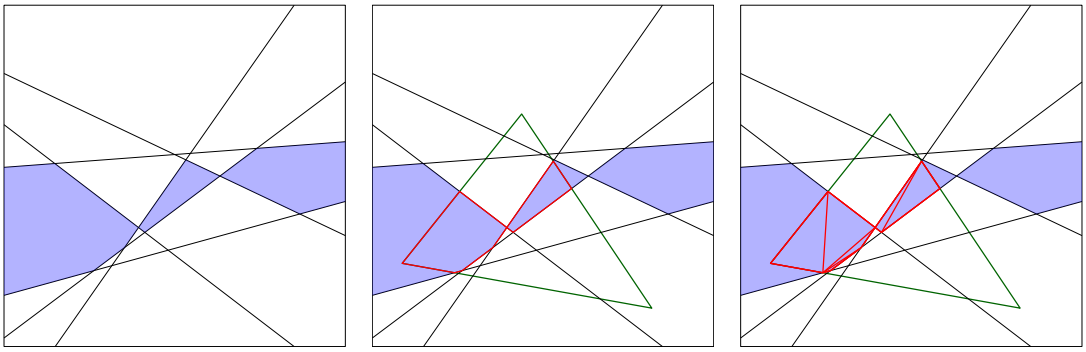


Figure 5.5: Left: A set of lines and the cells of the 3-level. Middle: A simplex Δ, with the portion of the 3-level inside Δ. Right: Triangulating the portion of the 3-level contained inside Δ. All red triangles together with the lower dimensional faces of the 3-level form the set of constraints $g(\Delta)$.

by P in the dual space, and $\big(H_k(P)\big)^\star$ is the k-level of $P^\star$. Our problem can be restated in the dual space as follows.

Problem 5.2. Let P be a set of points in $\mathbb{R}^d$ in general position and let k be a given integer. Compute the ball b of smallest radius so that all points in the k-level of $P^\star$ are contained inside the region $\mathsf{b}^\star$.

Let $L_k(P) = \big(H_k(P)\big)^\star$ denote the set of all points in the k-level of $P^\star$. Note that $L_k(P)$ consists of points which are either contained in the interior of some ℓ-dimensional flat, where $0 \leq \ell \leq d-1$, or in the interior of some d-dimensional cell of $\mathcal{A}(P^\star)$.

We take the same approach as the algorithm of Theorem 5.1—building a decider subroutine, and showing that the input space can be decomposed into subproblems efficiently. However the problem is more subtle, as the collection of constraints (i.e., median hyperplanes) is no longer a finite set.

The input space The input consists of a simplex Δ. The algorithm, in addition to Δ, maintains the set of hyperplanes

$$H = P^\star \cap \Delta = \{h \in P^\star \mid h \cap \Delta \neq \emptyset\},$$

and a parameter u which is equal to the number of hyperplanes of $P^\star$ lying completely below Δ.

The implicit constraint space Each input Δ maps to a region R which is the portion of the k-level $L_k(P)$ contained inside Δ. For each d-dimensional cell in R, we compute its bottom-vertex triangulation (see, e.g., [112, Section 6.5]), and collect all of these simplices, and all lower dimensional faces of R, into a set $g(\Delta)$, see Figure 5.5.

Let Ξ be the collection of all simplices formed from $d+1$ vertices of the arrangement $\mathcal{A}(P^\star)$. We let $\mathcal{H}$ be the union of the sets $g(\Delta)$ over all simplices $\Delta \in \Xi$. To see why this suffices, each simplex in the input space is a simplex generated by a cutting algorithm. One property of cutting algorithms [45] is that the simplices returned are induced by hyperplanes of $P^\star$. Indeed, each simplex has (at most) $d+1$ vertices, and upon

inspection of the cutting algorithm, each vertex is defined by d hyperplanes of $P^\star$. There are a finite number of simplices Δ to consider, and each Δ induces a fixed subset of constraints $g(\Delta) \subseteq \mathcal{H}$.

As such, $\mathcal{H}$ forms our constraint set, where each constraint is of constant size (depending on d). Clearly, a solution satisfies all constraints of $\mathcal{H}$ if and only if the solution intersects all hyperplanes in the set $H_k(P)$. For a given subset $\mathcal{C} \subseteq \mathcal{H}$, the objective function is the minimum radius ball b such that all regions of $\mathcal{C}$ are contained inside the region $\mathsf{b}^\star$. In particular, the problem of computing the minimum radius ball b such that $\mathsf{b}^\star$ contains all points of $L_k(P)$ in its interior is an LP-type problem of constant combinatorial dimension.

Constructing subproblems For a given input simplex Δ (along with the set $H = P^\star \cap \Delta$ and the number u) a collection of subproblems $\Delta_1, \ldots, \Delta_\psi$ (with the corresponding sets H_i and numbers u_i for $i = 1, \ldots, \psi$) can be constructed as described in Section 5.3.2, by computing a cutting of the planes H and clipping this cutting inside Δ. In particular, we have that $\bigcup_i g(\Delta_i) = g(\Delta)$. Strictly speaking, we have not decomposed the constraints of $g(\Delta)$ (as required by Lemma 5.1), but rather have decomposed the region which is the union of the constraints of $g(\Delta)$. This step is valid, as a solution satisfies the constraints of $\bigcup_i g(\Delta_i)$ if and only if it satisfies the constraints of $g(\Delta)$.

The decision procedure Given a candidate solution $\mathsf{b}^\star$, the problem is to decide if $\mathsf{b}^\star$ contains $g(\Delta)$ in its interior. The decision algorithm itself is similar as in the proof of Theorem 5.1. Consider the set $\Delta \cap (\mathbb{R}^d \setminus \mathsf{b}^\star)$, where Δ is a simplex, and notice that it is the union of the most two convex regions. Let Δ' be one of these two regions of interest. Observe that it suffices to check if there is a point on the boundary of Δ' which is part of the k-level. Let $H' \subseteq H$ be the subset of hyperplanes intersecting Δ'.

To this end, compute $\mathcal{Z}(\partial\Delta', H')$. For each $(d-1)$-dimensional face f of Δ', the collection of regions $\Xi = \{f \cap s \mid s \in \mathcal{Z}(\partial\Delta', H')\}$ forms a $(d-1)$-dimensional arrangement restricted to f. Furthermore, the complexity of this arrangement lying on f is at most $O(n^{d-1} \log n)$. Notice that the level of all points in the interior of a face of Ξ is constant, and two adjacent faces (sharing a boundary) have their level differ by at most a constant. The algorithm picks a face in Ξ, computes the level of an arbitrary point inside it (adding u to the count). Then, the algorithm walks around the arrangement, exploring all faces, using the level of neighboring faces to compute the level of the current face. If at any step a face has level k, we report that the input (Δ, H, u) violates the candidate solution $\mathsf{b}^\star$.

Analysis of the decision procedure We claim the running time of the algorithm is proportional to the complexity of the zone $\mathcal{Z}(\partial\Delta', H')$. Indeed, for each $(d-1)$-dimensional face f of Δ' (where f may either be part of a hyperplane or part of the boundary of $\mathsf{b}^\star$), we can compute the set $\{f \cap s \mid s \in \mathcal{Z}(\partial\Delta', H')\}$ in time proportional to the total complexity of $\mathcal{Z}(\partial\Delta', H')$ (assuming we can intersect a hyperplane with a portion of a constant degree surface efficiently). The algorithm then computes the level of an initial face naively in $O_d(|H'|)$ time, and computing the level of all other faces can be done in $O_d(|\mathcal{Z}(\partial\Delta', H')|)$ time by

performing a graph search on the arrangement.

Because the boundary of Δ' is constructed from $d+1$ hyperplanes and the boundary of the hyperboloid, Lemma 5.2 implies that the zone complexity is $O_d(|H|^{d-1}\log|H|)$. As such, our decision procedure runs in time $D(n) = O_d(n^{d-1}\log n)$.

Lemma 5.10. *Problem 5.2 can be solved in $O_d(n^{d-1}\log n)$ expected time, where $n = |P|$.*

Proof: Follows by plugging the above discussion into Lemma 5.1. QED.

By modifying the decision procedure appropriately, we also obtain a similar result to Corollary 5.1.

Corollary 5.2. *Let $P \subset \mathbb{R}^d$ be a set of n points in general position, and let $S \subset [\![n]\!]$. The smallest ball intersecting all hyperplanes in $\bigcup_{k\in S} H_k(P)$ can be computed in $O_d(n^{d-1}\log n)$ expected time.*

Theorem 5.2. *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. One can compute the yolk of P in $O_d(n^{d-1}\log n)$ expected time.*

Proof: The result follows by applying Corollary 5.2 with the appropriate choice of S. When n is even, Lemma 5.3 tells us to choose $S = \{n/2, n/2+1, \ldots, n/2+d\}$. When n is odd, we set $S = \{\lceil n/2\rceil, \lceil n/2\rceil + 1, \ldots, \lceil n/2\rceil + d - 1\}$. QED.

Remark 5.1. In $\mathbb{R}^3$, one can shave the $O(\log n)$ factor to obtain an $O(n^2)$ expected time algorithm for the yolk. We modify the decision procedure as follows, which avoids computing the zone $\mathcal{Z}(\partial\Delta', H')$. For each 2D face f of Δ', simply compute the arrangement of the set of lines $\{f \cap h \mid h \in H\}$ on f in $O(n^2)$ time. As before, we perform a graph search on this arrangement, computing the level of each face. If any time we discover a point on the boundary of Δ' of the desired level, we report that the given input violates the given candidate solution.

5.5 COMPUTING THE TUKEY BALL AND CENTER BALL

For a given point $q \in \mathbb{R}^d$ and point set $P \subset \mathbb{R}^d$, the ***Tukey depth*** of q is the largest integer k such that any closed halfspace γ containing q must contain at least k points of P. Equivalently, if $H_k(P)$ is the set of all open halfspaces containing more than $n-k$ points of P, then any point contained in the intersection $\mathcal{T}_k = \bigcap\{\gamma \mid \gamma \in H_k(P)\}$ is a point of Tukey depth at least k. The centerpoint theorem implies that there is always a point of Tukey depth at least $n/(d+1)$.

Definition 5.9. Let $P \subset \mathbb{R}^d$ be a set of n points in general position. For a parameter $k \leq n$, the ***Tukey ball*** of P is the smallest radius ball intersecting halfspaces in the set $H_k(P)$.

The Tukey median is a point in $\mathbb{R}^d$ with maximum Tukey depth. If the Tukey median of P has Tukey depth $k(P)$, then for $k > k(P)$ the set $\mathcal{T}_k$ is empty—the Tukey ball has non-zero radius. When $k \leq k(P)$, $\mathcal{T}_k$ is non-empty, implying that the Tukey ball has radius zero.

Definition 5.10. Let $P \subset \mathbb{R}^d$ be a set of n points in general position. For a parameter $k \leq k(P)$, the ***center ball*** of P is denoted as the ball of largest radius contained in the region $\mathcal{T}_k$.

Recently, Oh and Ahn [128] develop a $O(n^2 \log^4 n)$ time algorithm for computing the polytope $\mathcal{T}_k$ in $\mathbb{R}^3$ when $k = n/(d+1) = n/4$. In contrast, the center ball is the largest ball contained inside $\mathcal{T}_k$, and we show it can be computed in expected time $O(n^2 \log n)$.

5.5.1 The Tukey ball in the dual

For a set of n points P in general position, it suffices to restrict our attention to hyperplanes which contain d points of P, and one of the open halfspaces contains more than $n-k$ points of P. In the dual, each point $p \in P$ is mapped to a hyperplane $p^\star$ (see Definition 5.2). A hyperplane h passing through d points of P maps to a point $h^\star$ which is a vertex in the arrangement $\mathcal{A}(P^\star)$.

Definition 5.11 (Top and bottom levels). Let H be a set of hyperplanes in $\mathbb{R}^d$. The top-level (resp. bottom-level) of a point $p \in \mathbb{R}^d$ is the number of hyperplanes of H lying above (resp. below) p. The ***k-top level*** (resp. ***k-bottom level***), denoted as $T_k(H)$ (resp. $B_k(H)$), is the set of vertices of $\mathcal{A}(H)$ which have top-level (resp. bottom-level) equal to k.

Recall that by Lemma 5.5, a ball b in the primal maps to the region enclosed by two branches of a hyperboloid. Formally, the region $\mathsf{b}^\star$ is the collection of points $(x_1, \ldots, x_d) \in \mathbb{R}^d$ satisfying has the equation $(x_d/\alpha_d)^2 - \sum_{i=1}^{d-1}(x_i/\alpha_i)^2 \leq 1$, where $\alpha_1, \ldots, \alpha_d \in \mathbb{R}$ define the hyperboloid, and are determined by b. We say that a point $(x_1, \ldots, x_d)$ lies above the top branch of $\mathsf{b}^\star$ if the inequality $x_d \geq \alpha_d \sqrt{1 + \sum_{i=1}^{d-1}(x_i/\alpha_i)^2}$ holds. A point lying below the bottom branch of $\mathsf{b}^\star$ is defined analogously.

Let h be a hyperplane. Suppose the open halfspace h^- below h contains k points of P. In the dual, a vertical ray ρ_h shooting upwards from the point $h^\star$ intersects k hyperplanes of $P^\star$. When a hyperplane h intersects b in its interior, then $\mathsf{b} \cap h^- \neq \varnothing$ and $\mathsf{b} \not\subseteq h^-$. In the dual, $\mathsf{b}^\star$ contains the point $h^\star$, and the upward ray ρ_h intersects the boundary of $\mathsf{b}^\star$ once. Alternatively, if $\mathsf{b} \subseteq h^-$, then in the dual the upward ray ρ_h intersecting the boundary of $\mathsf{b}^\star$ twice (once each at the top and bottom branch). As such, if h^- is an open halfspace containing k points of P below it and does not intersect b, then the upward ray ρ_h does not intersect the boundary of $\mathsf{b}^\star$. Hence, ρ_h must lie entirely above the top branch of $\mathsf{b}^\star$. See Figure 5.6.

Summarizing the above discussion, the problem of computing the Tukey ball is equivalent to the following.

Problem 5.3. Let $P \subset \mathbb{R}^d$ be a set of n points in general position. The goal is to compute the ball b of smallest radius such that:

(I) for each vertex of the k-top level $T_k(P^\star)$, the vertical upward ray intersects $\mathsf{b}^\star$, and

(II) for each vertex of the k-bottom level $B_k(P^\star)$, the vertical downward ray intersects $\mathsf{b}^\star$.

Lemma 5.11. *Let $P \subset \mathbb{R}^d$ be a set of n points in general position and $k \leq n$ a parameter. The Tukey ball can be computed in $O_d(n^{d-1} \log n)$ expected time.*

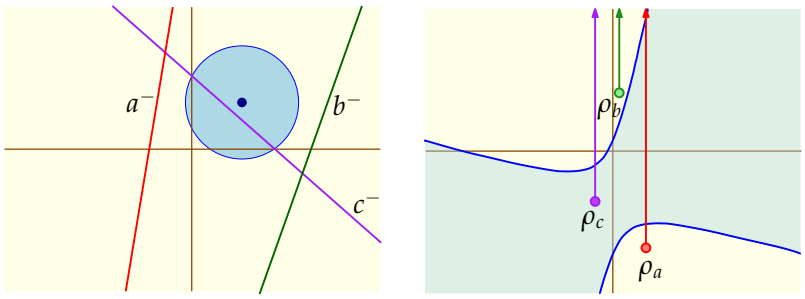


Figure 5.6: A ball and three lines. Each line induces a halfspace which lies below the line. In the dual, this corresponds to three vertically upward rays.

Proof: The proof uses Lemma 5.1 to solve the dual problem (this problem is LP-type with constant combinatorial dimension, where the constant depends on d). The input consists of a simplex Δ. A given input can be decomposed using cuttings, as in the algorithms for Theorem 5.1 and Theorem 5.2.

We sketch the decision procedure. Let $H = P^\star \cap \Delta$. Given a candidate ball b, we want to decide if $\mathsf{b}^\star$ violates any constraints induced by H. Equivalently, $\mathsf{b}^\star$ is an invalid solution if either condition holds: (i) there is a element of $T_k(P^\star)$ which is above the top branch of $\mathsf{b}^\star$, or (ii) there is a element of $B_k(P^\star)$ which is below the bottom branch of $\mathsf{b}^\star$. As such, a straight-forward modification of the decision procedure described in Section 5.3.1 yields a decider in $O_d(|H|^{d-1} \log |H|)$ expected time. QED.

Improved algorithm

Lemma 5.12. *Let $P \subset \mathbb{R}^d$ be a set of n points in general position and $k \leq n$ a parameter. The Tukey ball can be computed in $\widetilde{O}_d\left(k^{d-1}\left(1 + (n/k)^{\lfloor d/2 \rfloor}\right)\right)$ expected time.*

Proof: The algorithm is the same as described in Lemma 5.9 with a small change: compute a shallow cutting for the $(\leq k)$-top level and $(\leq k)$-bottom level of $P^\star$. Now run the randomized incremental algorithm of Lemma 5.9 on these collection of simplices with Lemma 5.11 as a black-box to solve the subproblems of smaller size. QED.

5.5.2 The center ball in the dual

For a parameter k, recall that our goal is to compute the largest ball which lies inside all open halfspaces containing more than $n - k$ points of P. From the discussion above, in the dual this corresponds to the following problem.

Problem 5.4. Let $P \subset \mathbb{R}^d$ be a set of n points in general position. The goal is to compute the ball b of largest radius such that:

(I) each vertex of the k-top level $T_k(P^\star)$ lies below the bottom branch of $\mathsf{b}^\star$, and

(II) each vertex of the k-bottom level $B_k(P^\star)$ lies above the top branch of $\mathsf{b}^\star$.

Lemma 5.13. *Let $P \subset \mathbb{R}^d$ be a set of n points in general position and $k \leq n$ a parameter. The center ball can be computed in $O_d(n^{d-1} \log n)$ expected time.*

Proof: As usual, we use Lemma 5.1 to solve the dual problem (this problem is LP-type with constant combinatorial dimension, where the constant depends on d). The input consists of a simplex Δ. A given input can be decomposed using cuttings, as in the algorithms for Theorem 5.1 and Theorem 5.2.

We sketch the decision procedure. Let $H = P^\star \cap \Delta$. Assume that we also know the number of hyperplanes lying below and above Δ. We are also given a candidate ball b. The algorithm computes the zone $\mathcal{Z}(\partial\Delta, H)$ and computes the level of each vertex of $\mathcal{Z}(\partial\Delta, H)$ inside Δ. If we find a vertex of either the k-top or k-bottom level which also lies inside $\mathsf{b}^\star$, we report the violated constraint. Otherwise, if we find a vertex of the k-top level lying above the top branch of $\mathsf{b}^\star$ or a vertex of the k-bottom level lying below the bottom branch of $\mathsf{b}^\star$, then the solution b is also deemed infeasible. This decision procedure can be implemented in $O_d(|H|^{d-1} \log |H|)$ expected time. QED.

Improved algorithm

Lemma 5.14. *Let $P \subset \mathbb{R}^d$ be a set of n points in general position and $k \leq n$ a parameter. The center ball can be computed in $\widetilde{O}_d\left(k^{d-1}\left(1 + (n/k)^{\lfloor d/2 \rfloor}\right)\right)$ expected time.*

Proof: The same argument for Lemma 5.12 applies here, using Lemma 5.13 as a black-box to solve the subproblems generated by the k-shallow cutting of the $(\leq k)$-top level and $(\leq k)$-bottom level. QED.

5.6 COMPUTING THE K-BALL

Lemma 5.15. *Let P be a set of n points in $\mathbb{R}^d$ in general position, and let k be a parameter. For a set $X \subseteq \mathbb{R}^d$, let $\mathsf{conv}(X)$ denote the convex-hull of X. Let $\mathcal{C}(P,k) = \left\{\mathsf{conv}(Q) \mid Q \in \binom{P}{k}\right\}$. For $d \geq 4$, one can compute the minimum radius ball intersecting all of the sets of $\mathcal{C}(P,k)$ in $O_d(n^{\lfloor d/2 \rfloor} k^{2d})$ expected time. For $d \in [2,3]$, the expected running time is $O(nk^{2d} \log k + n \log n)$.*

Proof: Naively, there are $\binom{n}{k}$ sets that one has to consider. However, consider an optimal solution (i.e., a ball b in $\mathbb{R}^d$) which is tangent to $d+1$ sets of $\mathcal{C}(P,k)$. Fix a subset $Q \subseteq P$ of size k such that $\mathsf{conv}(Q)$ is tangent to b. Let h be the common tangent hyperplane to b and $\mathsf{conv}(Q)$, and let h^+ be its closed halfspace that does not contain the interior of b. We have that $Q \subseteq h^+ \cap P$. Under general position assumptions, if $|h^+ \cap P|$ contains more than $k+d$ points, then there is a subset $R \subseteq P$ of k points that is fully contained in the interior of h^+. But then $\mathsf{conv}(R)$ does not intersect b, contradicting feasibility.

This implies that all the constraints that can participate in defining the optimal solution are subsets of k points of a set $h^+ \cap P$, where $h^+ \cap P$ is of size at most $k+d$. As such, let H denote the collection of all halfspaces containing at most $k+d$ points of P. It is known that $|H| = O_d(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ [47]. Furthermore,

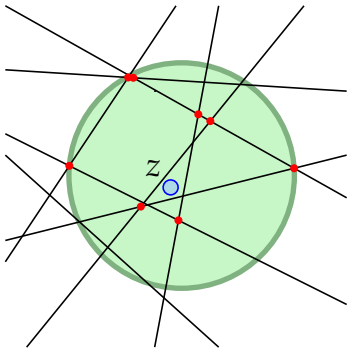


Figure 5.7: A disk containing all vertices of $\mathcal{A}(L)$ lying within crossing distance at most three from z.

the collection of halfspaces can be computed in $O(|H|)$ expected time for $d \geq 4$ [123] (for $d \in [2,3]$, H can be constructed in time $O(|H| + n \log n)$ [36, 60]).

Finally, each halfspace $h^+ \in H$ induces $\binom{|h^+ \cap P|}{k} \leq \binom{k+d}{k} = O(k^d)$ constraints. This implies there are at most $O_d(n^{\lfloor d/2 \rfloor} k^{d+\lceil d/2 \rceil})$ constraints of interest. Compute this set of constraints explicitly and for each constraint consisting of k points, compute their convex hull in $O_d(k^{\lfloor d/2 \rfloor} + k \log k)$ time [44]. Finally, solve the resulting LP-type problem in time proportional to the number of constraints [78, 141]. QED.

5.7 SMALLEST DISK ENCLOSING ALL VERTICES WITH CROSSING DISTANCE $\leq K$

Let L be a set of lines in the plane. For two points $p, z \in \mathbb{R}^2$, the ***crossing distance*** $\mathsf{d}_L(p,z)$ is the number of lines of L intersecting the segment pz.

Given a point $z \in \mathbb{R}^2$ not lying on any line of L, and a parameter k, let

$$S_k(z) = \{p \in \mathsf{V}(\mathcal{A}(L)) \mid \mathsf{d}_L(p,z) \leq k\}$$

be the set of vertices of $\mathcal{A}(L)$ with crossing distance at most k from z. The goal is to compute the smallest disk enclosing all points of $S_k(z)$, see Figure 5.7.

When the constraints (points) are explicitly given, this problem is LP-type with constant combinatorial dimension. We now apply Lemma 5.1 to obtain an efficient algorithm for this problem.

The decision procedure The input is a simplex Δ, the set of lines $L' \subseteq L$ intersecting Δ, and a number u which is the number of lines of L separating Δ and z (specifically, a line ℓ separates Δ and z if they lie on opposite sides of ℓ). Given a candidate disk D, the goal is to determine if there is a vertex of $\mathcal{A}(L')$ which lines outside D and has crossing distance at most k from z.

To this end, compute the zone $\mathcal{Z}(\partial\Delta, L')$. The algorithm chooses a vertex v of $\mathcal{Z}(\partial\Delta, L')$ inside Δ and computes $\mathsf{d}_L(v,z) = \mathsf{d}_{L'}(v,z) + \mathsf{u}$. Next, walk around the set of vertices in $\mathcal{Z}(\partial\Delta, L') \cap \Delta$ and compute the crossing values using previously computed crossing values. If at any time a vertex of crossing value at most

k which is outside D is encountered, report that D is an invalid solution.

The running time of the decision procedure is dominated by computing the zone $\mathcal{Z}(\partial\Delta, L')$, which can be achieved in $O(|L'| \log |L'|)$ time by Lemma 5.2.

Constructing subproblems Given Δ and the lines $L' \subseteq L$ intersecting Δ, we compute a $(1/c)$-cutting of L' and clip this cutting inside Δ. For each cell Δ_i in the new cutting, we compute the lines of L' intersecting Δ_i and the number of lines separating Δ_i from z. By choosing c sufficiently large to meet the requirements of Lemma 5.1, the subproblems can be constructed in at most $O(|L'| \log |L'|)$ time.

Lemma 5.16. *Let L be a set of n lines in the plane and let $z \in \mathbb{R}^2$ be a point not lying on any point of L. In $O(n \log n)$ expected time, one can compute the smallest disk enclosing all vertices of $\mathcal{A}(L)$ within crossing distance at most k from z.*

Part III

Geometric separation

6 Separating points by lines

"In the fields of observation chance favours only the prepared mind.

— Louis Pasteur

For a set P of n points in $\mathbb{R}^d$, a set L of hyperplanes ***separates*** P, if for any pair of points of $p, q \in P$, there is a hyperplane in L that intersects the interior of the segment pq (which also does not contain p or q). In $\mathbb{R}^2$, L is a set of lines. The ***separability*** of P, denoted by $S_n = \text{sep}(P)$, is the size of the smallest set of lines that separates P. The separability of a point set captures how grid-like the point set is. In particular, the separability of the $\sqrt{n} \times \sqrt{n}$ grid is $2\sqrt{n} - 2$, while for n points in convex position the separability is $\lceil n/2 \rceil$ (and this is the worst case assuming general position).

In this chapter, we investigate the separability of a point set. For a collection of n points P chosen uniformly at random from the unit square, where $S_n = \text{sep}(P)$ is now a random variable, we prove that with high probability $S_n = O(n^{2/3})$ and $S_n = \Omega(n^{2/3} \log\log n / \log n)$. This bound also extends to higher dimensions. When $P \subset \mathbb{R}^d$, we show that $S_n = \Omega(n^{2/(d+1)} \log\log n / \log n)$ with high probability. We also give an efficient randomized approximation algorithm for computing the minimum number of separating lines in the plane. For an input set $P \subset \mathbb{R}^2$ of size n with $S_n = \text{sep}(P)$, the algorithm returns a collection of lines separating P of size $O(S_n \log S_n)$ and runs in expected time $O(n^{2/3} S_n^{5/3} \log^{O(1)} n)$.

6.1 BACKGROUND

6.1.1 Grid versus random points

There is a striking similarity between the behavior of random point sets and uniform grid point sets. For example, the convex hull of a set of n random points inside a triangle has $O(\log n)$ vertices in expectation, and the same bound holds for the convex hull of the $\sqrt{n} \times \sqrt{n}$ grid points when clipped to a triangle. There are many other examples of this surprising similarity in behavior (see [74] and references therein). Another striking example of this similarity is in the number of layers of the convex hull—it is $O(n^{2/3})$ for n random points [53], and the same bound holds for a grid of n points [77].

6.1.2 Previous work

Freimer et al. [68] showed that computing the separability of a given point set is NP-complete, and studied an extension of the problem to polygons in the plane. Nandy et al. [127] studied the problem of separating segments. Călinescu et al. [29] gave a two approximation when restricting the problem to separation via axis-parallel lines. There has also been work on the *bichromatic* version of the problem: given a collection of red points and blue points, determine the minimum number of lines needed to separate all red-blue pairs (i.e., so that each cell in the arrangement of lines is monochromatic). This problem is known to be W[1]-hard when the lines have arbitrary slope [25], but it is FPT in the solution size when the lines are restricted to be axis-parallel [100]. Other work on this and related problems includes [54, 64].

6.1.3 Motivation

Separating and breaking point sets, usually into clusters, is a fundamental task in computer science, needed for divide and conquer algorithms. It is thus natural to ask what can be done if restricted to lines, and if one can do the partition in a global fashion (i.e., if the partition is done locally only to the current subproblem, this results in a *binary space partition* (BSP)). Specifically, we have the following connections:

(A) *Geometric hitting set.* The separability problem reduces to a geometric hitting set problem. In recent years there was a lot of work on speeding up approximation algorithms for such problems, and it is a natural question to ask what can be done in this specific case. See [1, 4] and references therein.

(B) *Polynomial partition.* For divide and conquer algorithms for lines, the classical tool to use is cuttings [42], and for points there are partitions [110]. More recently, the polynomial ham-sandwich theorem was used to partition point sets—see [3] and references therein for some recent work. This yields partitions that have stronger properties than the partitions of Matoušek [110] in some cases, but are (in many cases) algorithmically less convenient to use. It is thus natural to ask what else can be done when only using hyperplanes (or lines in $\mathbb{R}^2$).

(C) *Extracting features.* Recently, there was increased interest in *autoencoders* in machine learning—here, one is interested in finding a representation of the data of a set of features, where the number of features is significantly smaller than the ambient dimension, see [89]. Thus, the separability problem can be interpreted as finding a minimum number of linear features, such that all data points are distinguishable. The problem is usually of interest in higher dimensions, but even in constant dimension it is already challenging.

6.2 RESULTS

6.2.1 Low separability implies partitions

In Section 6.3 we define the problem formally, and show how low separability implies partitions (see Definition 6.4) in two and three dimensions with almost optimal parameters. Specifically, if a point set P in $\mathbb{R}^d$ for $d \in \{2,3\}$ has separability $O(n^{1/d})$, then for any $r > 0$, P can be partitioned into $O(r)$ sets, where each set is of size $\leq n/r$. Furthermore, for each set in the partition we can find a simplex that contains (at least) the points in the set. Lemma 6.2 shows that for $d = 2$, any line intersects roughly $O(\sqrt{r})$ triangles containing these point sets. In three dimensions, the guarantee is that any plane intersects (roughly) $O(r^{2/3})$ simplices that contains these sets (Lemma 6.3). Surprisingly, in the three dimensional case, any line intersects (roughly) $O(r^{1/3})$ such simplices, and it is not known how to construct partitions in three dimensions that have this property in the general case (when using only planes—the polynomial method yields partitions that have this property).

6.2.2 Separability of a random point set

Let P be a set of n points picked uniformly at random from the unit square $[0,1]^2$. In Section 6.4 we study the random variable $S_n = \mathrm{sep}(P)$. An initial guess might be that $\mathbb{E}[S_n] = \Theta(\sqrt{n})$ since random points in the unit square may look like grid points. This is not the situation here— surprisingly, in Theorem 6.3 we show that $S_n = O(n^{2/3})$, and $S_n = \Omega(n^{2/3} \log\log n / \log n)$ (both of these bounds hold with high probability). For $d \geq 2$, we prove that $\mathbb{E}[S_{n,d}] = O(n^{2/(d+1)})$ and $S_{n,d} = \Omega(n^{2/(d+1)} \log\log n / \log n)$, with high probability, where the Ω and O notations hide constants that depend polynomially on d. See Corollary 6.3.

What is going on? Consider the closest pair of points in P—the distance between this pair of points is in expectation roughly $1/n$. Indeed, there are $\binom{n}{2}$ pairs of points, and the probability of a specific pair of them to be in distance $\leq 1/n$ from each other is π/n^2 (ignoring boring and minor boundary issues).[1] By linearity of expectation, the expected number of pairs to be in distance $\leq 1/n$ from each other is $\binom{n}{2}\pi/n^2 \geq 1$. Of course, the closest pair distance in the grid $\left\{(i/\sqrt{n}, j/\sqrt{n}) \mid 1 \leq i,j \leq \sqrt{n}\right\}$ is $1/\sqrt{n}$. Thus there is a dichotomy between a random collection of points and points from a grid.

It turns out that the situation is similar in separating random points by lines—there are, in expectation, roughly $n^{2/3}$ pairs of points in P that are in distance $\leq 1/n^{2/3}$ from each other. Namely, there are many pairs of close points in P, and a line can separate only few of these pairs (this of course requires a proof). Thus, implying the lower bound. The upper bound follows readily by using a grid with cells of side length $1/n^{2/3}$, and then separating every bad (i.e., short) pair on its own.

[1] To see this, fix a point $p \in P$. The probability that another point $q \in P$ is at distance $\leq 1/n$ from p is equal to the probability that q falls in the disk d of radius $1/n$ centered in p. It follows that the event occurs with probability equal to $\mathsf{area}(\mathsf{d}) = \pi/n^2$.

What is not going on It is natural to think that maybe there is a convex subset of P of size $\Theta(n^{2/3})$. Since separating k points in convex position requires $\lceil k/2 \rceil$ lines, this would readily imply the lower bound. However it is known that for n random points, the size of the largest subset of points in convex position is $\Theta(n^{1/3})$ with high probability [8].

Similarly, one might try to blame the number of convex layers, which is indeed $\Theta(n^{2/3})$ for random points [53]. The similarity in the bounds seems to be a coincidence, since it is easy to construct examples of n points with $\Omega(n)$ convex layers that can be separated with $O(\sqrt{n})$ lines. See Figure 6.1.

6.2.3 A fast algorithm for approximating the separability

For a given set P of n points in the plane, in Section 6.5 we present an output-sensitive reweighting algorithm for approximating the separability, with running time that depends on the size of the optimal solution. The improved running time follows by implicitly storing the set of $\approx n^2$ candidate separating lines the solution can use. This requires using duality (see Definition 5.2$_{\text{p65}}$ and [78, Chapter 25]) and range searching data structures to implicitly maintain the set of separating lines (and their weights). For a given set of n points in the plane, the resulting algorithm computes a separating set of size $O(\sigma \log \sigma)$, in time $O\left(n^{2/3}\sigma^{5/3}\log^{O(1)} n\right)$, where σ is the separability of the given point set, see Theorem 6.4. Even in the worst case scenario when $\sigma = \Theta(n)$, the running time is $\widetilde{O}(n^{7/3})$ (where $\widetilde{O}$ hides polylogarithmic factors) which is a significant speedup over the "naive" $\widetilde{O}(n^3)$ time algorithm.

6.3 PROBLEM DEFINITION AND AN APPLICATION

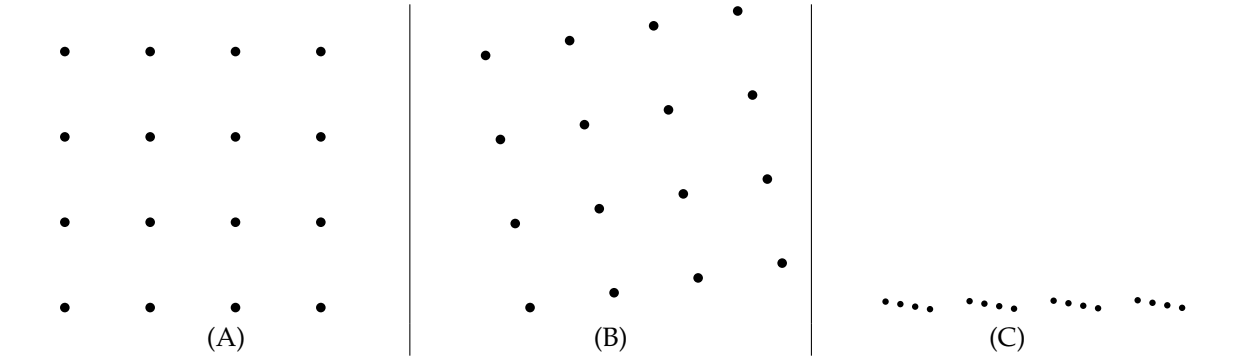
Definition 6.1. A set of n points P in the plane is in ***general position*** if no three of them are on a common line.

Definition 6.2. A set of lines L ***separates*** a set of points P, if for every pair $p, q \in P$, p and q are on different sides of some line $\ell \in L$.

Definition 6.3. For a set P of n points in the plane, its ***separability***, denoted by $\mathrm{sep}(P)$, is the size of the smallest set of lines that separates P.

Remark 6.1. (i) Assuming no three points are colinear, one might relax the definition, and allow points to be on the separating lines. Given such a separating set of lines L of size m, one can generate a set of lines of size at most $3m$ that properly separates all the pairs of points. Indeed, for each line ℓ, replace it by two lines that are parallel copies close to it. In addition, add an arbitrary line that properly separates the at most two points that might be on ℓ (by the general position assumption, no line can contain three points of P). (ii) For a point $p \in P$ and a separating set of lines L, there is a unique facet of the arrangement $\mathcal{A}(L)$ that the only point of P it contains is p. Since an arrangement of m hyperplanes in $\mathbb{R}^d$ has $O(m^d)$ faces of all dimensions[2], it follows

[2]The constant depends polynomially on d.



(D)

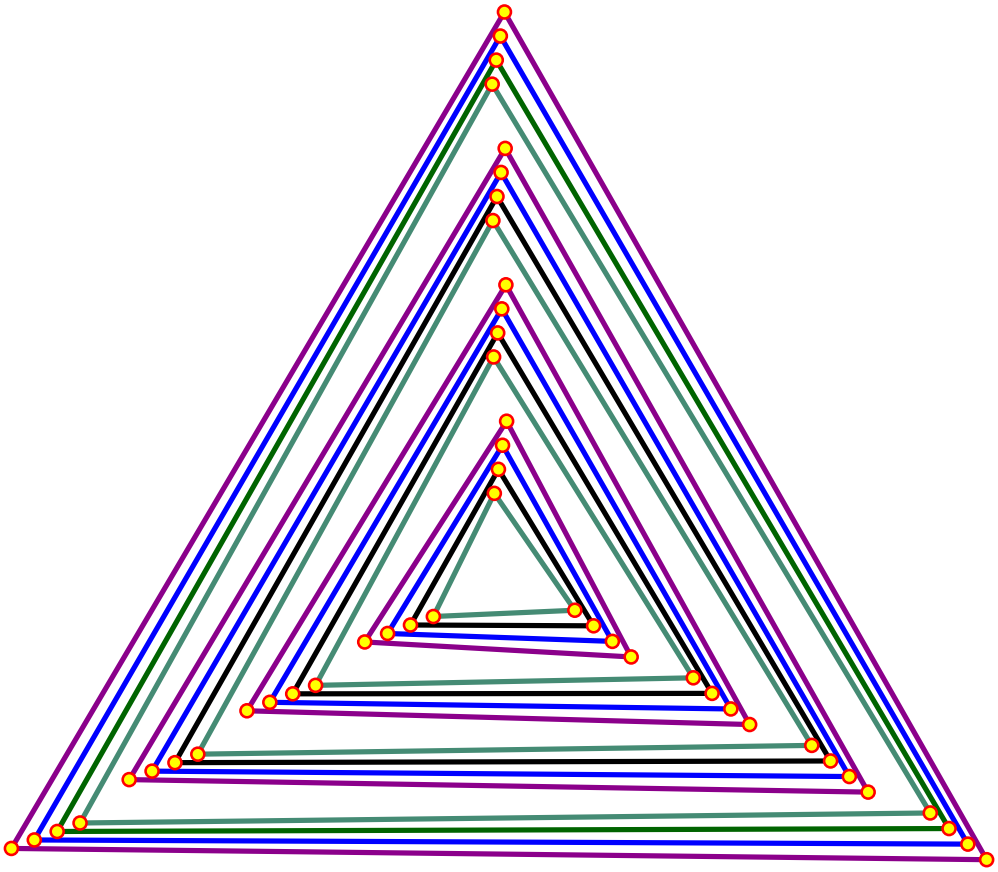


Figure 6.1: An example of a point set with $3n$ points and n convex layers, that can be separated with $6\sqrt{n}$ lines: (A) Start with a $\sqrt{n} \times \sqrt{n}$ grid. (B) Rotate the grid slightly. (C) Scale it linearly down in the y direction, so that the resulting point set almost lies on a line. Note that the resulting point set can still be separated by $2\sqrt{n}$ lines. (D) Take three rotated copies of the sets from (C), placing them along three rays emanating from a common point, with 120° degrees between the rays. Clearly, the resulting point set has $3n$ points, it can be separated using $6\sqrt{n}$ lines, and has n convex layers.

that $\text{sep}(P) = \Omega(n^{1/d})$. (iii) For the grid point set $P \equiv n^{1/d} \times \cdots \times n^{1/d}$ we have that the separability is $\leq dn^{1/d}$—indeed, use the natural axis-parallel hyperplanes separating layers of the grid. (iv) Consider a set P of n points spread on a strictly convex curve γ in $\mathbb{R}^d$ (i.e., γ is a convex curve that lies in some two dimensional plane). Any hyperplane, that does not contain γ, intersects γ in two points. It follows, that to separate the n points, we need $n-1$ break points along the curve. Hence, $\text{sep}(P) \geq (n-1)/2$ in this case.

An upper bound The following is an easy consequence of the results of Steiger and Zhao [147] (and is probably implied by earlier work).

Corollary 6.1. *Let Q, R be two point sets in the plane that are separated by a line, and furthermore, there are no three colinear points in $Q \cup R$. Then, for any choice of integers x, y, such that $1 \leq x < |Q|$, $1 \leq y < |R|$, there exists a line ℓ such that:*

(a) ℓ does not contain any point of $Q \cup R$,

(b) ℓ splits Q into two sets of size x and $|Q| - x$, respectively, and

(c) ℓ splits R into two sets of size y and $|R| - y$, respectively.

Lemma 6.1. *Let P be a set of points in $\mathbb{R}^d$ so that no three of them are on a common line. Then, $\text{sep}(P) \leq \lceil n/2 \rceil$.*

Proof: If $d > 2$, we project P into a randomly rotated two dimensional plane. Almost surely no three points in the projected point sets are colinear. In particular, a partition of the projected points by m lines can be lifted back, in the natural way, to a set of m hyperplanes separating the point set. Thus, from this point on, we assume the points of P are in the plane.

The splitting algorithm works as follows. Split P into two sets P_L and P_R of sizes $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$, respectively, by a vertical line. In the ith iteration of the algorithm, if $|P_R| \geq 3$, then by Corollary 6.1, there exists a line ℓ_i that splits P_L and P_R each into two sets, such that P_R (resp. P_L) gets split into one set with two points, and another set with $|P_R| - 2$ (resp. $|P_L| - 2$) points. We remove these four points from P_R and P_L, and split these two pairs of points by another line ℓ_i'.

Note, that this algorithm preserves the invariant that $|P_L| \geq |P_R|$ (and these sizes differ by at most one). If after the last iteration we are left with P_L ad P_R having sizes 3 and 2 respectively, then we split the set with three elements into a set with 2 and a single element, and then split the two pairs by a single line. The case that P_L ad P_R are both size 2 can be handled by a single splitting line, as is the case that P_L has two points, and P_R is a singleton.

The number of cutting lines used is $\lceil n/2 \rceil$ as an easy case analysis based on the value of $n \bmod 4$ shows. QED.

6.3.1 Application: Partition via separability in two and three dimensions

Definition 6.4. For a set P of n points in $\mathbb{R}^d$, and a parameter $r > 0$, an ***r-partition*** [110] is a partition of P into $t = O(r)$ disjoint sets $P_1, \ldots, P_t$, with associated simplices $\Delta_1, \ldots, \Delta_t$, such that:

(i) $\forall i$: $P_i \subseteq \Delta_i$,

(ii) $\forall i$: $|P_i| \leq n/r$, and

(iii) any hyperplane h intersects at most $f(r) = O(r^{1-1/d})$ simplices of $\Delta_1, \ldots, \Delta_t$.

It is not hard to see that such a partition exists for the grid point set. It is quite surprising that such a partition exists in the general case. The construction is due to Matoušek [110], and it is somewhat involved. Here, we show that if a point set has low separability, then one can easily construct a partition.

Lemma 6.2. *Let P be a set of n points in the plane with $\mathrm{sep}(P) = O(\sqrt{n})$ and $r > 0$ an integer parameter. One can compute a triangulation of the plane with $O(r \log^2 r)$ triangles, such that each triangle contains $\leq n/r$ points of P, and any line intersects at most $O(\sqrt{r} \log^2 r)$ triangles.*

Proof: Let L be a set of lines that separates P and realizes $\mathrm{sep}(P)$. Let $m = \mathrm{sep}(P) = |L|$. Consider a random sample S of size $O(\rho \log \rho)$ from L, where $\rho = \alpha\sqrt{r}$ and α is a sufficiently large constant.

Consider a face f of $\mathcal{A}(S)$—it is a convex polygon with $\rho' = O(\rho \log \rho)$ sides. We triangulate it by connecting consecutive even vertices (i.e., every other vertex as we travel along the boundary of f), and repeat this process until the face is fully triangulated. It is easy to verify that any line can intersect at most $O(\log \rho') = O(\log \rho)$ triangles in this triangulation of the face. Repeating this triangulation for all faces of $\mathcal{A}(L)$ results in a triangulation of the plane. Let T be the resulting set of triangles. Clearly, any line intersects at most $O(\rho \log^2 \rho)$ triangles of T.

The ε-net theorem (Theorem 4.1$_{\text{p44}}$, [88]) states that for a sample $S \subseteq L$ of size $O(\rho \log \rho)$, any triangle $\triangle$ that intersects more than m/ρ lines of L must also intersect at least one line of S. Furthermore this property holds for all ranges with at least some constant probability. Since all triangles of T were ultimately constructed from the set S, it follows from the ε-net theorem that any triangle $\triangle$ of T intersects at most m/ρ lines of L in its interior. By assumption, there is a constant c'' such that $m \leq c''\sqrt{n}$. Therefore the arrangement of L restricted to $\triangle$ can have at most $c'(m/\rho)^2 \leq c'\left(c''\sqrt{n}/\alpha\sqrt{r}\right)^2 \leq n/r$ faces (including edges on the boundary of $\triangle$), for some constant c', and for a sufficiently large constant α. This also bounds the number of points of P in $\triangle$, thus establishing the claim. QED.

Lemma 6.3. *Let P be a set of n points in $\mathbb{R}^3$ with $\mathrm{sep}(P) = O(n^{1/3})$ and $r > 0$ an integer parameter. One can compute a triangulation with $O(r \log^2 r)$ simplices, such that each simplex contains $\leq n/r$ points of P, any plane intersects at most $O(r^{2/3} \log^2 r)$ simplices, and any line intersects at most $O(r^{1/3} \log^2 r)$ simplices.*

Proof: We follow the proof of Lemma 6.2. Let L be a set of planes that separates P of size $m = O(n^{1/3})$. Let S be a random sample from L of size $O(\rho \log \rho)$, where $\rho = \alpha r^{1/3}$, where α is a sufficiently large constant. For a face f of $\mathcal{A}(S)$, which is a convex polytope (or convex polyhedra, if it is unbounded), we decompose it into simplices using the Dobkin-Kirkpatrick hierarchy [55]. If the face has t vertices, the resulting decomposition has $O(t)$ simplices, and furthermore, any line intersects at most $O(\log t)$ such simplices. Let T be the resulting set of simplices when applying this decomposition for all the faces of $\mathcal{A}(S)$.

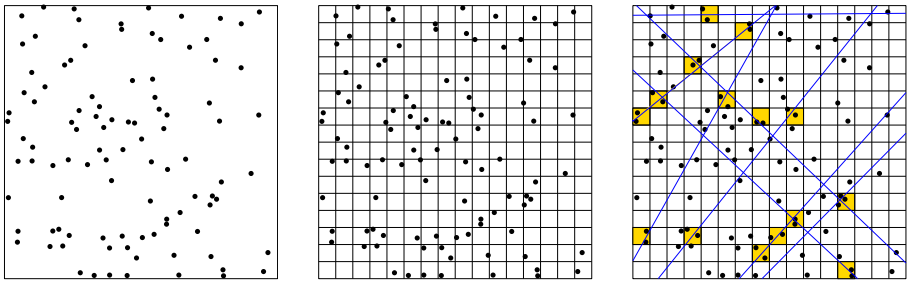


Figure 6.2: An illustration of the proof of Lemma 6.5.

As before, by the ε-net theorem (Theorem 4.1$_{\text{p44}}$), a simplex $\triangle \in T$ intersects at most m/ρ planes of L. For this reason, the arrangement of $\mathcal{A}(L)$ when restricted to $\triangle$, can have at most $c((m/\rho)^3) \leq n/r$ facets, which in turn bounds the number of points of P inside such a simplex by n/r.

Any line intersects $|S| - 1$ faces of S, and as such at most $O(|S| \log \rho) = O(r^{1/3} \log^2 r)$ simplices of T. For any plane h, the total number of vertices that belong to faces of $\mathcal{A}(S)$ that intersects h is $O(|S|^2)$ by the zone theorem [144]. Since a face is decomposed into a number of simplices that is proportional to its complexity, it follows that h intersects at most $O(r^{2/3} \log^2 r)$ simplices. QED.

6.4 SEPARATING RANDOM POINTS BY LINES

Here we consider the separability of a set P of n points picked uniformly, randomly and independently in the unit square, and the random variable $S_n = \text{sep}(P)$, which is the separability of P.

6.4.1 The upper bound

Let G be the uniform grid that partitions the unit square into $N \times N$ cells, where $N = n^{2/3}$. This grid is defined by $2(N-1)$ lines, and the area of each grid cell is $p = 1/N^2 = (1/n^{2/3})^2 = 1/n^{4/3}$. A ***grid collision*** is when two points $p, q \in P$ belong to the same cell of G, and in such a case p and q ***collide***.

Lemma 6.4. *Let Z be the number of pairs of points of P that collide in the grid G (i.e., Z is a random variable). Then for n sufficiently large, we have $n^{2/3}/3 \leq \mathbb{E}[Z] \leq n^{2/3}/2$.*

Proof: Let $P = \{p_1, \ldots, p_n\}$, where the exact location of each point in this set is yet to be determined. The probability for two points p_i and p_j to collide, that is to fall into the same cell in the grid, is $p = 1/N^2$—indeed, first throw in the point p_i, and the desired probability is the probability of p_j to fall into the cell that contains x_i. By linearity of expectation, the expected number of colliding pairs is $\mathbb{E}[Z] = \binom{n}{2} p \leq n^2/(2n^{4/3}) = n^{2/3}/2$.

For the lower bound, observe that $\mathbb{E}[Z] = \binom{n}{2} p = \dfrac{n(n-1)}{2N^2} \geq \dfrac{n^2}{3n^{4/3}} = \dfrac{n^{2/3}}{3}$, for $n \geq 3$. QED.

Lemma 6.5. $\mathbb{E}[S_n] = O(n^{2/3})$.

Proof: Let L be the set of $2(n^{2/3}-1)$ separating lines used in creating G. By Lemma 6.4, the expected number of pairs of points of P colliding is $O(n^{2/3})$. For each such colliding pair, we add to L a line that separates this pair (thus $|L|$ is a random variable). At the end of this process all points of P are separated, see Figure 6.2. As claimed, we have $\mathbb{E}[S_n] \leq \mathbb{E}[|L|] = O(n^{2/3} + \mathbb{E}[Z]) = O(n^{2/3})$. QED.

6.4.2 A detour to balls into bins

The problem at hand is related to the problem of balls and bins. Here, given n balls, one throws them into m bins, where $\mathsf{m} \geq \mathsf{n}$.

A ball that falls into a bin with i or more balls is ***i*-heavy**. Let $B_{\geq i}$ be the number of i-heavy balls. It turns out that a strong concentration on $B_{\geq i}$ follows readily from Talagrand's inequality. While this is probably already known, we were unable to find it in the literature, and we provide a self contained proof here for the sake of completeness.

The expectation of $B_{\geq i}$

Lemma 6.6. *For $i > 1$, consider throwing n balls into m bins, where $\mathsf{m} \geq 3\mathsf{n}$. Then,*

$$e^{-2}F_i \leq \mathbb{E}[B_{\geq i}] \leq 6e^{i-1}F_i, \qquad \text{where} \qquad F_i = \mathsf{n}\Big(\frac{\mathsf{n}}{i\mathsf{m}}\Big)^{i-1},$$

and $B_{\geq i}$ is the number of i-heavy balls. In addition, the expected number of pairs of i-heavy balls that are colliding, denoted by β_i, is $\beta_i \leq c_i \mathsf{n}(\mathsf{n}/\mathsf{m})^{i-1}$, where $c_i = O(i(e/i)^{i-1})$.

Proof: Let $p = 1/\mathsf{m}$. A specific ball falls into a bin with exactly i balls if there are $i-1$ balls (of the remaining $n-1$ balls) that fall into the same bin. Hence the probability for that is $\gamma_i = p^{i-1}(1-p)^{\mathsf{n}-i}\binom{\mathsf{n}-1}{i-1}$. We have that a specific ball is i-heavy with probability

$$\begin{aligned}
\alpha = \sum_{j=i}^{\mathsf{n}} \gamma_j &= \sum_{j=i-1}^{\mathsf{n}-1} \binom{\mathsf{n}-1}{j} p^j (1-p)^{\mathsf{n}-j-1} \leq \sum_{j=i-1}^{\mathsf{n}-1} \Big(\frac{e(\mathsf{n}-1)}{j\mathsf{m}}\Big)^j \leq (\mathsf{n}/\mathsf{m})^{i-1} \sum_{j=i-1}^{\mathsf{n}-1} \Big(\frac{e}{j}\Big)^j \\
&\leq (\mathsf{n}/\mathsf{m})^{i-1} 2.1 \Big(\frac{e}{i-1}\Big)^{i-1} \leq (\mathsf{n}/\mathsf{m})^{i-1} 2.1e \Big(\frac{e}{i}\Big)^{i-1} \leq 6e^{i-1}\Big(\frac{\mathsf{n}}{i\mathsf{m}}\Big)^{i-1},
\end{aligned}$$

as $(n/i)^i \leq \binom{n}{i} \leq (en/i)^i$, $i > 1$, and the summation behaves like a geometric series dominated by its first term. Since $(1-p)^{\mathsf{n}-j-1} \geq (1-1/\mathsf{m})^{\mathsf{m}-1} \geq 1/e$, we have

$$\alpha \geq \frac{1}{e} \sum_{j=i-1}^{\mathsf{n}-1} \Big(\frac{\mathsf{n}-1}{j\mathsf{m}}\Big)^j \geq \frac{1}{e}\Big(\frac{\mathsf{n}-1}{\mathsf{n}} \cdot \frac{\mathsf{n}}{(i-1)\mathsf{m}}\Big)^{i-1} \geq \frac{1}{e^2}\Big(\frac{\mathsf{n}}{i\mathsf{m}}\Big)^{i-1}.$$

We conclude that $\mathbb{E}[B_{\geq i}] = \mathsf{n}\alpha = \Theta(\mathsf{n}(\mathsf{n}/\mathsf{m})^{i-1})$.

If a ball is in a bin with exactly j balls, for $j \geq i$, then it collides directly with $j-1$ other i-heavy balls. Thus, the expected number of collisions that a specific ball has with i-heavy balls is in expectation $\sum_{j=i}^{\mathsf{n}}(j-1)\gamma_j = \sum_{j=i-1}^{\mathsf{n}-1} j\gamma_{j+1}$. Summing over all balls, and arguing as above, we have that the expected overall number of such collisions is

$$\beta_i \leq \mathsf{n} \sum_{j=i-1}^{\mathsf{n}-1} j\gamma_{j+1} = \mathsf{n} \sum_{j=i-1}^{\mathsf{n}} j \binom{\mathsf{n}-1}{j} p^j (1-p)^{\mathsf{n}-j-1} = O\Big(\mathsf{n} i \Big(\frac{e\mathsf{n}}{i\mathsf{m}}\Big)^{i-1}\Big)$$

(note, that this counts every collision twice). QED.

Concentration of $B_{\geq i}$ Let $f(\mathsf{x})$ be a real-valued function over some product probability space $\Omega = \Omega_1 \times \cdots \times \Omega_n$. The function f is ***r-certifiable***, if for every $\mathsf{x} \in \Omega$, there exists a set of indices $J(\mathsf{x}) \subseteq \{1, \ldots, n\}$, such that:

(A) $|J(\mathsf{x})| \leq r f(\mathsf{x})$, and

(B) if $\mathsf{y} \in \Omega$ agrees with x on the coordinates in $J(\mathsf{x})$, then $f(\mathsf{y}) \geq f(\mathsf{x})$.

The function f is ***c-Lipschitz*** if for two values $\mathsf{x}, \mathsf{y} \in \Omega$ that agree on all coordinates except one, we have that $|f(\mathsf{x}) - f(\mathsf{y})| \leq c$. For a real valued random variable f, its ***median***, denoted by $\nu(f)$, is the infimum value ν, such that $\mathbf{Pr}[f < \nu] \leq 1/2$ and $\mathbf{Pr}[f > \nu] \leq 1/2$.

The version of Talagrand's inequality we need is the following.

Theorem 6.1 ([56, Theorem 11.3]). *Let $f : \Omega \to \mathbb{R}$ be a c-Lipschitz and r-certifiable function, for some constants r and c, with its median being $\nu = \nu(f)$. Then, for all $t > 0$, we have $\mathbf{Pr}\big[|f - \nu| > t\big] \leq 4\exp\Big(\frac{-t^2}{4c^2 r(\nu+t)}\Big)$.*

Lemma 6.7. *Consider throwing n balls into m bins, where $\mathsf{m} \geq 3\mathsf{n}$. Furthermore, let i be a small constant integer, $B_{\geq i}$ be the number of balls that are contained in bins with i or more balls, and let $\nu_i = \nu(B_{\geq i})$. In addition, assume that $\nu_i \geq 16 i^2 c \ln \mathsf{n}$, where c is some arbitrary constant. Then $\mathbf{Pr}\big[|B_{\geq i} - \nu_i| \geq 4i\sqrt{c\nu_i \ln \mathsf{n}}\big] \leq 1/\mathsf{n}^c$. Furthermore, for some constant c', we have $|\nu_i - \mathbb{E}[B_{\geq i}]| \leq c' i\sqrt{\nu_i}$, and thus*

$$\mathbf{Pr}\Big[\big|B_{\geq i} - \mathbb{E}[B_{\geq i}]\big| \geq c' i\sqrt{\nu_i} + 4i\sqrt{c\nu_i \ln \mathsf{n}}\Big] \leq 1/\mathsf{n}^c.$$

Proof: Observe that $B_{\geq i}$ is 1-certifiable—indeed, the certificate is the list of indices of all the balls that are contained in bins with i or more balls. The variable $B_{\geq i}$ is also i-Lipschitz. Changing the location of a single ball, can make one bin that contains i balls, into a bin that contains $i-1$ balls, thus decreasing $B_{\geq i}$ by i. Applying Theorem 6.1 (Talagrand's inequality), with $t = 4i\sqrt{c\nu_i \ln \mathsf{n}}$ we have

$$\mathbf{Pr}\Big[\big|B_{\geq i} - \nu_i\big| > t\Big] \leq 4\exp\left(-\frac{t^2}{4i^2(\nu_i + t)}\right) \leq \frac{1}{\mathsf{n}^c},$$

assuming $t \leq \nu_i$.

The estimate on the distance of ν_i and $\mathbb{E}[B_{\geq i}]$ follows by estimating the expectation, by breaking the real

line into intervals of length $O(i\sqrt{\nu_i})$, and using the exponential decay of the probability in each such interval as we get away from ν_i, as implied by the above. Formally, we have that

$$\begin{aligned}|\nu_i - \mathbb{E}[B_{\geq i}]| &= |\mathbb{E}[\nu_i - B_{\geq i}]| \leq \mathbb{E}[|\nu_i - B_{\geq i}|] \\ &\leq \sum_{k\geq 0} (k+1) i\sqrt{\nu_i} \cdot \mathbf{Pr}[|B_{\geq i} - \nu_i| > ki\sqrt{\nu_i}].\end{aligned}$$

Now applying Theorem 6.1 and using the assumption ν_i is sufficiently large,

$$\begin{aligned}|\nu_i - \mathbb{E}[B_{\geq i}]| &\leq \sum_{k\geq 0} (k+1) i\sqrt{\nu_i} \cdot \mathbf{Pr}[|B_{\geq i} - \nu_i| > ki\sqrt{\nu_i}] \\ &\leq \sum_{k\geq 0} (k+1) i\sqrt{\nu_i} \cdot 4\exp\left(-\frac{k^2 i^2 \nu_i}{4i^2(\nu_i + ki\sqrt{\nu_i})}\right) \\ &\leq \sum_{k\geq 0} (k+1) i\sqrt{\nu_i} \cdot 4\exp\Big(-k^2/4(1+k)\Big) = O(i\sqrt{\nu_i}).\end{aligned}$$

The final inequality is readily implied by combining the two earlier statements. QED.

In the following, we need Chernoff's inequality, which we state explicitly for completeness [56].

Theorem 6.2. *Let $X_1, \ldots, X_n$ be n independent random variables where $\mathbf{Pr}\big[X_i = 1\big] = p_i$, and $\mathbf{Pr}\big[X_i = 0\big] = 1 - p_i$. And let $X = \sum_{i=1}^{b} X_i$, and $\mu = \mathbb{E}\big[X\big] = \sum_i p_i$. For any $\delta \geq 0$, we have*

$$\mathbf{Pr}\Big[X > (1+\delta)\mu\Big] < \begin{cases} \exp\Big(-\mu\delta^2/4\Big) & 2e - 1 \geq \delta \geq 0 \\ 2^{-\mu(1+\delta)} & \delta \geq 2e - 1 \\ \exp\big(-(\mu\delta/2)\ln\delta\big) & \delta \geq e^2. \end{cases}$$

Similarly, we have $\mathbf{Pr}[X < (1-\delta)\mu] < \exp\Big(-\mu\delta^2/2\Big)$.

Not too many shared birthdays The *birthday paradox* states that if one throws n balls (i.e., birthday dates of n people) into $\mathsf{m} = \Theta(\mathsf{n}^2)$ bins (i.e., days of the year), then the number of bins containing two or more balls is non-zero with constant probability. The following proves that the number of such bins can not be too large.

Lemma 6.8. *Consider throwing n balls into $\mathsf{m} = c\mathsf{n}^2$ bins, where c is some constant. Then, with high probability, the total number of bins that contain two or more balls is $O(\log \mathsf{n} / \log\log \mathsf{n})$.*

Proof: Partition the set B of n balls into two sets X and Y, each of size $\mathsf{n}/2$. Let Y be the number of bins that contains balls of X—clearly, $Y \leq \mathsf{n}/2$. As such, the probability of a ball of Y to fall into a bin with a ball of X, is $\alpha = Y/\mathsf{m} \leq (\mathsf{n}/2)/\mathsf{m} = 1/(2c\mathsf{n})$. Now the expected number of bins that contains balls from both X and Y is $|Y|\,\alpha = (\mathsf{n}/2)\alpha \leq 1/4c$. By Chernoff's inequality, this quantity is smaller than $T = O(\log \mathsf{n} / \log\log \mathsf{n})$, with high probability[3].

[3] By Theorem 6.2 with $\mu = 1/4c$, and $\delta = c_1 \log \mathsf{n} / \log\log \mathsf{n}$, where c_1 is a sufficiently large constant.

This approach allows us to count the number of bins that contain balls from both X and Y. However, to count the number of bins that contain two or more balls, we need to count those bins which may only contain balls from X (or from Y). To overcome this, we repeat the above experiment, generating new partitions $(X_1, Y_1), \ldots, (X_M, Y_M)$, as above, such that any pair $p, q \in B$ appears in a constant fraction of these partitions on different sides. This is easy to do—match the balls of B in pairs. To generate the ith partition, the algorithms goes over the pairs in the matching (p, q), and puts p in X_i and q in Y_i with probability half, and otherwise it assigns p to Y_i and q to X_i. Observe that $|X_i| = |Y_i| = \mathsf{n}/2$.

Repeating this $M = c_2 \log \mathsf{n}$ times, guarantees with high probability, that any two balls $p, q \in B$ appear in opposing sides of at least one of these partitions (two points that are an edge in the matching are in different sides in all partitions). Furthermore, by Chernoff's inequality, each pair appears in at least $\gamma = \Omega(\log \mathsf{n})$ pairs, with high probability[4].

Consequently, there are at most $\beta = O(M \log \mathsf{n} / \log\log \mathsf{n})$ heavy bins with balls that belong to different sides of some partition. Each such heavy bin gets counted at least γ times, thus implying that the number of heavy bins is at most $\beta/\gamma = O(\log \mathsf{n} / \log\log \mathsf{n})$. QED.

The following is not required for the proof of the main result, and we include it since it might be of independent interest. Note that the next lemma bounds the number of balls colliding, while Lemma 6.8 bounded the number of bins.

Lemma 6.9. *Consider throwing* n *balls into* $\mathsf{m} = c\mathsf{n}^2$ *bins, where* c *is some constant. Then, with high probability, the total number of colliding pairs of balls is* $O(\log \mathsf{n} / \log\log \mathsf{n})$.

Proof: Let i be a sufficiently large constant (in particular $i \geq e/c$). By Lemma 6.6, the expected number of collisions of pairs of i-heavy balls is $\beta_i = O\big(\mathsf{n} i \big(\frac{e\mathsf{n}}{i\mathsf{m}}\big)^{i-1}\big) = O\big(\mathsf{n} i / \mathsf{n}^{i-1}\big) = O\big(1/\mathsf{n}^{i-3}\big)$. By Markov's inequality, the probability that there are any collisions involving i-heavy balls is at most β_i. As for collisions of pairs that are not i-heavy, by Lemma 6.8, with high probability there are at most $O(\log \mathsf{n} / \log\log \mathsf{n})$ bins that contain between 2 and $i-1$ balls, and each such bin contributes at most $\binom{i-1}{2}$ colliding pairs. We conclude, that with high probability, the total number of collisions is $O(i^2 \log \mathsf{n} / \log\log \mathsf{n}) = O(\log \mathsf{n} / \log\log \mathsf{n})$, as claimed. QED.

Remark 6.2. Somewhat disappointingly, the upper bound $O(\log \mathsf{n} / \log\log \mathsf{n})$ on the number of colliding balls, in Lemma 6.9, is tight if the probability of success is required to be $\geq 1 - 1/\mathsf{n}^{\tau}$, where τ is some constant. To see that, consider the partition (X, Y) from the proof of Lemma 6.8. With high probability, the number of bins containing balls of X is $\geq \mathsf{n}/4$—this follows by similar to, but easier, argument to the one used in the proof of Lemma 6.7. As such, the probability of a ball of Y to collide with a ball of X is at least $p = 1/(4c\mathsf{n})$. Thus, the probability that exactly i such collisions happen is at least $\zeta = \binom{\mathsf{n}/2}{i} p^i (1-p)^{\mathsf{n}/2-i} \geq$

[4] Observe that a pair p, q is separated by a partition with probability at least half. Let X be the number of partitions separating this pair. The expected number of partitions separating this pair is $\mu = \mathbb{E}[X] = M/2 = (c_2/2)\log \mathsf{n}$. Setting $\delta = 1/2$, we have by Theorem 6.2 that $\mathbf{Pr}[X \leq (1-\delta)\mu] \leq \exp(-\mu\delta^2/2) \leq 1/n^{\Omega(1)}$, by making c_2 sufficiently large. It follows that with high probability, the pair is separated in at least $\gamma = (1-\delta)\mu = (c_2/4)\log \mathsf{n}$ partitions.

$\left(\frac{\mathsf{n}/2}{i}\right)^i \left(\frac{1}{4c\mathsf{n}}\right)^i = \frac{1}{(8ci)^i}$. If we require the last quantity to be larger than $1/\mathsf{n}^\tau$, then we have $\mathsf{n}^\tau \geq (8ci)^i \iff \tau \ln \mathsf{n} \geq i \ln(8ci)$, which holds for $i = \Theta(\log \mathsf{n} / \log\log \mathsf{n})$, as τ and c are constants. Namely, for the specified value of i, we have that i collisions happen with probability $\geq \zeta \geq 1/n^\tau$.

How many collisions are there, anyway? It is useful to think about the point set P as being generated by throwing $\mathsf{n} = n$ balls into $\mathsf{m} = N^2 = n^{4/3}$ bins – here every grid cell is a bin. Lemma 6.6 and Lemma 6.7 together implies the following.

Corollary 6.2. *When throwing n balls into $n^{4/3}$ bins, we have, with high probability, that $B_{\geq 2} = \Theta(n^{2/3})$ and $B_{\geq 3} = \Theta(n^{1/3})$, where $B_{\geq i}$ is the number of balls that are in bins with i or more balls.*

Lemma 6.10. *Let P be a set of n random point picked uniformly in the unit square. Let Z be the number of active grid cells—namely, the number of grid cells that contains two or more points of P. We have, with high probability, that $Z = \Theta(n^{2/3})$.*

Proof: We interpret this as throwing n balls into $n^{4/3}$ bins (a grid cell is a bin). The number of balls colliding with exactly one other ball is $B_{\geq 2} - B_{\geq 3}$. Therefore the number of bins containing exactly two balls is $(B_{\geq 2} - B_{\geq 3})/2$. By Corollary 6.2, we have $Z \geq (B_{\geq 2} - B_{\geq 3})/2 = \Theta(n^{2/3})$. QED.

A single line can not be involved in too many active cells

Lemma 6.11. *Let S be a given set of $2N$ grid cells. A cell of S is **active** if it contains two or more points of P. Let Y be the number of cells of S that are active. We have that $Y = O(\log n / \log\log n)$, with high probability (i.e., $\geq 1 - 1/n^{\Omega(1)}$).*

Proof: For any i, let X_i be the indicator variable that is one if the ith point of P falls into a cell of S, and let $Y = \sum_{i=1}^n X_i$. The probability of a point $p \in P$ to fall into a cell of S is at most $p' = p2N = 2/N$. Hence $\mu = \mathbb{E}[Y] = np' \leq 2n/N = 2n^{1/3}$. By Chernoff's inequality (Theorem 6.2), we have that

$$\mathbf{Pr}\left[Y \geq 3n^{1/3}\right] \leq \mathbf{Pr}[Y \geq (1+1/2)\mu] \leq \exp\left(-\frac{\mu(1/2)^2}{4}\right) \leq \exp\left(-\frac{n^{1/3}}{8}\right).$$

From this point on, we assume that $Y \leq 3n^{1/3}$. Thus, we are throwing at most $3n^{1/3}$ balls into $2N = 2n^{2/3}$ bins. By Lemma 6.8, with high probability, there are at most $O(\log n / \log\log n)$ bins with two or more balls. QED.

The result

Theorem 6.3. *Let P be a set of n points picked uniformly and randomly from the unit square. Let S_n be the separability of P— the minimum number of lines separating P. Then, with high probability we have $S_n = \Omega(n^{2/3} \log\log n / \log n)$ and $S_n = O(n^{2/3})$.*

Proof: We remind the reader that G is the grid partitioning the unit square into $N \times N$ cells, where $N = n^{2/3}$. For a line ℓ that avoids the vertices of G, consider the set of grid cells that it intersects, formally $B(\ell) = \{\Delta \mid \Delta \in G \text{ and } \Delta \cap \ell \neq \emptyset\}$. Since ℓ intersects $\leq N-1$ horizontal and $\leq N-1$ vertical lines of the grid inside the unit square, it follows that $|B(\ell)| \leq 2N-1$. Fix an arbitrary ordering of the cells of G, and add cells according to this ordering to $B(\ell)$ until this set is of size $2N$. The resulting set, $\mathrm{sgn}(\ell)$ is the ***signature*** of ℓ.

Let $\mathcal{L}$ be a set of representative lines. Specifically, among all lines with the same signature, pick one of them to be in $\mathcal{L}$. It is easy to verify that $|\mathcal{L}| = O(N^4) = O(n^{8/3})$.

We are now ready for the proof itself. Consider the randomly generated point set P. We consider two points to be separated if they belong to different grid cells. It remains to separate points that collide in the grid (i.e., belong to the same grid cell). So consider a minimal separating set of lines L. A line in L intersects $\leq 2N$ cells of the grid, and by Lemma 6.11, with high probability, its signature contains at most $T = O(\log n / \log\log n)$ active grid cells. Namely, each such line can at best only separate pairs that belong to these active cells.

However, Lemma 6.10 implies that, with high probability, the number of active grid cells is at least $n^{2/3}/c'$, where c' is some constant. It follows that any set of lines that separates all the pairs of points that collide, must be of size $\geq (n^{2/3}/c')/T$, with high probability.

As for the upper bound, it follows from the argument of Lemma 6.5. The number of points that need separation from other points (after we use the grid lines) is $B_{\geq 2}$. However, $B_{\geq 2} = \Theta(n^{2/3})$ with high probability, by Corollary 6.2. Now use three lines to separate each point from all the other points of P. This implies the upper bound. QED.

6.4.3 Extensions

Other domains Both the lower bound and upper bound for S_n in Theorem 6.3 hold if n points are sampled uniformly at random within any given convex region C. Indeed, John's ellipsoid theorem [78] implies that after appropriate scaling, there is a disk D such that $D \subseteq C \subseteq 2D$. In particular, we can inscribe in D a large square S, see Figure 6.3. This square contains a constant fraction of the area of C, since

$$\mathsf{area}(S) = (2/\pi) \cdot \mathsf{area}(D) = (1/2\pi) \cdot \mathsf{area}(2D) \geq (1/8) \cdot \mathsf{area}(C).$$

By sampling n points uniformly at random from C, a Chernoff bound (Theorem 6.2) implies that with high probability, $\Theta(n)$ points will fall inside S. This implies that we can focus on separating pairs of points which fall inside the square S. In particular, the number of lines needed to separate all pairs of points inside C is at least the number of lines needed to separate all pairs of points inside S. The lower bound of Theorem 6.3 follows.

As for the upper bound on S_n, the argument is similar. Given the disk $2D \supseteq C$, whose area is a constant

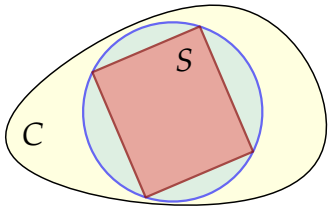


Figure 6.3: Inscribing a large square inside a convex body.

factor larger than the area of C, we can circumscribe a square S' around the disk $2D$. An identical argument as above implies that the area of S' is at most a constant factor larger than the area of C. Thus we can consider sampling n points from this square S', where $\Theta(n)$ of them land in C with high probability. It follows that the number of lines needs to separate pairs of points inside C is at most the number lines needs to separate pairs of points in S'.

Higher dimensions

Corollary 6.3. *Let P be a set of n points picked uniformly and randomly from the unit cube $[0,1]^d$. With high probability, the minimum number of hyperplanes separating P is $\Omega_d(n^{2/(d+1)} \log\log n / \log n)$. Similarly, in expectation, one can separate P using $O(dn^{2/(d+1)})$ hyperplanes.*

Proof: One can easily extend the two dimensional analysis to higher dimensions. We quickly sketch the calculations without going into the low level details, which follows readily by retracing the same argumentation.

In the following $f \approx g$, means that $f = \widetilde{\Theta}(g)$. We now consider the unit cube $[0,1]^d$. As before, we partition it into N^d grid cells, in the natural way, where the value of N is to be determined shortly. Let G denote the resulting grid. Any hyperplane intersects at most $H \approx N^{d-1}$ grid cells. We would like to guarantee that that there are $\approx O(1)$ cells that contain two and more points, for a fixed hyperplane h. By the birthday paradox, this means that we should have at most $\approx \sqrt{H}$ random points falling into the H cells associated with h, if we want a constant number of collisions. Since the probability of a point to fall into a grid cell that h intersects is H/N^d, we get that

$$\sqrt{H} \approx nH/N^d \implies H \approx (N^d/n)^2 \implies N^{d-1} \approx N^{2d}/n^2 \implies n^2 \approx N^{d+1} \implies N = n^{2/(d+1)}.$$

The overall number of grid cells that contain two or more points is

$$\binom{n}{2}/N^d \approx n^2/N^d = n^{2-2d/(d+1)} = n^{2/(d+1)}.$$

Finally, with high probability, a hyperplane can intersects only $\approx O(1)$ active grid cells, which means that the number of hyperplanes needed to separate n random points is $\approx n^{2/(d+1)}/O(1)$.

The lower bound follows by plugging in the above sketch, into the detailed analysis of the two dimensional case.

For the upper bound: In the grid G, the volume of each grid cell is $p = 1/N^d = 1/n^{2d/(d+1)}$. Thus the expected number of collisions happening inside the grid cells is $\mathbb{E}[Z] = \binom{n}{2}p \leq n^2/2n^{2d/(d+1)} = O(n^{2/(d+1)})$. We separate each such colliding pair by its own hyperplane. Note, that creating the grid G, requires $d(N-1)$ separating hyperplanes. As such, the expected number of separating hyperplanes one needs is at most $O(dN + \mathbb{E}[Z]) = O(dn^{2/(d+1)})$. QED.

Remark 6.3. A set of n points of the grid $n^{1/d} \times \cdots \times n^{1/d}$ in $\mathbb{R}^d$ requires $\leq dn^{1/d}$ hyperplanes to separate them. Therefore the gap demonstrated in two dimensions (between the grid and random points) also holds in higher dimensions.

Allowing more points to collide Here, we change the problem—we allow groups of up to t points to not be separated by the points.

Lemma 6.12. *Let $t > 1$ be a fixed integer constant. Consider a set P of n random points chosen uniformly and independently from $[0,1]^2$. In expectation there is a set L of $O(\mathsf{n}^{(t+1)/(2t+1)})$ lines, such that every face of $\mathcal{A}(L)$ contains at most t points of L.*

Proof: Let $N = \mathsf{n}^{(t+1)/(2t+1)}$ and consider the set of lines forming the $N \times N$ grid. Let $\mathsf{m} = N^2$. Consider the distribution of the points of P in the grid cells. Any grid cell that contains more than t points, is further split by introducing additional lines until every cell in the resulting arrangement contains at most t points.

To bound the number of these additional fix-up lines, recall the balls and bins interpretation. By Lemma 6.6, the number of points that falls into grid cells with $t+1$ or more balls is

$$\Theta\left(\mathsf{n}^{t+1}/\mathsf{m}^t\right) = \Theta\left(\mathsf{n}^{t+1}/\mathsf{n}^{2t(t+1)/(2t+1)}\right) = \Theta\left(\left(\mathsf{n}^{1-2t/(2t+1)}\right)^{t+1}\right) = O(\mathsf{n}^{(t+1)/(2t+1)}).$$

Clearly, this also provides an upper bound on the number of fix-up lines needed. QED.

6.5 APPROXIMATING A MINIMUM SEPARATING SET OF LINES

6.5.1 Problem statement and a slow algorithm

Given a set P of n points in general position (i.e., no three points are colinear) in the plane, our goal is to approximate the minimal set of lines L separating all the pairs of points of P.

Reduction to hitting set Given a set P as above, one can restate the problem as a hitting set problem. Indeed, let $\mathcal{C} = \{\text{line}(p,q) \mid p,q \in P\}$ be the set of candidates, which contains all lines that pass through every pair of points of P, where $\text{line}(p,q)$ denotes the line passing through p and q.

To simplify the description of the algorithm, we use an alternative definition of separation as suggested in Remark 6.1 (A). Specifically, we modify Definition 6.2 by allowing points to lie on the separating lines. In particular, two points p and q are additionally considered to be separated if the line separating them passes through p or q.

For each pair of points $p, q \in P$, consider the set of all lines of $\mathcal{C}$ that intersect this segment pq:

$$L_{pq} = \{\ell \in \mathcal{C} \mid pq \cap \ell \neq \varnothing\}.$$

By our modified definition, any of the lines of L_{pq} can be interpreted as separating the two points p and q. Consider the set system

$$\mathsf{S} = (\mathcal{C}, \mathcal{R}), \quad \text{where } \mathcal{R} = \big\{L_{pq} \mid p, q \in P, p \neq q\big\}. \tag{6.1}$$

We refer the reader to Section 4.2.1$_{\text{p43}}$ for the relevant definitions on set systems and VC dimension.

Observation 6.1. *Given a set L' of m lines that separates P, there exists a subset $L \subseteq \mathcal{C}$ of m lines, such that L separates P. Indeed, translate and rotate every line of L' till it passes through two points of P. Clearly, the resulting set of lines separates the points of P.*

Lemma 6.13. *The set system S defined by Eq. (6.1) has VC dimension at most 11.*

Proof: The following argument is due to Kynčl [102]. The arrangement of m lines in the plane has at most $f = m(m+1)/2 + 1$ faces. As such, there are at most $\binom{f}{2}$ distinct segments (as far as what lines they intersect). If a set L of m lines is shattered by the range space, then we must have $2^m \leq \binom{f}{2} \leq (m(m+1)/2+1)^2$, and this inequality breaks for $m = 12$, which implies that the VC dimension is at most 11. QED.

A further improvement on the bound of the above Lemma might be possible by more involved argument [102], but one has to be careful since the lines of L are not in general position.

It follows that one can compute a separating set by computing (approximately) a hitting set for the set system S, using known approximation algorithms for hitting sets for spaces with bounded VC dimension [78].

The basic approximation algorithm for hitting set for S We next describe the standard reweighting algorithm for hitting set in our context. Such reweighting algorithms go back to the work by Chazelle and Welzl [43]. In the context of geometric set-cover/hitting-set, Clarkson [46] was the first to suggest such an algorithm. Clarkson's algorithm was further formalized by Brönnimann and Goodrich [26]. See [78, Chapter 6] for more details.

The algorithm Given S as above, let L_{opt} be the optimal solution, and let $\sigma = \text{sep}(P) = \left|L_{\text{opt}}\right|$ denote the separability of P. The algorithm will perform an exponential search for the value of σ. Let k be the current guess for the value of σ (at the beginning of the algorithm $k = 1$).

Initially, each line in $\ell \in \mathcal{C}$ is assigned weight $\omega(\ell) = 1$. For a subset $L \subseteq \mathcal{C}$, its *weight* is $\omega(L) = \sum_{\ell \in L} \omega(\ell)$. In each iteration, the algorithm samples a set of lines $S \subseteq \mathcal{C}$ of size $O(\varepsilon^{-1} \log \varepsilon^{-1})$ (where $\varepsilon = 1/4k$ and k is the current guess for σ) picked according to their weights. By the ε-net theorem (Theorem 4.1$_{\text{p44}}$), S is an ε-net with probability at least $1 - \varepsilon^c = 1 - 1/(4k)^c$ (for some sufficiently large constant c). The algorithm next checks if the sample S separates P, and if so, it returns the sample as the desired separating set.

To this end, the algorithm builds the arrangement $\mathcal{A}(S)$ and preprocesses it for point-location queries. Next, it locates all of the faces in this arrangement that contain points of P. If there is a pair of points $p, q \in P$ that are in the same face, then this pair is not separated by S. If the weight of the lines L_{pq} is at most an ε fraction of the total weight of $\mathcal{C}$ (formally, $\omega\left(L_{pq}\right) \leq \varepsilon\omega(\mathcal{C})$), the algorithm doubles the weight of all the lines in L_{pq}. Otherwise, this iteration failed, and the algorithm continues to the next iteration.

If after $16k \log n$ iterations the algorithm did not output a solution, then the guess k for σ is too small. In which case, the algorithm doubles the value of k and starts from scratch.

Lemma 6.14. *Given a set P of n points in general position, one can return a set of separating lines S of size $O(\sigma \log \sigma)$ in expected time $O\left(n^2\sigma \log n + \sigma^3 \log n \log^2 \sigma\right)$, where σ is the separability of P.*

Proof: For the sake of completeness, we sketch the proof of correctness of the algorithm. Assume that the guess k is such that $\sigma \leq k \leq 2\sigma$.

Initially, the total weight of the $\mathcal{C}$ is $\binom{n}{2}$. In each successful iteration, the total weight increases by a factor of at most ε. (Assume for the time being that all iterations are successful.) If W_i is the total weight of the lines of $\mathcal{C}$ in the end of the ith successful iteration, then $W_i \leq (1+\varepsilon)^i n^2$. On the other hand, any successful iteration doubles the weight of at least one the lines in the optimal hitting set L_{opt}. For a line $\ell \in L_{\text{opt}}$, let $h(\ell)$ be the number of times its weight had been doubled. We have that $\sum_{\ell \in L_{\text{opt}}} h(\ell) \geq i$ and $W_i \geq \sum_{\ell \in L_{\text{opt}}} 2^{h(\ell)}$. Clearly, the right side is minimized when all the "hits" are distributed uniformly. That is, we have that $W_i \geq \sum_{\ell \in L_{\text{opt}}} 2^{\lfloor i/\sigma \rfloor} \geq \sigma 2^{i/\sigma - 1}$. Consequently,

$$\begin{aligned}\exp\left(\frac{i}{2\sigma} - 1 + \ln\sigma\right) \leq \sigma 2^{i/\sigma-1} \leq W_i \leq (1+\varepsilon)^i n^2 \leq \left(1 + \frac{1}{4k}\right)^i n^2 \leq \left(1 + \frac{1}{4\sigma}\right)^i n^2 \\ \leq \exp\left(\frac{i}{4\sigma} + 2\ln n\right),\end{aligned}$$

since $k \geq \sigma$. This is equivalent to $\frac{i}{2\sigma} - 1 + \ln\sigma \leq \frac{i}{4\sigma} + 2\ln n \iff \frac{i}{4\sigma} \leq 2\ln n - \ln\sigma + 1$, which holds only for $i \leq 8\sigma \ln n$. Namely, the algorithm must stop after this number of successful iterations. Note that the separating lines returned is a sample of size $O(k \log k) = O(\sigma \log \sigma)$ that separates all the points of P.

By the ε-net theorem, every iteration is successful with probability $1 - \varepsilon^c \geq 1 - 1/\sigma^c$, where the constant c

is sufficiently large. As such, the number of failed iterations is tiny compared to the number of successful iterations, and we can ignore this issue.

In each iteration, the algorithm samples a set S of size $r = O(\varepsilon^{-1} \log \varepsilon^{-1}) = O(k \log k)$. The arrangement $\mathcal{A}(S)$ is constructed in $O(r^2)$ time. We then perform n point location queries in $\mathcal{A}(S)$, in $O(\log r) = O(\log k)$ time per query. Thus, the running time for a fixed value of k is $O\big((r^2 + n \log k + n^2) k \log n\big) = O\Big(\Big(k^2 \log^2 k + n \log k + n^2\Big) k \log n\Big)$. Here, the $O(n^2)$ term is the time it takes to scan the lines of $\mathcal{C}$ and update their weights. Summing this over exponentially growing values of $k = 2^0, 2^1, \ldots$, where the final k is at most 2σ, the total running time is bounded by the sum (where $\lg = \log_2$ denotes the binary logarithm):

$$O\left(\sum_{i=0}^{\lceil \lg(2\sigma) \rceil} \Big((2^i)^2 \log^2 2^i + n \log 2^i + n^2\Big) 2^i \log n\right) = O\left(\sum_{i=0}^{\lceil \lg(2\sigma) \rceil} i^2 8^i \log n + n^2 \log n \sum_{i=0}^{\lceil \lg(2\sigma) \rceil} 2^i\right)$$
$$= O\Big(\sigma^3 \log n \log^2 \sigma + n^2 \sigma \log n\Big). \qquad \text{QED.}$$

6.5.2 Faster algorithm

Challenge and the main ideas We want to get a faster algorithm than the "naive" algorithm described above. In the above algorithm, the bottleneck is the $O(n^2)$ term in the running time, which is the result of explicitly maintaining the set $\mathcal{C}$ and the weights for each line in $\mathcal{C}$. Note that the number of iterations the algorithm performs is pretty small, only $O(\sigma \log n)$.

The idea is to maintain the set $\mathcal{C}$ and the weights implicitly. To this end, consider the given set P of n points. In the dual, the set $P^\star$ corresponds to a set of n lines (see Definition 5.2_{p65} and [78, Chapter 25] for more details about duality). A line $\ell \in \mathcal{C}$ corresponds to an intersection point between two lines $p^\star, q^\star \in P^\star$—that is, a vertex of $\mathcal{A}(P^\star)$ (and this vertex represents ℓ uniquely).

Now, in the ith iteration of the (inner) algorithm, it doubles the weight of the lines that are in the set $L_{p_i q_i}$. In other words, the lines that intersect the segment $s_i = p_i q_i$. In the dual, the segment s_i is a double-wedge $D_i = s_i^\star$. The ***double-wedge*** is the region "sandwiched" between the two dual lines $p_i^\star$ and $q_i^\star$, and its interior are all the points in the plane that are above exactly one line of out of $p_i^\star$ and $q_i^\star$.

At the end of the ith iteration, the dual plane is partitioned into the arrangement $\mathcal{A}(\mathcal{D}_i)$, where $\mathcal{D}_i = \{D_1, \ldots, D_i\}$. A vertex $v \in \mathcal{A}(P^\star)$, at the end of the ith iteration, has weight $2^{h(v)}$ where $h(v)$ is the number of double wedges of $\mathcal{D}_i$ that contains v.

Observe that the arrangement $\mathcal{A}(\mathcal{D}_i)$ has complexity $O(i^2)$, which is relatively small, and it can be maintained efficiently. The problem is that to implement the algorithm, one needs to be able to sample efficiently a line from $\mathcal{C}$ according to their weights. To this end, we need to maintain for each face of $\mathcal{A}(\mathcal{D}_i)$ the number of vertices of $\mathcal{A}(P^\star)$ that it contains.

We next describe data structures for counting intersections inside a simple region, sampling a vertex from such a region, and how to maintain such a partition of the plane under insertion of double-wedges.

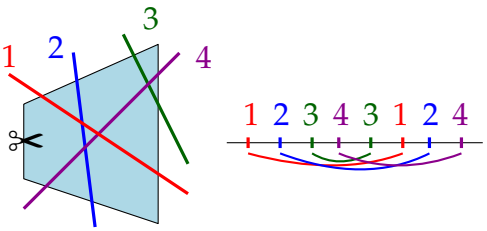


Figure 6.4: Mapping the intersections of lines with a convex polygon to intervals on the real line.

Counting and sampling intersections

Lemma 6.15. *Let ψ be a convex polygon in the plane with constant number of edges, and let L be a set of m lines. The number of vertices of $\mathcal{A}(L)$ that lie in ψ can be computed in $O(m \log m)$ time.*

Furthermore, this algorithm constructs a data structure, using $O(m \log m)$ space, such that one can uniformly at random pick, in $O(\log m)$ time, a vertex of $\mathcal{A}(L)$ that lies in ψ.

Proof: Conceptually, select a point on the boundary of ψ and cut ψ at that point. Take this (now open) polygon and straighten it into a straight line. Finally, translate and rotate the plane, so that this straightened line becomes parallel to the x-axis, see Figure 6.4.

Furthermore, for a line $\ell \in L$ that intersects $\partial\psi$, treat the segment $s = \ell \cap \psi$ as a rubber band. In the end of this straightening process, s became an interval on the x-axis. For two lines $\ell, \ell' \in L$ that have an intersection inside ψ, this results in two intervals I, I', such that each interval contains exactly one endpoint of the other interval in its interior. This also holds in the other direction—two intervals that have this property corresponds to a common intersection of the original lines inside ψ. Counting such pairs is quite easy by sweeping the x-axis from left to right. We next describe this algorithm more formally in the original setup.

Assume that $L = \{\ell_1, \ldots, \ell_m\}$. The algorithm computes the intersection points of the lines of L with the boundary of ψ, and sorts them in their counterclockwise order on the boundary of ψ (starting, say, in the top left vertex of ψ).

The resulting order is a sequence $p_1, \ldots, p_{m'}$, where $m' \leq 2m$, and every point p_i has a label $\alpha = \mathrm{id}(p_i)$ which is the index of the line $\ell_\alpha \in L$ that defines it (i.e., $p_i \in \partial\psi \cap \ell_\alpha$). Next, the algorithm scans this sequence:

- When it encounters an intersection p_j such that $\mathrm{id}(p_j)$ was not seen before, it inserts the line of p_j into a balanced binary search tree (BST), using the value of j for the ordering. This BST has the added feature that each internal node stores the number of elements stored in its subtree.
- When the algorithm encounters a point p_k such that the line defining it was already inserted into the BST (i.e., $\mathrm{id}(p_k) = \mathrm{id}(p_j)$ for some $j < k$), the algorithm reports the number of lines stored in the tree between j and k, which corresponds to the number of lines of L that intersects the line of p_k in ψ. Next, we remove the line of p_k (stored with the key value j) from the tree.

All of these operations can be implemented in $O(\log m)$ time, so that the overall running time is $O(m \log m)$. Observe, that every relevant intersection is counted exactly once by this process.

To get the sampling data structure, rerun the above algorithm using a BST with persistence. This persistence costs $O(\log m)$ additional space per operation, since we use the path copying approach. This modification does not effect the overall running time. Thus, the resulting data structure uses $O(m \log m)$ space. Now, every line $\ell \in L$, corresponds to an interval $I_\ell = [i(\ell), i'(\ell)]$ in the BST. Furthermore, the lines intersecting ℓ in ψ, are stored in the BST (in the version just after ℓ was deleted) in the interval I_ℓ.

As such, every line intersecting ψ has an associated interval, with an associated weight (i.e., the number of intersections assigned to it by the construction). To pick a random vertex, the algorithm first picks an interval according to their weights—this corresponds to a random line ℓ. Next, given this random line, the algorithm picks a random element stored in the $O(\log m)$ subtrees representing the lines in I_ℓ. Since the algorithm used path copying, it has the exact number of lines stored in each subtree, and it is straightforward to sample a line in uniform. This second random line ℓ', such that $\ell \cap \ell' \in \psi$ is the desired random vertex. QED.

Sampling a trapezoid The algorithm maintains a collection of m trapezoids that are interior-disjoint, such that their (disjoint) union covers the plane. Furthermore, assume that each such trapezoid ψ already has the data structure of Lemma 6.15 built for it.

Definition 6.5. Consider a set $\mathcal{D}$ of double-wedges and a trapezoid ψ such that its interior is contained in a single face of $\mathcal{A}(\mathcal{D})$. For a set of lines L, the number of vertices of $\mathcal{A}(L)$ in ψ is the ***support*** of ψ, and it is denoted by $\#(\psi)$. The ***depth*** of ψ is the number of double-wedges of $\mathcal{D}$ that fully contain ψ in their interior. The depth of ψ is denoted by $\mathrm{depth}(\psi)$. The ***mass*** of ψ is defined as $\mathrm{mass}(\psi) = \#(\psi) 2^{\mathrm{depth}(\psi)}$.

Lemma 6.16. *Given a (dynamic) set at most m interior-disjoint trapezoids, covering the plane, each with the associated data structure of Lemma 6.15 and their known mass, one can sample a random vertex from $\mathcal{A}(L)$ in $O(\log m + \log m')$ time, where m' is the maximum size of a conflict list of such a trapezoid. Furthermore, one can update this data structure under insertion and deletion in $O(\log m)$ time.*

Proof: The task at hand is to pick a vertex of $\mathcal{A}(L)$ uniformly at random according to these weights. To this end, we construct a balanced binary search tree having the trapezoids as leafs—a trapezoid is stored together with its mass. Every internal node of this tree has the total mass of the leafs in its subtree.

Now, one can traverse down the tree randomly, starting at the root, as follows. If the current node is u, consider its two children v and v'. The algorithm picks an integer number randomly and uniformly in the range $[1, 1 + \mathrm{mass}(v) + \mathrm{mass}(v')]$. If this number is in the range $[1, \mathrm{mass}(v)]$, the algorithm continues the traversal into v, otherwise, it continues into v'. Clearly, this traversal randomly and uniformly chooses a leaf of the tree (according to their mass). Once the algorithm arrived to such a leaf, it uses the data structure of Lemma 6.15 to pick a random vertex inside the associated trapezoid. QED.

Maintaining vertex weights efficiently under insertions Our purpose here is to present an efficient data structure that solves the following problem.

Problem 6.1. Given a set L of n lines and a parameter k, we would like to maintain a vertical decomposition of the plane, such that each trapezoid ψ in this decomposition maintains the sampling data structure of Lemma 6.15 for the vertices of $\mathcal{A}(L)$. This data structure should support insertions of up to $O(k \log n)$ double-wedges. Here, each trapezoid maintains its support, depth, and mass, see Definition 6.5.

The basic scheme

Lemma 6.17. *One can maintain a data structure for Problem 6.1, over $O(k \log n)$ insertions, with total running time $O((k^3 + nk) \log^3 n)$.*

Proof: Let S be a random sample of L of size $K = O(k \log n)$, where L is the set of n lines that are dual to the original set of points. Compute the vertical decomposition of S. For each trapezoid ψ in this decomposition, we compute the conflict list of ψ (i.e., the set of lines from L intersecting the interior of ψ). This can be done in $O(K^2 + Kn)$ time, using standard algorithms, see [18]. Next, the algorithm computes for each trapezoid the data structure of Lemma 6.15.

By the ε-net theorem, every vertical trapezoid that does not intersect a line of S in its interior intersects at most εn lines of L (where $\varepsilon = 1/4k$). This property holds with high probability. As such, the conflict lists that the algorithm deals with are of size $O(n/k)$.

Let $L_0 = S$. In the ith iteration, the ith double-wedge D_i is inserted. To this end, the two lines ℓ_i, ℓ_i' bounding the double wedge are inserted into the current vertical decomposition, splitting and merging trapezoids as necessary. At the end of this process we have the vertical decomposition of $L_i = L_{i-1} \cup \{\ell_i, \ell_i'\}$. This involves creating $O(K + i)$ new trapezoids, since the zone complexity of a line in $\mathcal{A}(L_{i-1})$ is $O(K + i) = O(K)$, and $i = O(K)$. For each such trapezoid we rebuild the data structure of Lemma 6.15, which takes overall $O((n/k) \log(n/k) \cdot K) = O(n \log^2 n)$ time. Finally, we scan all the vertical trapezoids, and update their depth count, if they are contained inside the inserted wedge. This takes (naively) $O(K^2)$ time.

Recall that we perform $O(K)$ insertions in total, and therefore the overall running time of the data structure is $O\Big(K\Big(K^2 + n \log^2 n\Big)\Big) = O((k^3 + nk) \log^3 n)$. QED.

A more efficient scheme The overall running time of Lemma 6.17 can be further improved by using dynamic partition trees.

Lemma 6.18. *One can maintain a data structure for Problem 6.1 with running time $O(nk \log^3 n + k^2 \log^{O(1)} n)$. (This running time includes $O(k \log n)$ double-wedge insertions.) Furthermore, one can sample a random vertex of $\mathcal{A}(L)$ according to their weight in $O(\log n)$ time.*

Proof: Partition trees are used to maintain the depth of the vertical trapezoids. This maintenance step is the bottleneck in the scheme of Lemma 6.17, since the algorithm must scan all of the existing trapezoids to update their depth after each insertion of a double wedge.

A partition tree is a hierarchical partition of the point set, until each leaf has a constant number of points. Each node uses a partition (see Definition 6.4) to break its point set into subsets, and for each subset a partition tree is constructed recursively. Performing a simplex query in a partition tree is done by starting at the root, inspecting its children simplices. If such a simplex Δ lies entirely within the query, the algorithm reports the number of points inside it. Otherwise if Δ intersects the query, the algorithm recurses on that child node. Given a set of n points in $\mathbb{R}^2$, Matoušek showed that one can construct a partition tree in $O(n \log n)$ time and return the number of points inside the simplex query in time $O(\sqrt{n} \log^{O(1)} n)$ [110].

For our purposes, we pick a point inside a vertical trapezoid (in the current vertical decomposition) to represent it. Overall, there are $m = O(K^2) = O(k^2 \log^2 n)$ representatives at any given time. We next build the data structure of Matoušek [110] to dynamically maintain this point-set under insertions and deletions (each operation takes amortized $O(\log^2 m)$ time). Updating the weight of a trapezoid corresponds to two simplex queries, where we have to increase the depth count for the canonical sets reported by this range-searching query. There are $O(\sqrt{m} \log^{O(1)} m) = O(k \log^{O(1)} n)$ such canonical sets, and this is the time to perform such an update. Thus an insertion of a double wedge with respect to this partition tree takes $O(K \log^2 K + k \log^{O(1)} n)$ time. Therefore, over the $O(K)$ insertions, the algorithm requires $O(k^2 \log^{O(1)} n)$ time to maintain the weights of the vertices of $\mathcal{A}(L)$.

Using the above, and the sampling data structure of Lemma 6.16, implies the claim. QED.

Putting everything together

Remark 6.4 (More efficient point-location). Each iteration of the algorithm needs to solve the following subproblem. Given a set of m lines L and n points P, compute for each point $p \in P$ the face of $\mathcal{A}(L)$ containing p. Agarwal et al. [2] describe an algorithm for this problem with running time $O((n + m + n^{2/3} m^{2/3}) \log n)$. This subroutine can be applied in each iteration of our algorithm on the $m = O(k \log k)$ lines sampled. Substituting the value for m in the preceding bound, we conclude the subroutine can be completed in time $O\left(n \log n + k \log^2 n + n^{2/3} k^{2/3} \log^2 n\right)$.

Theorem 6.4. *Given a set P of n points in the plane, one can compute a set of $O(\sigma \log \sigma)$ lines that separates all the points of P, where σ is the minimal set of lines that separates P. The overall expected running time of this algorithm is $O\left(n^{2/3} \sigma^{5/3} \log^{O(1)} n\right)$.*

Proof: We implement the algorithm of Lemma 6.14 using the data structure of Lemma 6.18 to maintain the vertices of the dual arrangement, and use the point-location data structure of Remark 6.4. For a fixed value

of k, the algorithm performs $O(k \log n)$ inner iterations, and the resulting running time is

$$O\big((n + k + n^{2/3}k^{2/3})k \log^3 n + nk \log^3 n + k^2 \log^{O(1)} n\big) = O\big(nk \log^3 n + n^{2/3}k^{5/3} \log^{O(1)} n\big).$$

Summing the above bound over exponentially growing values of k, ending at $k = O(\sigma)$ (as in the proof of Lemma 6.14), the overall running time is $O\Big(n\sigma \log^3 n + n^{2/3}\sigma^{5/3} \log^{O(1)} n\Big)$. Observe that by Remark 6.1 (B), $\sigma = \text{sep}(P) = \Omega(\sqrt{n})$ which implies that the second term is bigger than the first term. The result then follows. QED.

Remark 6.5. To appreciate Theorem 6.4, consider the grid-like case where $\sigma = O(\sqrt{n})$. The running time then becomes $O(n^{3/2} \log^{O(1)} n)$, which is well below quadratic time. The worst case for this algorithm is when $\sigma = \Omega(n)$ (for example, if the input points are in convex position), where the running time becomes $O(n^{7/3} \log^{O(1)} n)$.

7 Active-learning a convex body in low dimensions

> *Failure is simply the opportunity to begin again, this time more intelligently.*
>
> **— Henry Ford**

Consider a set $P \subseteq \mathbb{R}^d$ of n points, and a convex body C provided via a separation oracle. The task at hand is to decide for each point of P if it is in C using the fewest number of oracle queries. We show that one can solve this problem in two and three dimensions using $O(\bigcirc_P \log n)$ queries, where $\bigcirc_P$ is the size of the largest subset of points of P in convex position. In 2D, we provide an algorithm that efficiently generates these adaptive queries.

Furthermore, we show that in two dimensions one can solve this problem using $O(\circledcirc(P, C) \log^2 n)$ oracle queries, where $\circledcirc(P, C)$ is a lower bound on the minimum number of queries that any algorithm for this specific instance requires.

As an application of the above, we show that the discrete geometric median of a point set P in $\mathbb{R}^2$ can be computed in $O(n \log^2 n \, (\log n \log\log n + \bigcirc_P))$ expected time.

7.1 BACKGROUND

7.1.1 Active learning

Active learning is a subfield of machine learning. At any time, the learning algorithm is able to query an oracle for the label of a particular data point. One model for active learning is the membership query synthesis model [9]. Here, the learner wants to minimize the number of oracle queries, as such queries are expensive—they usually correspond to either consulting with a specialist, or performing an expensive computation. In this setting, the learning algorithm is allowed to query the oracle for the label of any data point in the instance space. See [142] for a more in-depth survey on the various active learning models.

7.1.2 PAC learning

A classical approach for learning is using random sampling, where one gets labels for the samples (i.e., in the above setting, the oracle is asked for the labels of all items in the random sample). PAC learning

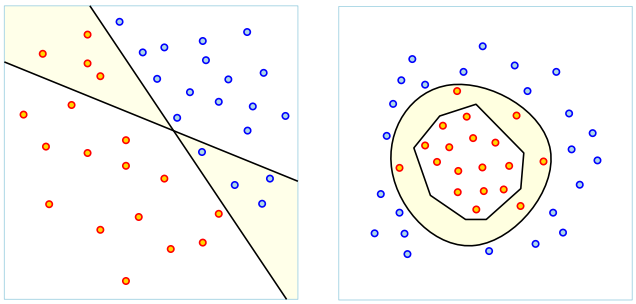
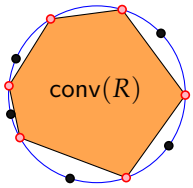


Figure 7.1: The shaded region shows the symmetric difference between the hypothesis and true classifier. (I) Learning halfspaces. (II) Learning arbitrary convex regions.

studies the size of the sample needed. For example, consider the problem of learning a halfplane for n points $P \subset \mathbb{R}^2$, given a parameter $\varepsilon \in (0,1)$. The first stage is to take a labeled random sample $R \subseteq P$. The algorithm computes any halfplane that classifies the sample correctly (i.e., the hypothesis). The misclassified points lie in the symmetric difference between the learned halfplane, and the (unknown) true halfplane, see Figure 7.1. In this case, the error region is a double wedge, and it is well known that its VC-dimension [151] is a constant (at most eight). As such, by the ε-net Theorem [88, Theorem 4.1$_{\text{p44}}$], a sample of size $O(\varepsilon^{-1} \log \varepsilon^{-1})$ is an ε-net for double wedges, which implies that this random sampling algorithm has at most εn error.

A classical example of a hypothesis class that cannot be learned is the set of convex regions (even in the plane). Indeed, given a set of points P in the plane, any sample $R \subseteq P$ cannot distinguish between the true region being $\mathsf{conv}(R)$ or $\mathsf{conv}(P)$. Intuitively, this is because the hypothesis space in this case grows exponentially in the size of the sample (instead of polynomially).



We stress that the above argument does not necessarily imply these types of hypothesis classes are unlearnable in practice. In general, there are other ways for learning algorithms to handle hypothesis classes with high (or even infinite) VC-dimension (for example, using regularization or assuming there is a large margin around the decision boundary).

7.1.3 Weak ε-nets

Because ε-nets for convex ranges do not exist, an interesting direction to overcome this problem is to define weak ε-nets. Recall Definition 4.5$_{\text{p44}}$: a set of points R in the plane—not necessarily a subset of P—is a *weak ε-net* for P if any convex body C containing at least εn points of P also contains a point of R. As discussed in Section 4.1$_{\text{p40}}$, the state of the art weak ε-net construction is by Rubin [137, 138]. However, weak ε-nets cannot be used for learning such concepts. Indeed, the analysis above required an ε-net for the symmetric difference of two convex bodies of finite complexity, see Figure 7.1.

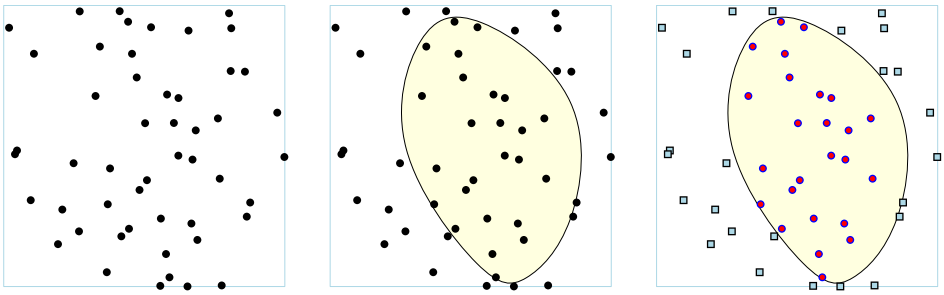


Figure 7.2: (I) A set of points P. (II) The unknown convex body C. (III) Classifying all points of P as either inside or outside C.

7.1.4 PAC learning with additional parameters

If one assumes the input instance obeys some additional structural properties, then random sampling can be used. For example, suppose that the point set P has at most k points in convex position. For an arbitrary convex body C, the convex hull $\mathsf{conv}(P \cap C)$ has complexity at most k. Let $R \subseteq P$ be a random sample, and C' be the learned classifier for R. The region of error is the symmetric difference between C and C'. In particular, since k-vertex polytopes in $\mathbb{R}^d$ have VC-dimension bounded by $O(d^2 k \log k)$ [101], this implies that the error region also has VC-dimension at most $O(d^2 k \log k)$. Hence if R is a random sample of size $O(d^2 k \log k \varepsilon^{-1} \log \varepsilon^{-1})$, the ε-net Theorem [88, Theorem 4.1$_{\text{p44}}$] implies that this sampling algorithm has error at most εn. However, even for a set of n points chosen uniformly at random from the unit square $[0,1]^2$, the expected number of points in convex position is $O(n^{1/3})$ [8]. Since we want $|R| < n$, this random sampling technique is only useful when ε is larger than $\log^2 n / n^{2/3}$ (ignoring constants).

To summarize the above discussions, random sampling on its own does not seem powerful enough to learn arbitrary convex bodies, even if one allows some error to be made. In this paper we focus on developing algorithms for learning convex bodies in low dimensions, where the algorithms are deterministic and do not make any errors.

7.2 PROBLEM, MOTIVATION, AND RESULTS

7.2.1 The problem

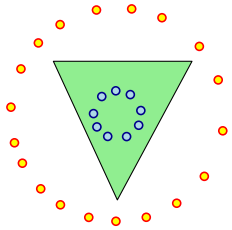
In this chapter, we consider a variation on the active learning problem, in the membership query synthesis model. Suppose that the learner is trying to learn an unknown convex body C in $\mathbb{R}^d$. Specifically, the learner is provided with a set P of n unlabeled points in $\mathbb{R}^d$, and the task is to label each point as either inside or outside C, see Figure 7.2. For a query $z \in \mathbb{R}^d$, the oracle either reports that $z \in C$, or returns a hyperplane separating z and C (as a proof that $z \notin C$). Note that if the query is outside the body, the oracle answer is significantly more informative than just the label of the point. The problem is to minimize the overall number of queries performed. We remark that this separation model is the same model as used in Section 4.3$_{\text{p44}}$.

Recall that the model was previously utilized to construct functional nets, where the aim was to detect the presence of heavy bodies while avoiding the side effects of the curse of dimensionality.

7.2.2 Hard and easy instances

Note that in the worst case, an algorithm may have to query the oracle for all input points—such a scenario happens when the input points are in convex position, and any possible subset of the points can be the points in the (appropriate) convex body. As such, the purpose here is to develop algorithms that are *instance sensitive*—if the given instance is easy, they work well. If the given instance is hard, they might deteriorate to the naive algorithm that queries all points.

Natural inputs where one can hope to do better, are when relatively few points are in convex position. Such inputs are grid points, or random point sets, among others. However, there are natural instances of the problem that are easy, despite the input having many points in convex position. For example, consider when the convex body is a triangle, with the input point set being $n/2$ points spread uniformly on a tiny circle centered at the origin, while the remaining $n/2$ points are outside the convex body, spread uniformly on a circle of radius 10 centered at the origin. Clearly, such a point set can be fully classified using a sequence of a constant number of oracle queries. See Figure 7.6 for some related examples.



7.2.3 Additional motivation and previous work

Separation oracles The use of separation oracles is a common tool in optimization (e.g., solving exponentially large linear programs) and operations research. It is natural to ask what other problems can be solved efficiently when given access to this specific type of oracle. For example, Bárány and Füredi [17] study the problem of computing the volume of a convex body in $\mathbb{R}^d$ given access to a separation oracle.

Other types of oracles Various models of computation utilizing oracles have been previously studied within the computational geometry community. Examples of other models include nearest-neighbor oracles (i.e., black-box access to nearest neighbor queries over a point set P) [76], proximity probes (in which given a convex polygon C and a query z, returns the distance from z to C) [130], and linear queries. Recently, Ezra and Sharir [61] gave an improved algorithm for the problem of point location in an arrangement of hyperplanes. Here, a *linear query* consists of a point x and a hyperplane h, and outputs either that x lies on h, or else the side of h contains x. Alternatively, their problem can be interpreted as querying whether or not a given point lies in a halfspace h^+. Here, we study the more general problem as the convex body can be the intersection of many halfspaces.

Furthermore, other types of active learning models (in addition to the membership query model) have also been studied within the learning community, see, for example, [9].

Active learning As discussed, the problem at hand can be interpreted as active learning a convex body in relation to a set of points P that need to be classified (as either inside or outside the body), where the queries are via a separation oracle. We are unaware of any work directly on this problem in the computational geometry community, while there is some work in the learning community that studies related active learning classification problems [51, 73, 95, 142].

For example, Kane et al. [95] study the problem of actively learning halfspaces with access to *comparison queries*. Given a halfspace h^+ to learn, the model has two types of queries:

(i) label queries (given $x \in \mathbb{R}^d$, is $x \in h^+$?), and

(ii) comparison queries (given $x_1, x_2 \in \mathbb{R}^d$, is x_1 closer to the boundary of h^+ than x_2?).

For example, they show that in the plane, one can classify all points using $O(\log n)$ comparison/label queries in expectation.

Discretely optimizing convex functions As an application of this particular query model, we explore the connection between active learning a convex body and minimizing a convex function. Concretely, suppose we are given a set of n points P in the plane and a convex function $f : \mathbb{R}^2 \to \mathbb{R}$ equipped with an oracle that can evaluate f or the derivative of f at a given point. Our goal is to compute the point in P minimizing $\min_{p \in P} f(p)$ using the fewest number of oracle queries (i.e., evaluations of f or the derivative). We discuss the result in full in Section 7.6.

We show that there is a natural connection between the studied query model and this problem. Namely, the level sets of a convex function are convex bodies, and the gradient of f can be used to construct separating lines for the level set. Thus, developing algorithms for active learning a convex body in the membership query synthesis model in conjunction with the two aforementioned facts leads to alterative methods for minimizing a convex function over a discrete collection of points. Importantly, the running time of such algorithms depend not only on how quickly we can evaluate f, but also on the structure of the point set P, as we aim to develop instance sensitive algorithms.

7.2.4 Results

(A) We develop a greedy algorithm, for points in the plane, that solves the problem using $O(\bigcirc_P \log n)$ oracle queries, where $\bigcirc_P$ is the size of the largest subset of points of P in convex position. See Theorem 7.1. It is known that for a random set of n points in the unit square, $\mathbb{E}[\bigcirc_P] = \Theta(n^{1/3})$ [8], which readily implies that classifying these points can be solved using $O(n^{1/3} \log n)$ oracle queries. A similar bound holds for the $\sqrt{n} \times \sqrt{n}$ grid. An animation of this algorithm is on YouTube [75]. We also show that this algorithm can be implemented efficiently, using dynamic segment trees, see Lemma 7.5.

We remark that Kane et al. [95] develop a framework and randomized algorithm for learning a concept C, where the expected number of queries depends near-linearly on a parameter they define as the *inference dimension* [95, Definition III.1] of the concept class. For our problem, one can show that the inference dimension is $O(\hexagon_P)$. As a corollary of their framework, one can obtain a randomized algorithm that solves our problem where the expected number of queries is $O(\hexagon_P \log n)$. Note that our algorithm has the same query complexity but is deterministic. See also Section 7.3.4 for additional details.

(B) The above algorithm naturally extends to three dimensions, also using $O(\hexagon_P \log n)$ oracle queries. While the proof idea is similar to that of the algorithm in 2D, we believe the analysis in three dimensions is also technically interesting. See Theorem 7.3.

(C) For a given point set P and convex body C, we define the separation price $\circledcirc(P, C)$ of an instance (P, C), and show that any algorithm classifying the points of P in relation to C must make at least $\circledcirc(P, C)$ oracle queries (Lemma 7.14).

As an aside, we show that when P is a set of n points chosen uniformly at random from the unit square and C is a (fixed) smooth convex body, $\mathbb{E}[\circledcirc(P, C)] = O(n^{1/3})$, and this bound is tight when C is a disk (the bound also generalizes to higher dimensions). For randomly chosen points, the separation price is related to the expected size of the convex hull of $P \cap C$, which is also known to be $\Theta(n^{1/3})$ [153]. We believe this result may be of independent interest; the result is proven in [84].

(D) In Section 7.5 we present an improved algorithm for the 2D case, and show that the number of queries made is $O(\circledcirc(P, C) \log^2 n)$. This result is a $O(\log^2 n)$ approximation to the optimal solution, see Theorem 7.4.

(E) Section 7.6 presents an application of the above results, we consider the problem of minimizing a *convex* function $f : \mathbb{R}^2 \to \mathbb{R}$ over a point set P. Specifically, the goal is to compute $\arg\min_{p \in P} f(p)$. If f and its derivative can be efficiently evaluated at a given query point, then f can be minimized over P using $O(\hexagon_P \log^2 n)$ queries to f (or its derivative) in expectation. We refer the reader to Lemma 7.22.

Given a set of n points P in $\mathbb{R}^d$, the discrete geometric median of P is a point $p \in P$ minimizing the function $\sum_{q \in P} \|p - q\|$. As a corollary of Lemma 7.22, we obtain an algorithm for computing the discrete geometric median for n points in the plane. The algorithm runs in $O(n \log^2 n \cdot (\log n \log\log n + \hexagon_P))$ expected time. See Lemma 7.23. In particular, if P is a set of n points chosen uniformly at random from the unit square, it is known that $\mathbb{E}[\hexagon_P] = \Theta(n^{1/3})$ [8] and hence the discrete geometric median can be computed in $O(n^{4/3} \log^2 n)$ expected time.

While there has been ample work on approximating the geometric median (recently, Cohen et al. [50] gave a $(1+\varepsilon)$-approximation algorithm to the geometric median in $O(dn \log^3(1/\varepsilon))$ time), we are unaware of any *exact* sub-quadratic algorithm for the discrete case even in the plane.

7.3 THE GREEDY ALGORITHM IN TWO DIMENSIONS

7.3.1 Preliminaries

For a set of points $P \subseteq \mathbb{R}^2$, let $\mathsf{conv}(P)$ denote the convex hull of P. Given a convex body $C \subseteq \mathbb{R}^d$, two points $p, x \in \mathbb{R}^d \setminus \mathsf{int}(C)$ are ***mutually visible***, if the segment px does not intersect $\mathsf{int}(C)$, where $\mathsf{int}(C)$ is the interior of C. We also use the notation $P \cap C = \{p \in P \mid p \in C\}$.

Recall also the definition of a centerpoint (Definition 3.1$_{\text{p29}}$). For a point set $P \subseteq \mathbb{R}^d$, a ***centerpoint*** of P is a point $c \in \mathbb{R}^d$, such that for any closed halfspace h^+ containing c, we have $|h^+ \cap P| \geq |P| / (d+1)$. A centerpoint always exists, and it can be computed exactly in $O(n^{d-1} + n \log n)$ time [33].

Let C be a convex body in $\mathbb{R}^d$ and $q \in \mathbb{R}^d$ be a point such that q lies outside C. A hyperplane h ***separates*** q from C if q lies in the *closed* halfspace h^+ bounded by h, and C is contained in the *open* halfspace h^- bounded by h. This definition allows the separating hyperplane to contain the point q, and will simplify the descriptions of the algorithms.

Given a set of points P in $\mathbb{R}^2$ and a convex body C specified via a separation oracle, recall that the problem is to classify, for all the points of P, whether or not they are in C, using the fewest oracle queries possible. We define some operations the algorithm will use before stating the algorithm in full. Finally, we analyze the the number of queries the algorithm performs.

7.3.2 The algorithm

Initially, the algorithm copies P into a set U of unclassified points. The algorithm is going to maintain an inner approximation $B \subseteq C$. There are two types of updates (Figure 7.3 illustrates the two operations):

(A) **expand**(p): Given a point $p \in C \setminus B$, the algorithm is going to:

  (i) Update the inner approximation: $B \leftarrow \mathsf{conv}(B \cup \{p\})$.

  (ii) Remove (and mark) newly covered points: $U \leftarrow U \setminus B$.

(B) **remove**(ℓ): Given a closed halfplane ℓ^+ such that $\mathsf{int}(C) \cap \ell^+ = \emptyset$, the algorithm marks all the points of $U_\ell = U \cap \mathsf{int}(\ell^+)$ as being outside C, and sets $U \leftarrow U \setminus U_\ell$.

The algorithm repeatedly performs rounds, as described next, until the set of unclassified points is empty.

At every round, if the inner approximation B is empty, then the algorithm sets $U^+ = U$. Otherwise, the algorithm picks a line ℓ that is tangent to B with the largest number of points of U on the other side of ℓ than B. Let ℓ^- and ℓ^+ be the two closed halfspace bounded by ℓ, where $B \subseteq \ell^-$. The algorithm computes the point set $U^+ = U \cap \ell^+$. We have two cases:

A. Suppose $|U^+|$ is of constant size. The algorithm queries the oracle for the status of each of these points. For every point $p \in U^+$, such that $p \in C$, the algorithm performs **expand**(p). Otherwise, the oracle returned a separating line ℓ, and the algorithm calls **remove**(ℓ^+).

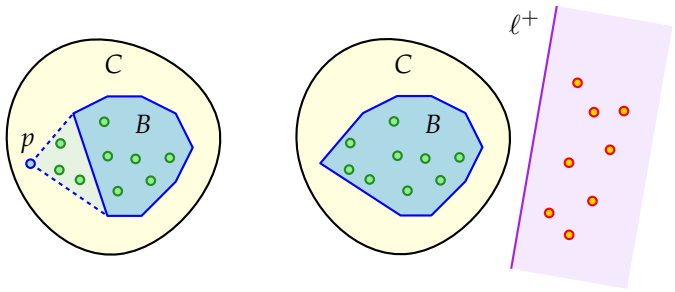


Figure 7.3: (I) Performing **expand**(p), and marking points inside C. (II) Performing **remove**(ℓ), and marking points outside C.

B. Otherwise, $|U+|$ does not have constant size. The algorithm computes a centerpoint $c \in \mathbb{R}^2$ for U^+, and asks the oracle for the status of c. There are two possibilities:

   B.I. If $c \in C$, then the algorithm performs **expand**(c).

   B.II. If $c \notin C$, then the oracle returned a separating line $\hbar$, and the algorithm performs **remove**$(\hbar)$.

7.3.3 Analysis

Let B_i be the inner approximation at the start of the ith iteration, and let z be the first index where B_z is not an empty set. Similarly, let U_i be the set of unclassified points at the start of the ith iteration, where initially $U_1 = U$.

Lemma 7.1. *The number of (initial) iterations in which the inner approximation is empty is $z = O(\log n)$.*

Proof: As soon as the oracle returns a point that is in C, the inner approximation is no longer empty. As such, we need to bound the initial number of iterations where the oracle returns that the query point is outside C. Let $f_i = |U_i|$, and note that $U_1 = P$ and $f_1 = |P| = n$. Let c_i be the centerpoint of U_i, which is the query point in the ith iteration (c_i is outside C). As such, the line separating c_i from C returned by the oracle has at least $f_i/3$ points of U_i on the same side as c_i by the centerpoint property. All of these points get labeled in this iteration, and it follows that $f_{i+1} \leq (2/3)f_i$. This implies the claim, since $f_z < 1$ for $z = \lceil \log_{3/2} n \rceil + 1$. QED.

Definition 7.1 (Visibility graph). Consider the graph G_i over U_i, where two points $p, r \in U_i$ are connected $\iff$ the segment pr does not intersect the interior of B_i.

The visibility graph as an interval graph For a point $p \in U_i$, let $I_i(p)$ be the set of all directions v (i.e., vectors of length 1) such that there is a line perpendicular to v that separates p from B_i. Formally, a line ℓ separates p from B_i, if the interior of B_i is on one side of ℓ and p is on the (closed) other side of ℓ (if $p \in \ell$, the line is still considered to separate the two). Clearly, $I_i(p)$ is a circular interval on the unit circle. See Figure 7.4. The resulting set of intervals is $\mathcal{I}_i = \{I_i(p) \mid p \in U_i\}$. It is easy to verify that the intersection graph of $\mathcal{I}_i$ is G_i. Throughout the execution of the algorithm, the inner approximation B_i grows monotonically, this in turn

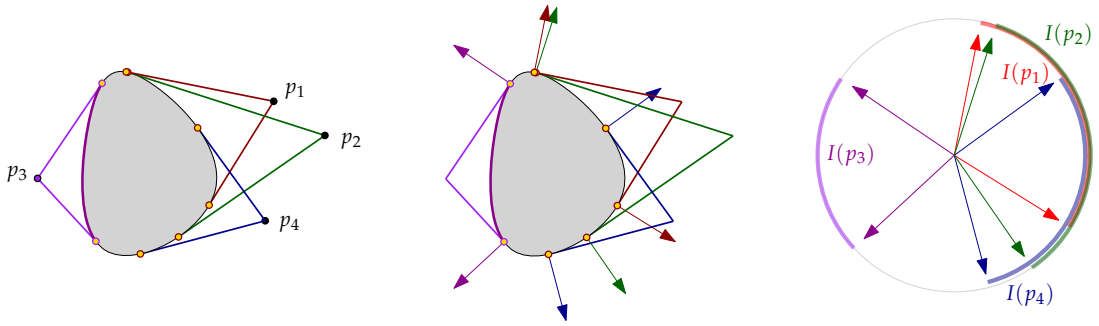


Figure 7.4: Four points and a convex body with their associated circular intervals.

implies that the visibility intervals shrink over time; that is, $I_i(p) \subseteq I_{i-1}(p)$, for all $p \in P$ and i. Intuitively, in each round, either many edges from G_i are removed (because intervals had shrunk and they no longer intersect), or many vertices are removed (i.e., the associated points are classified).

Definition 7.2. Given a set $\mathcal{I}$ of objects (e.g., intervals) in a domain D (e.g., unit circle), the ***depth*** of a point $p \in D$ is the number of objects in $\mathcal{I}$ that contain p. Let $\text{depth}(\mathcal{I})$ be the maximum depth of any point in D.

When it is clear, we use $\text{depth}(G)$ to denote $\text{depth}(\mathcal{I})$, where $G = (\mathcal{I}, E)$ is the intersection graph of the intervals $\mathcal{I}$ as defined above. Throughout, we commonly refer to G as the ***intersection graph***.

First, we bound the number of edges in this visibility graph G and then argue that in each iteration, either many edges of G are discarded or vertices are removed (as they are classified).

Lemma 7.2. *Let $\mathcal{I}$ be a set of n intervals on the unit circle, and let $G = (\mathcal{I}, E)$ be the associated intersection graph. Then $|E| = O(\alpha\omega^2)$, where $\omega = \text{depth}(\mathcal{I})$ and $\alpha = \alpha(G)$ is the size of the largest independent set in G. Furthermore, the upper bound on $|E|$ is tight.*

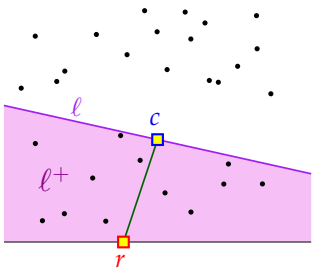
Proof: Let J be the largest independent set of intervals in G. The intervals of J divide the circle into $2|J|$ (atomic) circular arcs. Consider such an arc γ, and let $K(\gamma)$ be the set of all intervals of $\mathcal{I}$ that are fully contained in γ. All the intervals of $K(\gamma)$ are pairwise intersecting, as otherwise one could increase the size of the independent set. As such, all the intervals of $K(\gamma)$ must contain a common intersection point. It follows that $|K(\gamma)| \leq \omega$.

Let $K'(\gamma)$ be the set of all intervals intersecting γ. This set might contain up to 2ω additional intervals (that are not contained in γ), as each such additional interval must contain at least one of the endpoints of γ. Namely, $|K'(\gamma)| \leq 3\omega$. In particular, any two intervals intersecting inside γ both belong to $K'(\gamma)$. As such, the total number of edges contributed by $K'(\gamma)$ to G is at most $\binom{3\omega}{2} = O(\omega^2)$. Since there are at most 2α arcs under consideration, the total number of edges in G is bounded by $O(\alpha\omega^2)$, which implies the claim.

The lower bound is easy to see by taking an independent set of intervals of size α, and replicating every interval ω times. QED.

Lemma 7.3. *Let P be a set of n points in the plane lying above the x-axis, c be a centerpoint of P, and $S = \binom{P}{2}$ be set of all segments induced by P. Next, consider any point r on the x-axis. Then, the segment cr intersects at least $n^2/36$ segments of S.*

Proof: If the segment cr intersects the segment p_1p_2, for $p_1, p_2 \in P$, then we consider p_1 and p_2 to no longer be mutually visible. It suffices to lower bound the number of pairs of points that lose mutual visibility of each other.



Consider a line ℓ passing through the point c. Let ℓ^+ be the closed halfspace bounded by ℓ containing r. Note that $|P \cap \ell^+| \geq n/3$, since c is a centerpoint of P, and $c \in \ell$. Rotate ℓ around c until there are $\geq n/6$ points on each side of rc in the halfspace ℓ^+. To see why this rotation of ℓ exists, observe that the two halfspaces bounded by the line spanning rc, have zero points on one side, and at least $n/3$ points on the other side—a continuous rotation of ℓ between these two extremes, implies the desired property.

Observe that points in ℓ^+ and on opposite sides of the segment cr cannot see each other, as the segment connecting them must intersect cr. Consequently, the number of induced segments that cr intersects is at least $n^2/36$. QED.

For a graph G, we let $E(G)$ denote the set of edges in G, and let $|E(G)|$ denote the number of edges in G.

Lemma 7.4. *Let G_i be the intersection graph, in the beginning of the ith iteration, and let $m_i = |E(G_i)|$. After the ith iteration of the greedy algorithm, we have $m_{i+1} \leq m_i - \omega^2/36$, where $\omega = \mathrm{depth}(G_i)$.*

Proof: Recall that in the algorithm $U^+ = U_i \cap \ell^+$ is the current set of unclassified points and ℓ is the line tangent to B_i, where ℓ^+ is the closed halfspace that avoids the interior of B_i and contains the largest number of unlabeled points of U_i. We have that $\omega = |U^+|$.

If a **remove** operation was performed in the ith iteration, then the number of points of U^+ that are discarded is at least $\omega/3$. In this case, the oracle returned a separating line $\hbar$ between a centerpoint c of U^+ and the inner approximation. For the halfspace $\hbar^+$ containing c, we have $t_i = |U^+ \cap \hbar^+| \geq |U^+|/3 \geq \omega/3$. Furthermore, all the points of U^+ are pairwise mutually visible (in relation to the inner approximation B_i). Namely,

$$m_{i+1} = \left|E\left(G_i - (U^+ \cap \hbar^+)\right)\right| \leq m_i - \binom{t_i}{2} \leq m_i - \omega^2/36.$$

If an **expand** operation was performed, the centerpoint c of U^+ is added to the current inner approximation B_i. Let r be a point in $\ell \cap B_i$, and let c_i be the centerpoint of U_i computed by the algorithm. By Lemma 7.3

applied to r, c and U^+, we have that at least $\omega^2/36$ pairs of points of U^+ are no longer mutually visible to each other in relation to B_{i+1}. We conclude, that at least $\omega^2/36$ edges of G_i are no longer present in G_{i+1}.QED.

Definition 7.3. A subset of points $X \subseteq P \subseteq \mathbb{R}^2$ are in ***convex position***, if all the points of X are vertices of $\mathsf{conv}(X)$ (note that a point in the middle of an edge is not considered to be a vertex). The ***index*** of P, denoted by $\bigcirc_P$, is the cardinality of the largest subset of P of points that are in convex position.

Theorem 7.1. *Let C be a convex body provided via a separation oracle, and let P be a set of n points in the plane. The greedy classification algorithm performs $O\big((\bigcirc_P + 1)\log n\big)$ oracle queries. The algorithm correctly identifies all points in $P \cap C$ and $P \setminus C$.*

Proof: By Lemma 7.1, the number of iterations (and also queries) in which the inner approximation is empty is $O(\log n)$, and let $z = O(\log n)$ be the first iteration such that the inner approximation is not empty. It suffices to bound the number of queries made by the algorithm after the inner approximation becomes non-empty.

For $i \geq z$, let $G_i = (U_i, E_i)$ denote the visibility graph of the remaining unclassified points U_i in the beginning of the ith iteration. Any independent set in G_i corresponds to a set of points $X \subseteq P$ that do not see each other due to the presence of the inner approximation B_i. That is, X is in convex position, and furthermore $|X| \leq \bigcirc_P$.

For $0 \leq t \leq n$, let $s(t)$ be the first iteration i, such that $\mathrm{depth}(G_i) \leq t$. Since the depth of G_i is a monotone decreasing function, this quantity is well defined. An ***epoch*** is a range of iterations between $s(t)$ and $s(t/2)$, for any parameter t. We claim that an epoch lasts $O(\bigcirc_P)$ iterations (and every iteration issues only one oracle query). Since there are only $O(\log n)$ (non-overlapping) epochs till the algorithm terminates, as the depth becomes zero, this implies the claim.

So consider such an epoch starting at $i = s(t)$. We have $m = m_i = |E(G_i)| = O(\bigcirc_P t^2)$, by Lemma 7.2, since $\bigcirc_P$ is an upper bound on the size of the largest independent set in G_i. By Lemma 7.4, as long as the depth of the intervals is at least $t/2$, the number of edges removed from the graph at each iteration, during this epoch, is at least $\Omega(t^2)$. As such, the algorithm performs at most $O(m_i/t^2) = O(\bigcirc_P)$ iterations in this epoch, till the maximum depth drops to $t/2$. QED.

Implementing the greedy algorithm With the use of dynamic segment trees [116] we show that the greedy classification algorithm can be implemented efficiently.

Lemma 7.5. *Let C be a convex body provided via a separation oracle, and let P be a set of n points in the plane. If an oracle query costs time T, then the greedy algorithm can be implemented in $O\big(n \log^2 n \log\log n + T \cdot \bigcirc_P \log n\big)$ expected time.*

Proof: The algorithm follows the proof of Theorem 7.1. We focus on efficiently implementing the algorithm once inner approximation is no longer empty. Let $U \subseteq P$ be the subset of unclassified points. By binary

searching on the vertices of the inner approximation B, we can compute the collection of visibility intervals $\mathcal{I}$ for all points in U in $O(|U| \log m) = O(n \log n)$ time (recall that $\mathcal{I}$ is a collection of circular intervals on the unit circle). We store these intervals in a dynamic segment tree T with the modification that each node v in T stores the maximum depth over all intervals contained in the subtree rooted at v. Note that T can be made fully dynamic to support updates in $O(\log n \log \log n)$ time [116].

An iteration of the greedy algorithm proceeds as follows. Start by collecting all points $U^+ \subseteq U$ realizing the maximum depth using T. When $t = |U^+|$, this step can be done in $O(\log n + t)$ time by traversing T. We compute the centerpoint of U^+ in $O(t \log t)$ expected time [33] and query the oracle using this centerpoint. Either points of U are classified (and we delete their associated intervals from T) or we improve the inner approximation. The inner approximation (which is the convex hull of query points inside the convex body C) can be maintained in an online fashion with insert time $O(\log n)$ [132, Chapter 3]. When the inner approximation expands, the points of U^+ have their intervals shrink. As such, we recompute $I(p)$ for each $p \in U^+$ and reinsert $I(p)$ into T.

As defined in the proof of Theorem 7.1, an epoch is the subset of iterations in which the maximum depth is in the range $[t/2, t]$, for some integer t. During such an epoch, we make two claims:

(i) there are $\sigma = O(n)$ updates to T, and
(ii) the greedy algorithm performs $O(n/t)$ centerpoint calculations on sets of size $O(t)$.

Both of these claims imply that a single epoch of the greedy algorithm can be implemented in expected time $O(\sigma \log n \log \log n + n \log n + T \cdot \bigcirc_P)$. As there are $O(\log n)$ epochs, the algorithm can be implemented in expected time $O(n \log^2 n \log \log n + T \cdot \bigcirc_P \log n)$.

We now prove the first claim. Recall that we have a collection of intervals $\mathcal{I}$ lying on the circle of directions. Partition the circle into k atomic arcs, where each arc contains $t/10$ endpoints of intervals in $\mathcal{I}$. Note that $k = 20n/t = O(n/t)$. For each circular arc γ, let $\mathcal{I}_\gamma \subseteq \mathcal{I}$ be the set of intervals intersecting γ. As the maximum depth is bounded by t, we have that $|\mathcal{I}_\gamma| \leq t + t/10 = 1.1t$. In particular, if $G[\mathcal{I}_\gamma]$ is the induced subgraph of the intersection graph G, then $G[\mathcal{I}_\gamma]$ has at most $\binom{|\mathcal{I}_\gamma|}{2} = O(t^2)$ edges.

In each iteration, the greedy algorithm chooses a point in an arc γ (we say that γ is *hit*) and edges are only deleted from $G[\mathcal{I}_\gamma]$. The key observation is that an arc γ can only be hit $O(1)$ times before all points of γ have depth below $t/2$, implying that it will not be hit again until the next epoch. Indeed, each time γ is hit, the number of edges in the induced subgraph $G[\mathcal{I}_\gamma]$ drops by a constant factor (Lemma 7.4). Additionally, when $G[\mathcal{I}_\gamma]$ has less than $\binom{t/2}{2}$ edges then any point on γ has depth less than $t/2$. These two facts imply that an arc is hit $O(1)$ times.

When an arc is hit, we must reinsert $|\mathcal{I}_\gamma| = O(t)$ intervals into T. In particular, over a single epoch, the total number of hits over all arcs is bounded by $O(k)$. As such, $\sigma = O(kt) = O(n)$.

For the second claim, each time an arc is hit, a single centerpoint calculation is performed. Since each arc has depth at most t and is hit a constant number of times, there are $O(k) = O(n/t)$ such centerpoint

calculations in a single epoch, each costing expected time $O(t \log t)$. QED.

Section 7.6 presents an application of the greedy classification algorithm. Namely, we present an efficient algorithm for computing the discrete geometric median of a point set (Lemma 7.23).

7.3.4 An alternative algorithm via the inference dimension

Kane et al. [95] define the notion of *inference dimension*, which in our context is the minimum number of queries needed to classify all points.

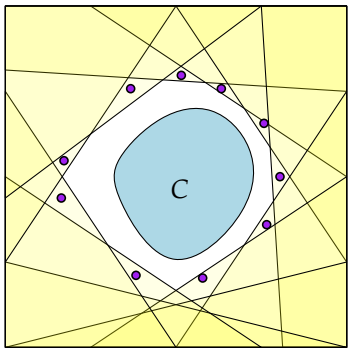


Figure 7.5: The minimal external set must be convex.

Lemma 7.6. *Let C be a convex body provided via a separation oracle, and let P be a set of n points in the plane. There is a set of $2\bigcirc_P$ oracle queries whose answers can be used to classify all points of P correctly.*

Proof: We put the at most $\bigcirc_P$ vertices of $\mathsf{conv}(P \cap C)$ into a query set. Querying these points is enough to label correctly all points inside the body C. As for the points of P outside C, let $Q \subseteq P \setminus C$ be the minimum size subset such that querying these points correctly labels all points outside C. Each point $p \in Q$ is associated with a halfspace h_p^+ that contains C. Let H be this set of halfspaces. Observe that for any point $p \in Q$, there is a point $\mathsf{witness}(p) \in P \setminus C$ for which h_p^+ does not contain $\mathsf{witness}(p)$ (as otherwise, p can be removed from Q). Let $R = \{\mathsf{witness}(p) \mid p \in Q\}$. The points of U are in the faces of the arrangement $\mathcal{A}(H)$ that are adjacent to the face $\cap_{p \in Q} h_p^+$, see Figure 7.5.

Since each point of R is separable by a line from the remaining points of R, it follows that R is convex. As such, $|Q| = |R| \leq \bigcirc_P$ which implies the result. QED.

The above lemma implies that the inference dimension of P is $2\bigcirc_P$. Plugging this into the algorithm of Kane et al. [95] results in an algorithm that labels all points correctly and performs the same number of queries as Theorem 7.1 in expectation. The advantage of Theorem 7.1 is that it does not require knowing the value of $\bigcirc_P$ in advance. However, one could perform an exponential search for a tight upper bound on $\bigcirc_P$, and still use the algorithm of Kane et al. [95]. We leave the question of experimentally comparing the two algorithms as an open problem for future research.

Sketch of the algorithm of [95] The algorithm of Kane et al. [95] specialized for our case works as follows. Start by randomly picking a sample of size $O(\bigcirc_P)$ and query the oracle with each of these points. Next, stream the unlabeled points through the computed regions, leaving only the points that are yet to be labeled. The algorithm repeats this process $O(\log n)$ times, in each iteration working on the remaining unlabeled points. By proving that in expectation at least half of the points are being labeled at each round, it follows that $O(\log n)$ iterations suffice.

7.4 THE GREEDY ALGORITHM IN THREE DIMENSIONS

Consider the 3D variant of the 2D problem: given a set of points P in $\mathbb{R}^3$ and a convex body C specified via a separation oracle, the task at hand is to classify for every point of P whether or not it is in C using the fewest number of oracle queries.

The greedy algorithm naturally extends, where at each iteration i a plane $\mathscr{e}_i$ is chosen that is tangent to the current inner approximation B_i, such that it's closed halfspace (which avoids the interior of B_i) contains the largest number of unclassified points from the set U_i. If the queried centerpoint is outside, the oracle returns a separating plane and as such points can be discarded by the **remove** operation. Similarly, if the centerpoint is reported inside, then the algorithm calls the **expand** operation and updates the 3D inner approximation B_i.

7.4.1 Analysis

Following the analysis of the greedy algorithm in 2D, we (conceptually) maintain the following set of objects: For a point $p \in U_i$, let $\mathscr{d}_i(p)$ be the set of all unit length directions $v \in \mathbb{R}^3$ such that a plane perpendicular to v separates p from B_i. Let $\mathcal{P}_i = \{\mathscr{d}_i(p) \mid p \in U_i\}$. A set of objects form a collection of ***pseudo-disks*** if the boundary of every pair of them intersect at most twice. The following claim shows that $\mathcal{P}_i$ is a collection of pseudo-disks on $\mathbb{S}$, where $\mathbb{S}$ is the sphere of radius one centered at the origin.

Lemma 7.7. *The set $\mathcal{P}_i = \{\mathscr{d}_i(p) \subseteq \mathbb{S} \mid p \in U_i\}$ is a collection of pseudo-disks.*

Proof: Fix two points $p, r \in U_i$ such that the boundaries of $\mathscr{d}_i(p)$ and $\mathscr{d}_i(r)$ intersect on $\mathbb{S}$. Let ℓ be the line in $\mathbb{R}^3$ passing through p and r. Consider any plane $\mathscr{e}$ such that ℓ lies on $\mathscr{e}$. Since ℓ is fixed, $\mathscr{e}$ has one degree of freedom. Conceptually rotate $\mathscr{e}$ until becomes tangent to B_i at point u'. The direction of the normal to this tangent plane, is a point in $X = \partial\mathscr{d}_i(p) \cap \partial\mathscr{d}_i(r)$. Note that this works also in the other direction—any point in X corresponds to a tangent plane passing through ℓ. The family of planes passing through ℓ has only two tangent planes to C. It follows that $|X| = 2$. As such, any two regions in $\mathcal{P}_i$ intersect as pseudo-disks. QED.

We need the following two classical results that follows from the Clarkson-Shor [47] technique.

Lemma 7.8. *Let $\mathcal{P}$ be a collection of n pseudo-disks, and let $V_{\leq k}(\mathcal{A})$ be the set of all vertices of depth at most k in the arrangement $\mathcal{A} = \mathcal{A}(\mathcal{P})$. Then $|V_{\leq k}(\mathcal{A})| = O(nk)$.*

Proof: Let $S \subseteq \mathcal{P}$ be a random sample where each pseudo-disk is independently placed into S with probability $1/k$. For each $p \in V_{\leq k}(\mathcal{A})$, let $\mathcal{E}_p$ be the event that p is a vertex in the union $\mathsf{U}(S)$ of this random subset of pseudo-disks. The probability that p is part of the union is at least the probability that both pseudo-disks defining p in $\mathcal{A}$ are sampled into S and the remaining $k-2$ objects containing p are not in S. Thus,

$$\mathbf{Pr}[\mathcal{E}_p] \geq \frac{1}{k^2}\left(1-\frac{1}{k}\right)^k \geq \frac{1}{e^2k^2},$$

since $1-1/x \geq e^{-2/x}$ for $x \geq 2$. If $|\mathsf{U}(S)|$ denotes the number of vertices on the boundary of the union, then linearity of expectations imply $\mathbb{E}[|\mathsf{U}(S)|] \geq |V_{\leq k}(\mathcal{A})|/(e^2k^2)$. On the other hand, it is well known the union complexity of a collection of n pseudo-disks is $O(n)$ [97]. Therefore, $\mathbb{E}[|\mathsf{U}(S)|] \leq \mathbb{E}[c|S|] \leq cn/k$, for some appropriate constant c. Putting both bounds on $\mathbb{E}[|\mathsf{U}(S)|]$ together, it follows that $cn/k \geq |V_{\leq k}(\mathcal{A})|/(e^2k^2) \iff |V_{\leq k}(\mathcal{A})| = O(nk)$. QED.

For a point p, let $\mathrm{depth}(p,\mathcal{P})$ denote the number of pseudo-disks of $\mathcal{P}$ containing p (see also Definition 7.2$_{\text{p114}}$).

Lemma 7.9. *Let $\mathcal{P}$ be a collection of n pseudo-disks. For two integers $0 < t \leq k$, a subset $X \subseteq \mathcal{P}$ is a (t,k)-**tuple** if*

(i) $|X| \leq t$,

(ii) $\exists p \in \cap_{d\in X} d$, *and*

(iii) $\mathrm{depth}(p,\mathcal{P}) \leq k$.

Let $L(t,k,n)$ be the set of all $(\leq t,k)$-tuples of $\mathcal{P}$. Then $|L(t,k,n)| = O(ntk^{t-1})$.

Proof: Let $S \subseteq \mathcal{P}$ be a random sample, where each pseudo-disk is independently placed into S with probability $1/k$. Consider a specific (t,k)-tuple X, with a witness point p of depth $\leq k$. Without loss of generality, by moving p, one can assume p is a vertex of $\mathcal{A}(\mathcal{P})$.

Let $\mathcal{E}_X$ be the event that p is of depth exactly t in $\mathcal{A}(S)$, and $X \subseteq S$. For $\mathcal{E}_X$ to occur, all the objects of X need to be sampled into S, and each of the at most $k-t$ pseudo-disks containing p in its interior are not in S. Therefore

$$\mathbf{Pr}[\mathcal{E}_X] \geq \frac{(1-1/k)^{\mathrm{depth}(p,\mathcal{P})-|X|}}{k^{|X|}} \geq \frac{(1-1/k)^k}{k^t} \geq \frac{1}{e^2k^t}.$$

Note, that a vertex of depth $\leq k$ in $\mathcal{A}(S)$ corresponds to at most one such an event happening. We thus have, by linearity of expectations, that

$$\frac{|L(t,k,n)|}{e^2k^t} \leq \mathbb{E}\left[|V_{\leq t}(\mathcal{A}(S))|\right] = O(tn/k),$$

by Lemma 7.8. QED.

Lemma 7.10. *Let $G_i = (\mathcal{P}_i, E_i)$ be the intersection graph of the pseudo-disks of $\mathcal{P}_i$ (in the ith iteration). If $\mathcal{A}(\mathcal{P}_i)$ has maximum depth k, then $|E_i| = O(nk)$. Furthermore, $\alpha(G_i) = \Omega(n/k)$, where $\alpha(G_i)$ denotes the size of the largest independent set in G_i.*

Proof: The first claim readily follows from Lemma 7.9. Indeed, $|E_i| = L(2,k,n) = O(nk)$—since every intersecting pair of pseudo-disks induces a corresponding $(2,k)$-tuple.

For the second part, Turán's Theorem states that any graph has an independent set of size at least $n/\left(d_{\text{avg}}(G_i)+1\right)$, where $d_{\text{avg}}(G_i) = 2|E_i|/n \leq ck$ is the average degree of G_i and c is some constant. It follows that $\alpha(G_i) \geq n/(ck+1) = \Omega(n/k)$. QED.

The challenge in analyzing the greedy algorithm in 3D is that mutual visibility between pairs of points is not necessarily lost as the inner approximation grows. As an alternative, consider the *hypergraph* $H_i = (\mathcal{P}_i, \mathcal{E}_i)$, where a triple of pseudo-disks $d_1, d_2, d_3 \in \mathcal{P}_i$ form a hyperedge $\{d_1, d_2, d_3\} \in \mathcal{E}_i \iff d_1 \cap d_2 \cap d_3 \neq \varnothing$ (this is equivalent to the condition that the corresponding triple of points span a triangle which does not intersect B_i).

As in the analysis of the algorithm in 2D, we first bound the number of edges in H_i and then argue that enough progress is made in each iteration.

Lemma 7.11. *Let $H_i = (\mathcal{P}_i, \mathcal{E}_i)$ be the hypergraph in iteration i, and let G_i be the corresponding intersection graph of $\mathcal{P}_i$. If $\mathcal{A}(\mathcal{P}_i)$ has maximum depth k, then $|\mathcal{E}_i| = O(\alpha(G_i)k^3)$.*

Proof: Lemma 7.10 implies that G_i has an independent set of size $\Omega(f_i/k)$, where $f_i = |\mathcal{P}_i|$. Lemma 7.9 implies that $|\mathcal{E}_i| \leq |L(3,k,f_i)| = O(f_i k^2) = O(\alpha(G_i)k^3)$. QED.

The following is a consequence of the Colorful Carathéodory Theorem [15], see Theorem 9.1.1 in [112].

Theorem 7.2. *Let P be a set of n points in $\mathbb{R}^d$ and c be the centerpoint of P. Let $S = \binom{P}{d+1}$ be the set of all $d+1$ simplices induced by P. Then for sufficiently large n, the number of simplices in S that contain c in their interior is at least $c_d n^{d+1}$, where c_d is a constant depending only on d.*

Next, we argue that in each iteration of the greedy algorithm, a constant fraction of the edges in H_i are removed. The following is the higher dimensional version of Lemma 7.3.

Lemma 7.12. *Let P be a set of n points in $\mathbb{R}^3$ lying above the xy-plane, c be the centerpoint of P and $T = \binom{P}{3}$ be the set of all triangles induced by P. Next, consider any point r on the xy-plane. Then the segment cr intersects at least $\Omega(n^3)$ triangles of T.*

Proof: Let $S = \binom{P}{d+1}$ be the set of all simplices induced by P. Theorem 7.2 implies that the centerpoint c is contained in n^4/c_1 simplices of S for some constant $c_1 > 1$. Let Δ be a simplex that contains c and observe the segment cr must intersect at least one of the triangular faces τ of Δ. As $\Delta \in S$, charge this simplex Δ to the triangular face τ. Applying this counting to all the simplices containing c, implies that at least n^4/c_1 charges are made. On the other hand, a triangle τ can be charged at most $n-3$ times (because a simplex can be formed from τ and one other additional point of P). It follows that cr intersects at least $(n^4/c_1)/(n-3) = \Omega(n^3)$ triangles of T. QED.

Lemma 7.13. *In each iteration of the greedy algorithm, the number of edges in the hypergraph $H_i = (\mathcal{P}_i, \mathcal{E}_i)$ decreases by at least $\Omega(k^3)$, where k is the maximum depth of any point in $\mathcal{A}(\mathcal{P}_i)$.*

Proof: Recall that $U^+ = U_i \cap e^+$ is the current set of unclassified points and e is the plane tangent to B_i, where e^+ is the closed halfspace that avoids the interior of B_i and contains the largest number of unlabeled points. Note that $|U^+| \geq k$.

In a **remove** operation, arguing as in Lemma 7.4, implies that the number of points of U^+ that are discarded is at least $t_i \geq k/4$. Since all of the discarded points are in a halfspace avoiding B_i, it follows that all the triples they induce are in H_i. Namely, at least $\binom{t_i}{3} = \Omega(k^3)$ hyperedges get discarded.

In an **expand** operation, the centerpoint c of U^+ is added to the current inner approximation B_i. Since all of the points of U^+ lie above the plane e, applying Lemma 7.12 on U^+ with the centerpoint c and a point lying on the plane e inside the (updated) inner approximation, we deduce that at least $\Omega(k^3)$ hyperedges are removed. QED.

Theorem 7.3. *Let $C \subseteq \mathbb{R}^3$ be a convex body provided via a separation oracle, and let P be a set of n points in $\mathbb{R}^3$. The greedy classification algorithm performs $O\big((\bigcirc_P + 1) \log n\big)$ oracle queries. The algorithm correctly identifies all points in $P \cap C$ and $P \setminus C$.*

Proof: The proof is essentially the same as Theorem 7.1. Arguing as in Lemma 7.1 implies that there are at most $O(\log n)$ iterations (and thus also oracle queries) in which the inner approximation is empty.

Now consider the hypergraph $H_1 = (\mathcal{P}_1, \mathcal{E}_1)$ at the start of the algorithm execution. As the algorithm progresses, both vertices and hyperedges are removed from the hypergraph. Let $H_i = (\mathcal{P}_i, \mathcal{E}_i)$ denote the hypergraph in the ith iteration of the algorithm. Recall that $\mathcal{P}_i$ is a set of pseudo-disks associated with each of the points yet to be classified. Observe that any independent set of pseudo-disks in the corresponding intersection graph G_i corresponds to an independent set of points with respect to the inner approximation B_i, and as such is a subset of points in convex position. Therefore, the size of any such independent set is bounded by $\bigcirc_P$.

Let k_i denote the maximum depth of any vertex in the arrangement $\mathcal{A}(\mathcal{P}_i)$. Lemma 7.11 implies that $|\mathcal{E}_i| = O\big(\bigcirc_P k_i^3\big)$. Lemma 7.13 implies that the number of hyperedges in the ith iteration decreases by at least $\Omega(k_i^3)$. Namely, after $O(\bigcirc_P)$ iterations, the maximum depth is halved. It follows that after $O(\bigcirc_P \log n)$ iterations, the maximum depth is zero, which implies that all the points are classified. Since the algorithm performs one query per iteration, the claim follows. QED.

7.5 AN INSTANCE-OPTIMAL APPROXIMATION IN TWO DIMENSIONS

Before discussing the improved algorithm, we present a lower bound on the number of oracle queries performed by any algorithm that classifies all the given points. We then present the improved algorithm, which matches the lower bound up to a factor of $O(\log^2 n)$.

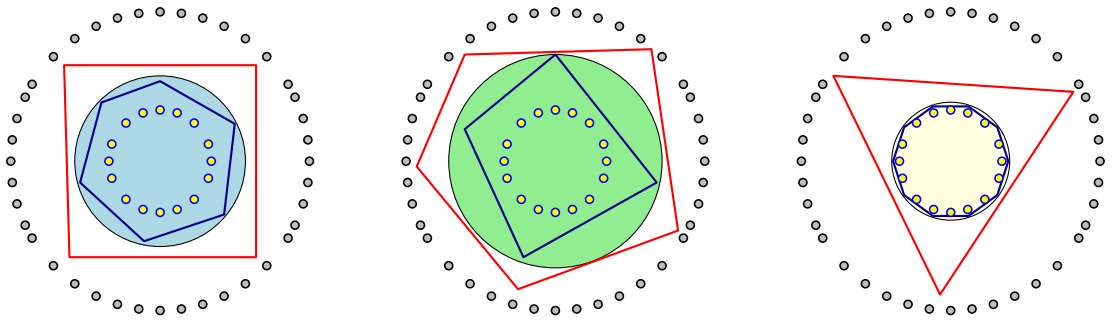


Figure 7.6: The separation price, for the same point set, is different depending on how "tight" the body is in relation to the inner and outer point set.

7.5.1 A lower bound

Given a set P of points in the plane, and a convex body C, the ***outer fence*** of P is a closed convex polygon F_{out} with minimum number of vertices, such that $C \subseteq F_{\text{out}}$ and $C \cap P = F_{\text{out}} \cap P$. Similarly, the ***inner fence*** is a closed convex polygon F_{in} with minimum number of vertices, such that $F_{\text{in}} \subseteq C$ and $C \cap P = F_{\text{in}} \cap P$. Intuitively, the outer fence separates $P \setminus C$ from ∂C, while the inner fence separates $P \cap C$ from ∂C. The ***separation price*** of P and C is

$$\circledcirc(P, C) = |F_{\text{in}}| + |F_{\text{out}}|,$$

where $|F|$ denotes the number of vertices of a polygon F. See Figure 7.6 for an example.

Lemma 7.14. *Let C be a convex body provided via a separation oracle, and let P be a point set in the plane. Any algorithm that classifies the points of P in relation to C, must perform at least $\circledcirc(P, C)$ separation oracle queries.*

Proof: Consider the set Q of queries performed by the optimal algorithm for the fixed input P and C. Partition Q into the points inside and outside C. The set of points inside, $Q_{\text{in}} = Q \cap C$ has the property that $Q_{\text{in}} \subseteq C$ and $\mathsf{conv}(Q_{\text{in}}) \cap P = C \cap P$—otherwise, there would be a point of $C \cap P$ that is not classified. Namely, the vertices of $\mathsf{conv}(Q_{\text{in}})$ are vertices of a fence that separates the points of P inside C from the boundary of C. As such, $|Q_{\text{in}}| \geq |\mathsf{conv}(Q_{\text{in}})| \geq |F_{\text{in}}|$.

Similarly, each query in $Q_{\text{out}} = Q \setminus Q_{\text{in}}$ gives rise to a separating halfplane. The intersection of the corresponding halfplanes is a convex polygon H that contains C, and furthermore contains no point of $P \setminus C$. Namely, the boundary of H behaves like an outer fence. Hence, $|Q_{\text{out}}| \geq |H| \geq |F_{\text{out}}|$.

Combining both inequalities, we obtain $|Q| = |Q_{\text{in}}| + |Q_{\text{out}}| \geq |F_{\text{in}}| + |F_{\text{out}}| = \circledcirc(P, C)$, as claimed. QED.

Remarks. (i) Naturally the separation price, and thus the proof of the lower bound, generalizes to higher dimensions [84]. (ii) The lower bound only holds for $d \geq 2$. In 1D, the problem can be solved using $O(\log n)$ queries with binary search. The above would predict that any algorithm needs $\Omega(1)$ queries. However it is not hard to argue a stronger lower bound of $\Omega(\log n)$. (iii) In [84], we show that when P is a set of n points

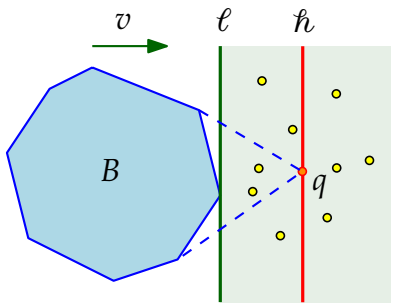


Figure 7.7: A directional climb. An iteration is done using the line ℓ. After updating B to include the query q, the algorithm chooses a new extreme line $\hbar$ tangent to B in the direction of v.

chosen uniformly at random from a square and C is a smooth convex body, $\mathbb{E}[\circledcirc(P, C)] = O(n^{1/3})$. Thus, when the points are randomly chosen, one can think of $\circledcirc(P, C)$ as growing sublinearly in n.

7.5.2 Useful operations

We start by presenting some basic operations that the new algorithm will use.

A directional climb Given a direction v, a ***directional climb*** is a sequence of iterations where in each iteration the algorithm finds the extreme line ℓ perpendicular to v that is tangent to the inner approximation B. The algorithm then performs an iteration with ℓ, as described in Section 7.3.2. Specifically, the algorithm computes the centerpoint z of all points in the halfspace bounded by ℓ that avoids C. Depending on whether $z \in C$, we either perform an **expand** or **remove** operation (see Section 7.3.2). We then classify points accordingly and recompute ℓ with the updated inner approximation B. See Figure 7.7 for an illustration. The directional climb ends when the outer halfspace induced by this line contains no unclassified point.

Lemma 7.15. *A directional climb requires $O(\log n)$ oracle queries.*

Proof: Consider the tangent to B in the direction of v. At each iteration, we claim the number of points in this halfplane is reduced by a factor of $1/3$. Indeed, if the query (i.e., centerpoint) is outside C then at least a third of these points got classified as being outside. Alternatively, the tangent halfplanes moves in the direction of v, since the query point is inside C. But then the new halfspace contains at most $2/3$ fraction of the previous point set—again, by the centerpoint property. QED.

Line cleaning A ***pocket*** is a connected region of $\mathsf{conv}(U \cup B) \setminus B$, see Figure 7.8. For the set P of input points, consider the set of all lines

$$L(P) = \{\mathrm{line}(p, r) \mid p, r \in P\} \tag{7.1}$$

they span.

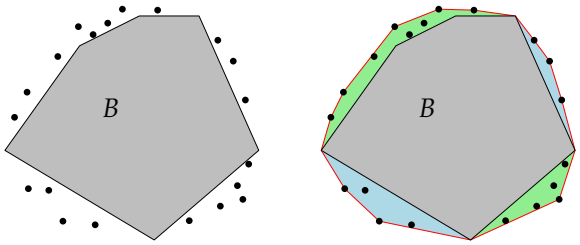


Figure 7.8: Unclassified points and their pockets.

Let ℓ be a line that splits a pocket Υ into two regions, and furthermore, it intersects B. Let $I = \ell \cap \Upsilon$, and consider all the intersection points of interest along I in this pocket. That is,

$$\Xi(\Upsilon, \ell, P) = I \cap L(P) = \big\{(\Upsilon \cap \ell) \cap \hbar \mid \hbar \in L(P)\big\}.$$

In words, we take all the pairs of points of P (each such pair induces a line) and we compute the intersection points of these lines with the interval I of interest. Ordering the points of this set along ℓ, a prefix of them is in C, while the corresponding suffix are all outside C. One can easily compute this prefix/suffix by doing a binary search, using the separation oracle for C—see the lemma below for details. Each answer received from the oracle is used to update the point set, using **expand** or **remove** operations. We refer to this operation along ℓ as ***cleaning*** the line ℓ. See Figure 7.9.

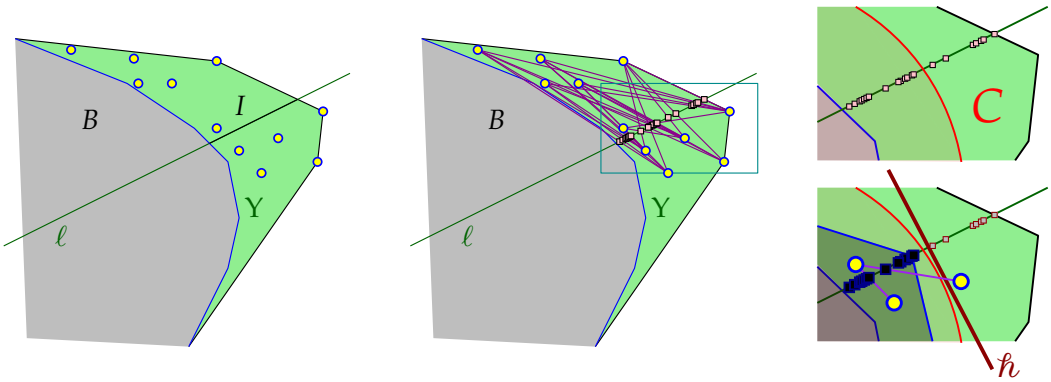


Figure 7.9: Line cleaning. All the intersection points of interest along ℓ are classified. The binary search results in the oracle returning a line $\hbar$ that separates the points outside from the points inside.

Lemma 7.16. *Given a pocket Υ, and a splitting line ℓ, one can clean the line ℓ—that is, classify all the points of $\Xi = \Xi(\Upsilon, \ell, P)$ using $O(\log n)$ oracle queries. By the end of this process, Υ is replaced by two pockets, Υ_1 and Υ_2 that do not intersect ℓ. The pockets Υ_1 or Υ_2 may be empty sets.*

Proof: First, we describe the line cleaning procedure in more detail. The algorithm maintains, in the beginning of the ith iteration, an interval J_i on the line ℓ containing all the points of Ξ that are not classified yet. Initially, $J_1 = \Upsilon \cap \ell$. One endpoint, say $p_i \in J_i$ is on ∂B_i, and the other, say p'_i, is outside C, where B_i is the inner

approximation in the beginning of the ith iteration.

In the ith iteration, the algorithm computes the set $\Xi_i = J_i \cap \Xi$. If this set is empty, then the algorithm is done. Otherwise, it picks the median point u_i, in the order along ℓ in Ξ_i, and queries the oracle with u_i. There are two possibilities:

(A) If $u_i \in C$ then the algorithm sets $\Xi_{i+1} = \Xi_i \setminus [p_i, u_i)$, and $J_{i+1} = J_i \setminus [p_i, u_i)$.

(B) If $u_i \notin C$, then the oracle provided a closed halfspace h^+ that contains C. Let h^- be the complement open halfspace that contains u_i. The algorithm sets $\Xi_{i+1} = \Xi_i \setminus h^-$ and $J_{i+1} = J_i \cap h^+$.

This resolves the status of at least half the points in Ξ_i, and shrinks the active interval. The algorithm repeats this till Ξ_i becomes empty. Since $|\Xi| = O(n^2)$, this readily implies that the algorithm performs $O(\log n)$ iterations.

We now argue that the pocket is split—that is, Y_1 and Y_2 do not intersect ℓ. Assume that it is false, and let B' be the inner approximation after this procedure is done. Let L (resp. R) be the points of $U_{\mathrm{Y}} = U \cap \mathrm{Y}$ that are unclassified on one side (resp. other side) of ℓ. If the pocket is not split, then there are two points $p \in L$ and $r \in R$, such that $pr \cap B' = \emptyset$, and $\partial\mathsf{conv}(B' \cup L \cup R)$ intersects ℓ at the point $u = pr \cap \ell$. However, by construction, the point $u \in \Xi$. As such, the point u is now classified as either being inside or outside C, as it is a point in Ξ. If u is outside, then the halfplane h^- that classified it as such, must had classified either p or r as being outside C, which is a contradiction. The other option is that u is classified as being inside, but then it is in B' which is again a contradiction, as it implies that B' intersects the segment pr. QED.

Vertical pocket splitting Consider a pocket Y such that all of its points lie vertically above B, and the bottom of Y is part of a segment of ∂B, see Figure 7.10. Such a pocket can be viewed as being defined by an interval on the x-axis corresponding to its two vertical walls. Let U_{Y} be the set of unclassified points in this pocket. In each iteration, the algorithm computes the centerpoint z of U_{Y}, and queries the separation oracle for the label of z. As long as the query point is outside C, the algorithm performs a **remove** operation using the returned separating line.

When the oracle returns that the query point z is inside C, the algorithm computes the vertical line ℓ_z through z. The algorithm now performs line cleaning on this vertical line. This operation splits Y into two sub-pockets. Crucially, since z was a centerpoint for U_{Y}, the number of points in each of the two sub-pockets is at most $2\left|U_{\mathrm{Y}}\right|/3$. See Figure 7.10.

7.5.3 The algorithm

The algorithm starts in the same way as the greedy algorithm of Section 7.3.2, which we restate for convenience. Recall that U is the set of unclassified points (initially $U = P$). At all times, the algorithm maintains the inner approximation $B \subseteq C$. At the beginning, B is uninitialized. The algorithm computes the centerpoint z of U and queries the oracle for the label of z. While z is outside, we classify the appropriate set

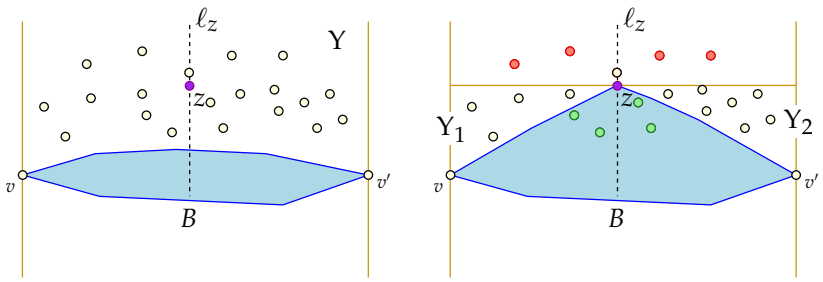


Figure 7.10: Vertical pocket splitting. In this example, the centerpoint z lies inside C. Thus we construct the vertical line ℓ_z through z (left). Next, we perform a line cleaning operation on ℓ_z. This splits the original pocket Y into two new pockets Y_1, Y_2, while classifying some points in the process (right). Observe that the unclassified points in Y_1 and Y_2 are no longer mutually visible to each other after the line cleaning operation.

of points as outside (according to the separating hyperplane returned from the oracle), update U, and repeat. As soon as the computed centerpoint z lies in C, we set $B = z$ and continue to the stage phase.

Next, the algorithm performs two directional climbs (Lemma 7.15) in the positive and negative directions of the x-axis. This uses $O(\log n)$ oracle queries by Lemma 7.15 and results in a computed segment $vv' \subseteq C$, where vv' are vertices of the inner approximation B, such that all unclassified points lie in the strip induced by the vertical line through v and the vertical line through v', see also Figure 7.10.

The algorithm now handles all points of U lying above vv' (the points below the line are handled in a similar fashion). Let B^+ be the set of vertices of B in the top chain. Note that B^+ consists of at most $O(\log n)$ vertices. For each vertex v of B^+, the algorithm performs line cleaning on the vertical line going through v. This results in $O(\log n)$ vertical pockets, where all vertical lines passing originally through B^+ are now clean.

The algorithm repeatedly picks a vertical pocket. If the pocket contains less than three points the algorithm queries the oracle for the classification of these points, and continues to the next pocket. Otherwise, the algorithm performs a vertical pocket splitting operation, as described above. The algorithm stops when there are no longer any pockets (i.e., all the points above the segment vv' are classified). The algorithm then runs the symmetric procedure below this segment vv'.

7.5.4 Analysis

Lemma 7.17. *Given a point set P, and a convex polygon σ that is an inner fence for $P \cap C$; that is, $P \cap C \subseteq \sigma \subseteq C$. Then, there is a convex polygon π, such that*

(A) $P \cap C \subseteq \pi \subseteq \sigma$.

(B) $|\pi| \leq 2\,|\sigma|$ *(where $|Q|$ denotes the number of vertices of the polygon Q).*

(C) Every edge of π lies on a line of $L(P)$, see Eq. (7.1).

Proof: Any edge e of σ that does not contain any point of P on it can be moved parallel to itself into the polygon until it passes through a point of P. Next, split the edges that contain only a single point of P, by adding this point as a vertex.

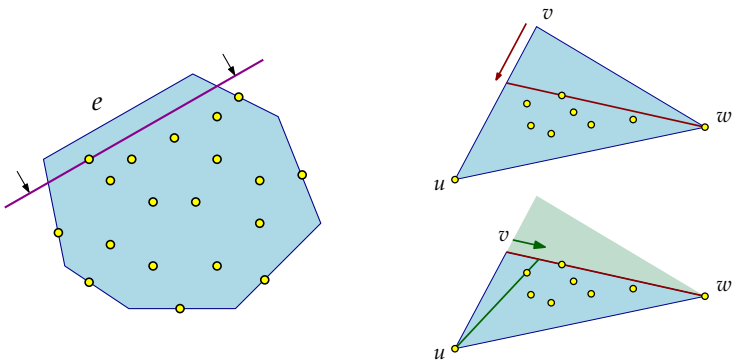


Figure 7.11: Constructing the polygon π from an inner fence σ.

Consider a vertex v of the polygon that is not in P— and consider the two adjacent vertices u, w, which must be in P. If $\triangle uvw \setminus uw$ contains no point of P, then we delete v from the polygon and replace it by the edge uw. Otherwise, move v towards u, until the edge vw hits a point of P. Next, move v towards w, till the edge vu hits a point of P. See Figure 7.11.

Repeating this process so that all edges contain two points of P means that properties (A) and (C) are met. Additionally, the number of edges of the new polygon π is at most twice the number of edges of σ, implying property (B). QED.

Consider the inner and outer fences F_{in} and F_{out} of P in relation to C. Applying Lemma 7.17 to F_{in}, results in a convex polygon π that separates $P \cap C$ from ∂C, that has at most $2\,|F_{\text{in}}|$ vertices. Let V be the set of all vertices of the polygons $F_{\text{in}}, F_{\text{out}}$ and π.

The following two Lemmas state that if a vertical pocket Y containing no vertex of V, then all points in Y can be classified using $O(\log n)$ oracle queries. Finally, we analyze the scenario when Y contains at least one vertex of V.

Lemma 7.18. *Let* Y *be a vertical pocket created during the algorithm with current inner approximation B. Suppose that $V \cap \mathrm{Y} = \varnothing$, then all points in $P \cap \mathrm{Y}$ are outside C.*

Proof: Assume without loss of generality that Y lies above B. Let $U = P \cap \mathrm{Y}$ be the set of unclassified points in the pocket. Note that Y is bounded by two vertical lines that were previously cleaned.

By assumption, Y does not contain any vertex of π. It follows that there is a single edge of π that intersects the two vertical lines bounding Y. Let u_L, u_R be these two intersection points, one lying on each line. By definition, we have $u_L, u_R \in C$. Furthermore, u_L, u_R lie on lines of $L(P)$ by construction of π. Since both vertical lines bounding Y were cleaned, it must be that the segment $u_L u_R \subseteq B$. Since all points of U are above B, this implies that U lies above $u_L u_R$ and thus above π. Namely, all points of U are outside C. QED.

Lemma 7.19. *Let* Y *be a vertical pocket with $V \cap \mathrm{Y} = \varnothing$. Then during the vertical pocket splitting operation applied to* Y*, all oracle queries are outside C. In particular, all points of $P \cap \mathrm{Y}$ are classified after $O(\log n)$ oracle queries.*

Proof: Let $U = P \cap \mathrm{Y}$. By Lemma 7.18, all points of U lie outside C. Assume that the first statement of the Lemma is false, and let $U' \subseteq U$ be the set of unclassified points such that z was the centerpoint for U' and $z \in C$. Now z is inside a triangle induced by three points of U'. Namely, there are (at least) two points outside C in this pocket that are not mutually visible to each other with respect to C. But this implies that F_{out} must have a vertex somewhere inside the vertical pocket Y, which is a contradiction.

Hence, all oracle queries made by the algorithm are outside C. Each such query results in a constant reduction in the size of U, since the query point is a centerpoint of the unclassified points. It follows that after $O(\log |U|) = O(\log n)$ queries, all points in Y are classified. QED.

Theorem 7.4. *Let C be a convex body provided via a separation oracle, and let P be a set of n points in the plane. The improved classification algorithm performs $O\Big(\big[1 + \circledcirc(P, C)\big] \log^2 n\Big)$ oracle queries. The algorithm correctly identifies all points in $P \cap C$ and $P \setminus C$.*

Proof: The initial stage involves two directional climbs and $O(\log n)$ line cleaning operations, and thus requires $O(\log^2 n)$ queries.

A vertical pocket that contains a vertex of V is charged arbitrarily to any such vertex. Since the number of points in a pocket reduces by at least a factor of $1/3$ during a split operation, this means that a vertex of V is charged at most $O(\log n)$ times. Each time a vertex gets charged, it has to pay for the $O(\log n)$ oracle queries that were issued in the process of creating this pocket, and later on for the price of splitting it. Thus, we only have to account for queries performed in vertical pockets that do not contain a vertex of V. By Lemma 7.19, such a pocket will have all points inside it classified after $O(\log n)$ oracle queries.

However, the above implies that there are at most $O([1 + \circledcirc(P, C)] \log n)$ vertical pockets with no vertex of V throughout the algorithm execution. Since handling such a pocket requires $O(\log n)$ queries, the bound follows. QED.

7.6 LOWER BOUNDING A CONVEX FUNCTION, AGAIN

In this section we revisit a problem studied in Chapter 3. Namely, the problem of minimizing a convex function given oracle access to its gradient computation, see Section 3.3$_{\text{p38}}$.

Suppose we are given a set of n points P in the plane and a convex function $f : \mathbb{R}^2 \to \mathbb{R}$. Our goal is to compute the point in P minimizing $\min_{p \in P} f(p)$. Given a point $p \in \mathbb{R}^2$, assuming that we can evaluate f and the derivative of f at p efficiently, we show that the point in P minimizing f can be computed using $O(\circledcirc_p \log^2 n)$ evaluations to f or its derivative.

Recall the level set (Definition 3.3$_{\text{p38}}$) and subgradient (Definition 3.4$_{\text{p38}}$) of a function f. Let $\alpha = \min_{p \in P} f(p)$. We have that $\mathcal{L}_f(\alpha) \cap P = \{p \in P \mid f(p) = \alpha\}$ and $\mathcal{L}_f(\alpha') \cap P = \varnothing$ for all $\alpha' < \alpha$. Hence, the problem is reduced to determining the smallest value r such that $\mathcal{L}_f(r) \cap P$ is non-empty.

Lemma 7.20. *Let P be a collection of n points in the plane. For a given value r, let $C_r = \mathcal{L}_f(r)$. The set $C_r \cap P$ can be computed using $O(\bigcirc_P \log n)$ evaluations to f or its derivative. If T is the time needed to evaluate f or its derivative, the algorithm can be implemented in $O(n \log^2 n \log\log n + T \cdot \bigcirc_P \log n)$ expected time.*

Proof: The Lemma follows by applying Theorem 7.1. Indeed, let $C_r = \mathcal{L}_f(r)$ be the convex body of interest. It remains to design a separation oracle for C_r.

Given a query point $z \in \mathbb{R}^2$, first compute $c = f(z)$. If $c \leq r$, then report that $z \in C_r$. Otherwise, $c > r$. In this case, compute some gradient vector v in $\partial f(z)$. Using the vector v, we can obtain a line ℓ tangent to the boundary of $\mathcal{L}_f(c)$ at z. As $\mathcal{L}_f(r) \subseteq \mathcal{L}_f(c)$, ℓ is a separating line for z and C_r, as desired. As such, the number of separation oracle queries needed to determine $C_r \cap P$ is bounded by $O(\bigcirc_P \log n)$ by Theorem 7.1.

The implementation details of Theorem 7.1 are given in Lemma 7.5. QED.

The algorithm Let $\alpha = \min_{p \in P} f(p)$. For a given number $r \geq 0$, set $P_r = \mathcal{L}_f(r) \cap P$. We develop a randomized algorithm to compute α.

Set $P_0 = P$. In the ith iteration, the algorithm chooses a random point $p_i \in P_{i-1}$ and computes $r_i = f(p_i)$. Next, we determine P_{r_i} using Lemma 7.20. In doing so, we modify the separation oracle of Lemma 7.20 to store the collection of queries $S_i \subseteq P$ that satisfy $f(s) = r_i$ for all $s \in S_i$. We set $P_{i+1} = P_{r_i} \setminus S_i$. Observe that all points $p \in P_{i+1}$ have $f(p) < r_i$. The algorithm continues in this fashion until we reach an iteration j in which $\left|P_{j+1}\right| \leq 1$. If $P_{j+1} = \{q\}$ for some $q \in P$, output q as the desired point minimizing f. Otherwise $P_{j+1} = \varnothing$, implying that $P_{r_j} = S_j$, and the algorithm outputs any point in the set S_j.

Analysis We analyze the running time of the algorithm. To do so, we argue that the algorithm invokes the algorithm in Lemma 7.20 only a logarithmic number of times.

Lemma 7.21. *In expectation, the above algorithm terminates after $O(\log n)$ iterations.*

Proof: Let $V = \{f(p) \mid p \in P\}$ and $N = |V|$. For a number r, define $V_r = \{i \in V \mid i \leq r\}$. Notice that we can reinterpret the algorithm described above as the following random process. Initially set $r_0 = \max_{i \in V} i$. In the ith iteration, choose a random number $r_i \in V_{r_{i-1}}$. This process continues until we reach an iteration j in which $\left|V_{r_j}\right| \leq 1$.

We can assume without loss of generality that $V = \{1, 2, \ldots, N\}$. For an integer $i \leq N$, let $T(i)$ be the expected number of iterations needed for the random process to terminate on the set $\{1, \ldots, i\}$. We have that $T(i) = 1 + \frac{1}{i-1} \sum_{j=1}^{i-1} T(i-j)$, with $T(1) = 0$. This recurrence solves to $T(i) = O(\log i)$. As such, the algorithm repeats this random process $O(\log N) = O(\log n)$ times in expectation. QED.

Lemma 7.22. *Let P be a set of n points in $\mathbb{R}^2$ and let $f : \mathbb{R}^2 \to \mathbb{R}$ be a convex function. The point in P minimizing f can be computed using $O(\bigcirc_P \log^2 n)$ evaluations to f or its derivative. The bound on the number of evaluations holds in expectation. If T is the time needed to evaluate f or its derivative, the algorithm can be implemented in $O(n \log^3 n \log\log n + T \cdot \bigcirc_P \log^2 n)$ expected time.*

Proof: The result follows by combining Lemma 7.20 and Lemma 7.21. QED.

7.6.1 The discrete geometric median

Let P be a set of n points in $\mathbb{R}^d$. For all $x \in \mathbb{R}^d$, define the function $f(x) = \sum_{q \in P - x} \|x - q\|$. The ***discrete geometric median*** is defined as the point in P minimizing the quantity $\min_{p \in P} f(p)$.

Note that f is convex, as it is the sum of convex functions. Furthermore, given a point p, we can compute $f(p)$ and the derivative of f at p in $O(n)$ time. As such, by Lemma 7.22, we obtain the following.

Lemma 7.23. *Let P be a set of points in $\mathbb{R}^2$. Then the discrete geometric median of P can be computed in $O(n \log^2 n \cdot (\log n \log\log n + \bigcirc_P))$ expected time.*

Remark 7.1. For a set of n points P chosen uniformly at random from the unit square, it is known that in expectation $\bigcirc_P = \Theta(n^{1/3})$ [8]. As such, the discrete geometric median for such a random set P can be computed in $O(n^{4/3} \log^2 n)$ expected time.

Part IV

Conclusions and open problems

8 Conclusion

> *Do not ask whether a statement is true until you know what it means.*
>
> **— Errett Bishop**

We conclude this thesis with some discussion of future work and problems left open from each chapter.

8.1 GEOMETRIC ORDERS

In Chapter 2, we showed that any bounded subset of $\mathbb{R}^d$ has a collection of "few" orderings which captures proximity. This readily led to simplified and improved approximate dynamic data structures for many fundamental proximity-based problems in computational geometry. Beyond these improvements, we believe that the new technique could potentially be simple enough to be useful in practice, and could be easily taught in an undergraduate level class (replacing, for example, well-separated pair decomposition—a topic that is not as easily accessible).

We expect other applications to follow from the technique presented. For example, recently Buchin et al. [27] presented a near linear-sized construction for robust spanners. The idea is to build a robust spanner in one dimension, and then obtain a robust spanner in higher dimensions by applying the one-dimensional construction using the locality-sensitive orderings.

In terms of open problems, it would be interesting to determine the minimum collection of orders needed that still obey the locality-sensitive ordering properties.

Open Problem 8.1. *Does there exist a collection of locality-sensitive orderings of size $O_d(1/\varepsilon^d)$?*

For general metric spaces with doubling dimension λ, Filtser and Le [66] recently showed there exist locality-sensitive orderings of size $O(1/\varepsilon^{O(\lambda)})$. However, an initial estimate of the hidden constants inside the exponent $O(\lambda)$ seem to be quite large, and it is an open problem to improve these hidden constants.

8.2 GEOMETRIC CENTERS

In Chapter 3 we gave an improved randomized algorithm for computing an approximate centerpoint of a point set. Namely, we showed that a centerpoint of quality arbitrarily close to $\Omega(1/d^2)$ can be computed in

time $\widetilde{O}(d^7)$. The main open problem here is to develop any polynomial time algorithm which computes a better quality centerpoint. As far as the author is aware, obtaining an algorithm computing a $\Omega(1/d^{1.99})$-centerpoint in $\text{poly}(n, d)$ time is open. All current randomized approaches are based on the use of Radon points (see Definition 3.2$_{\text{p30}}$). We believe that a new idea is needed to obtain an efficient algorithm for computing a better quality centerpoint.

There are many questions one can ask about centerpoints. One particular question of interest is: a more efficient algorithm for computing the centerpoint on a grid of points. As we know by now, for general point sets, a $\Omega(1/(d+2)^2)$-centerpoint can be computed in $\widetilde{O}(d^7)$ time. For $d \geq 2$, exact centerpoints can be computed in $O(n^{d-1} + n \log n)$ expected time [33].

Open Problem 8.2. *Let $P \subseteq [\![U]\!]^d$ for some integer U, where $[\![U]\!] = \{0, \ldots, U-1\}$. Can anything better be achieved in this restricted setting, even for $d = 3$? Anything polynomial in n and U?*

Chapter 4 presented some variations on weak ε-nets. Of particular interest to the author is the notion of center nets, and whether there exist any other interesting constructions. Recall that Theorem 4.5 showed the existence of (ε, α)-center nets, where $\alpha \approx 1/(d \log(1/\varepsilon))$ and has size $\widetilde{O}((d^2/\varepsilon)^{d^2})$. However, observe that α depends on ε in the construction. Can we eliminate this dependency?

Open Problem 8.3. *Can one obtain a construction of (ε, α)-center nets when ε and α are both given as parameters?*

Naturally, as α grows, we would expect the size of the net to shrink.

The main open problem left in Section 4.5 is bounding the size of (k, ε)-nets in the general case. That is, the input is a set P of n points in $\mathbb{R}^d$, and we would like to compute a minimum set of k-flats which stab all convex bodies containing at least εn points of P. As noted in Section 4.1, there is a (k, ε)-net of asymptotically the same size as of a weak ε-net in $\mathbb{R}^{d-k}$. This follows by projecting the point set to a subspace of dimension $d-k$, constructing a regular weak ε-net, and lifting the net back to the original space. Can one do better than this somewhat naive construction?

Note that it is easy to show a lower bound of size $\Omega(1/\varepsilon)$ for $(1, \varepsilon)$-nets in the general case. Take a point set that consists of $\lceil 2/\varepsilon \rceil$ equally sized clusters of tightly packed points, such that no line passes through three clusters. Namely, our sublinear results in $1/\varepsilon$ are special for the uniform measure on the hypercube.

In Chapter 5, we showed that the yolk of a set of n points in $\mathbb{R}^d$ can be computed in expected time $O_d(n^{d-1} \log n)$. The natural open problem is to improve the running times for computing the yolk (and extremal yolk) even further. It seems believable, that for $d > 3$, the log factors in Theorem 5.1 and Theorem 5.2 might not be necessary. We leave this as an open problem for further research. Another potential direction for further research is to develop efficient algorithms for approximating the radius the yolk in higher dimensions. For example, in three dimensions we showed that the yolk can be computed exactly in $O(n^2)$ expected time, see Remark 5.1$_{\text{p75}}$. Can we obtain a near-linear time approximation algorithm?

Open Problem 8.4 (Approximating the yolk). *Does there exist an* FPTAS[1] *for approximating the radius of the yolk of n points in $\mathbb{R}^3$ such that the dependency on n in the running time is linear (or subquadratic)?*

In addition to the yolk, there are other variations of voting games which may be of interest to the community. A different generalization of the plurality point (the yolk is only one such generalization) is the finagle of a point set [154]. For a point set $P \subset \mathbb{R}^2$, the finagle ball is the ball B of smallest radius such that for any challenger $q \in \mathbb{R}^2$, there is a response $p \in B$ such that p beats q.

Open Problem 8.5 (Computing the finagle). *How quickly can the finagle ball be computed or approximated?*

We are not aware of any previous work studying the computational aspects of the finagle ball.

Recall that p beats q if there are more points of P closer to p than q. Less formally, q may initially beat the center c of B, but c is allowed to "finagle" a better response in a second round of interaction to obtain p, which beats q.

A slightly easier problem might be the following.

Open Problem 8.6 (The pseudo-finagle). *Compute the smallest ball B (which we call the* pseudo-finagle*) such that for any $q \in \mathbb{R}^2$, more than half of the points of P are closer to B than q. This removes the one round of interaction between the challenger and responder.*

Another generalization of the plurality point studied recently by Aronov et al. is the ***β-plurality point*** [11]. Here, the authors study a model in which given an existing policy $p \in \mathbb{R}^2$, a voter $v \in \mathbb{R}^2$ will swap to a competing policy $q \in \mathbb{R}^2$ if and only if $\|q - v\| \leq \beta\|p - v\|$, for some $\beta \in (0, 1]$. As such, a voter v swaps to a different policy only if they have a strong incentive to do so. For a given $\beta \in (0, 1]$ and set of n voters P, a point p is a β-plurality point if for all challengers $q \in \mathbb{R}^2$, the number of voters in P who swap to the policy q is less than $n/2$.

Naturally, one is interested in how quickly such a β-plurality point for P can be computed (if it exists), given β. Additionally, one can also ask to compute the largest value $\beta^\star(P)$ such that P admits a $\beta^\star(P)$-plurality point. Aronov et al. [11] give an $O_d(n^{d^2})$ time algorithm for computing $\beta^\star(P)$. It is possible that the running time of this exact algorithm can be improved. In addition, the authors of [11] develop an approximation algorithm, which returns a value β such that $\beta \geq (1 - \varepsilon)\beta^\star(P)$ and runs in time $\widetilde{O}_d(n^2/\varepsilon^{O(d)})$ (where $\widetilde{O}$ hides $\text{polylog}(n, 1/\varepsilon)$ factors).

There are many other computationally interesting voting games. For example, the plurality point is simply a point which beats all other policies in the policy space. When no plurality point exists, the yolk and finagle can be interpreted as capturing a collection of points which beat many—but not all—policies. Rather than considering such objects of constant description complexity, another possible generalization is to consider a finite set of points which collectively beat all other policies.

[1] An optimization problem admits an FPTAS (fully-polynomial time approximation scheme) if there exists an algorithm which $(1 + \varepsilon)$-approximates the problem and has running time $\text{poly}(n, 1/\varepsilon)$.

Open Problem 8.7 (One policy always wins enough). *Let $\alpha = 1/2 - \varepsilon$. Suppose we want to choose $k := k(\varepsilon)$ policies Q such that for any $p \in \mathbb{R}^2$, there exists a $q \in Q$ such that at least αn voters of P prefer q to p. Note that when $\varepsilon = 1/6$, $k = 1$ by choosing the centerpoint for P. What is the right choice of k, depending on ε? Can these policies Q be constructed efficiently?*

We are not aware of any previous work on the precise formulation of the problem defined above.

8.3 GEOMETRIC SEPARATION

In Chapter 6, we showed that a set of n points drawn uniformly at random from the unit square can be separated by $O(n^{2/3})$ lines, and require $\Omega(n^{2/3} \log\log n / \log n)$ separating lines. The key open problem suggested by our work is as follows.

Open Problem 8.8. *Can the lower bound of Theorem 6.3 be improved so that it matches the upper bound $O(n^{2/3})$ up to a constant?*

In Section 6.5 we developed an efficient randomized algorithm for approximating the separability of a point set. Now, consider the bichromatic version of the problem: given a set of red points R and blue points B in the plane, define $\mathrm{sep}(R, B)$ as the minimum number of lines needed to separate all red-blue pairs of points. As mentioned in Remark 6.1, the separability of a monochromatic point set of size n is $\Omega(\sqrt{n})$ in the plane. However in the bichromatic setting, there exist configurations of red and blue points with $\mathrm{sep}(R, B) = 1$. While our current algorithm should extend to the bichromatic case, the algorithm itself is not any faster and provides the same guarantees as for the monochromatic case. For this reason, one possible direction for further research is to study the bichromatic problem, and obtain approximation algorithms which are much more efficient than our proposed algorithm when $\mathrm{sep}(R, B)$ is sufficiently smaller than $\sqrt{n}$.

In Chapter 7 we presented various algorithms for classifying points with oracle access to an unknown convex body. As far as the author is aware, this problem has not been studied within the community previously. However we believe that this is an interesting and natural problem. We now pose some open problems.

The first question to ask is if the algorithm presented in Theorem 7.4, which makes $O(\circledcirc(P, C) \log^2 n)$ queries, can be reduced to $O(\circledcirc(P, C) \log n)$. This seems surprisingly challenging. In particular, the algorithm of Theorem 7.4 is not natural, and the analysis is somewhat complicated. It is an open problem to simplify the algorithm and its analysis. Alternatively, one can hope to develop algorithms in which the number of queries is parameterized by different functions of the input instance.

Additionally, while we get some results in 3D, we do not have an algorithm for which the number of queries matches the lower bound (i.e., separation price) up to $\mathrm{polylog}(n)$ factors. One can generalize the notion of separation price to higher dimensions, but the best solution we have in 3D so far is the greedy approach (and in general, $\bigcirc_P$ can be much larger than $\circledcirc(P, C)$).

We currently have no interesting algorithms for this problem for $d \geq 4$. The greedy algorithm naturally extends, however the number of queries made would most likely involve factors of the form $\bigcirc_P{}^{O(d)}$, which is only interesting when $\bigcirc_P \ll n^{1/d}$. It would be interesting to try and develop efficient algorithms in higher dimensions.

Another interesting direction for research is the *meta* problem of finding geometric problems and oracles (or computational models) to work with. Studying various geometric problems assuming access to a separation oracle, as done in Chapter 7, is only one such possible research direction. One can also consider the problem of learning non-convex bodies, and what type of oracles would be interesting from an algorithmic perspective.

Open Problem 8.9 (Learning non-convex bodies). *Do there exist active-learning algorithms for non-convex bodies under a suitable computational model?*

Clearly, if the body is non-convex, the separation oracle is no longer properly defined. Here is one proposed model: If a query point is inside the body, then the oracle returns the radius of the largest disk contained inside the body centered at q. Otherwise, it returns the radius of the largest disk which is disjoint from the body and centered at q. It may be also interesting to restrict the class of non-convex bodies considered (for example, by imposing that the boundary has bounded curvature).

Recently, Ashur and Har-Peled [14] studied various geometric problems with different oracles. For example, they study the ***undecided*** LP ***problem***, in which one is given a collection of hyperplanes but not the halfspaces induced by them. The goal is to find a feasible point in this linear program. For the undecided LP problem, they consider two types of oracle queries: exposure queries (given a hyperplane h, returns the halfspace associated with h), and separation oracle queries (given a point p, either return that p is feasible or return a halfspace bounded by one of the input hyperplanes that violates the feasibility of p). These are excellent examples of interesting problems to study, and provide a way for new oracles and computational models to be proposed. We anticipate much more progress on these class of problems in the future.

References

[1] P. K. Agarwal, E. Ezra, and M. Sharir. *Near-linear approximation algorithms for geometric hitting sets*. *Algorithmica*, 63(1-2): 1–25, 2012. DOI: 10.1007/s00453-011-9517-2.

[2] P. K. Agarwal, J. Matoušek, and O. Schwarzkopf. *Computing many faces in arrangements of lines and segments*. *SIAM J. Comput.*, 27(2): 491–505, 1998. DOI: 10.1137/S009753979426616X.

[3] P. K. Agarwal, J. Matoušek, and M. Sharir. *On range searching with semialgebraic sets. II*. *SIAM J. Comput.*, 42(6): 2039–2062, 2013. DOI: 10.1137/120890855.

[4] P. K. Agarwal and J. Pan. *Near-linear algorithms for geometric hitting sets and set covers*. *Proc. 30th Annu. Sympos. Comput. Geom.* (SoCG), 271, 2014. DOI: 10.1145/2582112.2582152.

[5] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. *Geometric approximation via coresets*. *Combinatorial and computational geometry*, 52: 1–30, 2005.

[6] N. Alon, I. Bárány, Z. Füredi, and D. J. Kleitman. *Point selections and weak ε-nets for convex hulls*. *Comb. Probab. Comput.*, 1: 189–200, 1992. DOI: 10.1017/S0963548300000225.

[7] B. Alspach. *The wonderful Walecki construction*. *Bull. Inst. Combin. Appl.*, 52: 7–20, 2008.

[8] G. Ambrus and I. Bárány. *Longest convex chains*. *Rand. Struct. & Alg.*, 35(2): 137–162, 2009. DOI: 10.1002/rsa.20269.

[9] D. Angluin. *Queries and concept learning*. *Machine Learning*, 2(4): 319–342, 1987. DOI: 10.1007/BF00116828.

[10] M. Anthony and P. L. Bartlett. *Neural network learning: theoretical foundations*. Cambridge University Press, 2002.

[11] B. Aronov, M. de Berg, J. Gudmundsson, and M. Horton. *On β-plurality points in spatial voting games*. *Proc. 36th Int. Annu. Sympos. Comput. Geom.* (SoCG), 7:1–7:15, 2020. DOI: 10.4230/LIPIcs.SoCG.2020.7.

[12] B. Aronov, M. Pellegrini, and M. Sharir. *On the zone of a surface in a hyperplane arrangement*. *Discrete Comput. Geom.*, 9: 177–186, 1993. DOI: 10.1007/BF02189317.

[13] S. Arora. *Polynomial time approximation schemes for Euclidean TSP and other geometric problems*. *J. Assoc. Comput. Mach.*, 45(5): 753–782, 1998. DOI: 10.1109/sfcs.1996.548458.

[14] S. Ashur and S. Har-Peled. *On undecided LP and active learning using proximity queries*. *Proc. 37th Int. Annu. Sympos. Comput. Geom.* (SoCG), to appear.

[15] I. Bárány. *A generalization of Carathéodory's theorem*. *Discrete Math.*, 40(2-3): 141–152, 1982. DOI: 10.1016/0012-365X(82)90115-7.

[16] I. Bárány, Z. Füredi, and L. Lovász. *On the number of halving planes*. *Combinatorica*, 10: 175–183, 1990. DOI: 10.1007/BF02123008.

[17] I. Bárány and Z. Füredi. *Computing the volume is difficulte*. *Discrete Comput. Geom.*, 2: 319–326, 1987. DOI: 10.1007/BF02187886.

[18] M. de Berg, O. Cheong, M. Kreveld, and M. H. Overmars. *Computational geometry: algorithms and applications*. 3rd. Santa Clara, CA, USA: Springer-Verlag, 2008. DOI: 10.1007/978-3-540-77974-2.

[19] M. de Berg, J. Chung, and J. Gudmundsson. *Computing the Yolk in Spatial Voting Games*. 2019.

[20] M. de Berg, K. Dobrindt, and O. Schwarzkopf. *On lazy randomized incremental construction*. *Discrete Comput. Geom.*, 14(3): 261–286, 1995. DOI: 10.1007/BF02570705.

[21] M. de Berg, J. Gudmundsson, and M. Mehr. *Faster algorithms for computing plurality points*. *ACM Trans. Algorithms*, 14(3): 36:1–36:23, 2018. DOI: 10.1145/3186990.

[22] M. W. Bern. *Approximate closest-point queries in high dimensions*. *Inform. Process. Lett.*, 45(2): 95–99, 1993. DOI: 10.1016/0020-0190(93)90222-U.

[23] B. K. Bhattacharya, S. Jadhav, A. Mukhopadhyay, and J.-M. Robert. *Optimal algorithms for some intersection radius problems*. *Computing*, 52(3): 269–279, 1994. DOI: 10.1007/BF02246508.

[24] D. Black. *On the rationale of group decision-making*. *Journal of Political Economy*, 56(1): 23–34, 1948. DOI: 10.1086/256633.

[25] E. Bonnet, P. Giannopoulos, and M. Lampis. *On the parameterized complexity of red-blue points separation*. *J. Comput. Geom.*, 10(1): 181–206, 2019. DOI: 10.20382/jocg.v10i1a7.

[26] H. Brönnimann and M. T. Goodrich. *Almost optimal set covers in finite vc-dimension*. *Discrete Comput. Geom.*, 14(4): 463–479, 1995. DOI: 10.1007/BF02570718.

[27] K. Buchin, S. Har-Peled, and D. Oláh. *A spanner for the day after*. *Discrete Comput. Geom.*, 64(4): 1167–1191, 2020. DOI: 10.1007/s00454-020-00228-6.

[28] B. Bukh, J. Matoušek, and G. Nivasch. *Lower bounds for weak epsilon-nets and stair-convexity*. *Proc. 25th Annu. Sympos. Comput. Geom.* (SoCG), 1–10, 2009. DOI: 10.1145/1542362.1542365.

[29] G. Călinescu, A. Dumitrescu, H. J. Karloff, and P. Wan. *Separating points by axis-parallel lines*. *Internat. J. Comput. Geom. Appl.*, 15(6): 575–590, 2005. DOI: 10.1142/S0218195905001865.

[30] P. B. Callahan and S. R. Kosaraju. *Faster algorithms for some geometric graph problems in higher dimensions*. *Proc. 4th ACM-SIAM Sympos. Discrete Alg.* (SODA), 291–300, 1993.

[31] T. H. Chan, M. Li, L. Ning, and S. Solomon. *New doubling spanners: Better and simpler*. *SIAM J. Comput.*, 44(1): 37–53, 2015. DOI: 10.1137/130930984.

[32] T. M. Chan. *A minimalist's implementation of an approximate nearest neighbor algorithm in fixed dimensions*. 2006.

[33] T. M. Chan. *An optimal randomized algorithm for maximum Tukey depth*. *Proc. 15th ACM-SIAM Sympos. Discrete Algs.* (SODA), 430–436, 2004.

[34] T. M. Chan. *Approximate nearest neighbor queries revisited*. *Discrete Comput. Geom.*, 20(3): 359–373, 1998. DOI: 10.1007/PL00009390.

[35] T. M. Chan. *Closest-point problems simplified on the RAM*. *Proc. 13th ACM-SIAM Sympos. Discrete Alg.* (SODA), 472–473, 2002.

[36] T. M. Chan. *Random sampling, halfspace range reporting, and construction of $(\leq k)$-levels in three dimensions*. *SIAM J. Comput.*, 30(2): 561–575, 2000. DOI: 10.1137/S0097539798349188.

[37] T. M. Chan. *Well-separated pair decomposition in linear time?* *Inform. Process. Lett.*, 107(5): 138–141, 2008. DOI: 10.1016/j.ipl.2008.02.008.

[38] T. M. Chan, S. Har-Peled, and M. Jones. *On locality-sensitive orderings and their applications*. *SIAM J. Comput.*, 49(3): 583–600, 2020. DOI: 10.1137/19M1246493.

[39] T. M. Chan and D. Skrepetos. *Dynamic data structures for approximate Hausdorff distance in the word RAM*. *Comput. Geom. Theory Appl.*, 60: 37–44, 2017. DOI: 10.1016/j.comgeo.2016.08.002.

[40] T. M. Chan and K. Tsakalidis. *Optimal deterministic algorithms for 2-d and 3-d shallow cuttings*. *Discrete Comput. Geom.*, 56(4): 866–881, 2016. DOI: 10.1007/s00454-016-9784-4.

[41] B. Chazelle. *The discrepancy method: randomness and complexity*. New York: Cambridge University Press, 2001.

[42] B. Chazelle and J. Friedman. *A deterministic view of random sampling and its use in geometry*. *Combinatorica*, 10(3): 229–249, 1990. DOI: 10.1007/BF02122778.

[43] B. Chazelle and E. Welzl. *Quasi-optimal range searching in space of finite vc-dimension*. *Discrete Comput. Geom.*, 4: 467–489, 1989. DOI: 10.1007/BF02187743.

[44] B. Chazelle. *An optimal convex hull algorithm in any fixed dimension*. *Discrete Comput. Geom.*, 10: 377–409, 1993. DOI: 10.1007/BF02573985.

[45] B. Chazelle. *Cutting hyperplanes for divide-and-conquer*. *Discrete Comput. Geom.*, 9: 145–158, 1993. DOI: 10.1007/BF02189314.

[46] K. L. Clarkson. *Algorithms for polytope covering and approximation*. *Proc. 3th Workshop Alg. Data Struct.* (WADS), vol. 709. 246–252, 1993. DOI: 10.1007/3-540-57155-8_252.

[47] K. L. Clarkson and P. W. Shor. *Applications of random sampling in computational geometry, II*. *Discrete Comput. Geom.*, 4: 387–421, 1989. DOI: 10.1007/BF02187740.

[48] K. L. Clarkson. *Las vegas algorithms for linear and integer programming when the dimension is small*. *J. ACM*, 42(2): 488–499, 1995. DOI: 10.1145/201019.201036.

[49] K. L. Clarkson, D. Eppstein, G. L. Miller, C. Sturtivant, and S.-H. Teng. *Approximating center points with iterative Radon points*. *Internat. J. Comput. Geom. Appl.*, 6(3): 357–377, 1996. DOI: 10.1142/S021819599600023X.

[50] M. B. Cohen, Y. T. Lee, G. L. Miller, J. Pachocki, and A. Sidford. *Geometric median in nearly linear time*. *Proc. 48th ACM Sympos. Theory Comput.* (STOC), 9–21, 2016. DOI: 10.1145/2897518.2897647.

[51] D. A. Cohn, L. E. Atlas, and R. E. Ladner. *Improving generalization with active learning*. *Machine Learning*, 15(2): 201–221, 1994. DOI: 10.1007/BF00993277.

[52] A. Czumaj and H. Zhao. *Fault-tolerant geometric spanners*. *Discrete Comput. Geom.*, 32(2): 207–230, 2004. DOI: 10.1007/s00454-004-1121-7.

[53] K. Dalal. *Counting the onion*. *Random Struct. Alg.*, 24(2): 155–165, 2004. DOI: 10.1002/rsa.10114.

[54] O. Devillers, F. Hurtado, M. Mora, and C. Seara. *Separating several point sets in the plane*. *Proc. 13th Canad. Conf. Comput. Geom.* (CCCG), 81–84, 2001.

[55] D. P. Dobkin and D. G. Kirkpatrick. *A linear algorithm for determining the separation of convex polyhedra*. *J. Algorithms*, 6(3): 381–392, 1985. DOI: 10.1016/0196-6774(85)90007-0.

[56] D. P. Dubhashi and A. Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

[57] H. Edelsbrunner, R. Seidel, and M. Sharir. *On the zone theorem for hyperplane arrangements*. *SIAM J. Comput.*, 22(2): 418–429, 1993. DOI: 10.1137/0222031.

[58] P. van Emde Boas. *Preserving order in a forest in less than logarithmic time and linear space*. *Inf. Process. Lett.*, 6(3): 80–82, 1977. DOI: 10.1016/0020-0190(77)90031-X.

[59] D. Eppstein. *Dynamic Euclidean minimum spanning trees and extrema of binary functions*. *Discrete Comput. Geom.*, 13: 111–122, 1995. DOI: 10.1007/bf02574030.

[60] H. Everett, J.-M. Robert, and M. van Kreveld. *An optimal algorithm for the $(\leq k)$-levels, with applications to separation and transversal problems*. *Proc. 9th Int. Annu. Sympos. Comput. Geom.* (SoCG), 38–46, 1993. DOI: 10.1145/160985.160994.

[61] E. Ezra and M. Sharir. *A nearly quadratic bound for point-location in hyperplane arrangements, in the linear decision tree model*. *Discrete Comput. Geom.*, 61(4): 735–755, 2019. DOI: 10.1007/s00454-018-0043-8.

[62] J. Fakcharoenphol, S. Rao, and K. Talwar. *A tight bound on approximating arbitrary metrics by tree metrics*. *J. Comput. Sys. Sci.*, 69(3): 485–497, 2004. DOI: 10.1016/j.jcss.2004.04.011.

[63] U. Feige and R. Krauthgamer. *Stereoscopic families of permutations, and their applications*. *Proc. 5th Israel Symp. Theo. Comput. and Systems* (ISTCS), 85–95, 1997. DOI: 10.1109/ISTCS.1997.595160.

[64] S. P. Fekete. *On simple polygonalizations with optimal area*. *Discrete Comput. Geom.*, 23(1): 73–110, 2000. DOI: 10.1007/PL00009492.

[65] J. Ferrera. *An introduction to nonsmooth analysis*. Boston: Academic Press, 2013. DOI: 10.1016/C2013-0-15234-8.

[66] A. Filtser and H. Le. *Reliable spanners: locality-sensitive orderings strike back*. *CoRR*, abs/2101.07428, 2021. arXiv: 2101.07428.

[67] M. L. Fredman and D. E. Willard. *Surpassing the information theoretic bound with fusion trees*. *J. Comput. Sys. Sci.*, 47(3): 424–436, 1993. DOI: 10.1016/0022-0000(93)90040-4.

[68] R. Freimer, J. S. B. Mitchell, and C. D. Piatko. *On the complexity of shattering using arrangements*. Technical Report TR 91-1197. Ithaca, NY: Dept. Comput. Sci., Cornell Univ., Apr. 1991.

[69] L. Gottlieb and L. Roditty. *An optimal dynamic spanner for doubling metric spaces*. *Proc. 16th Annu. Euro. Sympos. Alg.* (ESA), 478–489, 2008. DOI: 10.1007/978-3-540-87744-8_40.

[70] L. Gottlieb and L. Roditty. *Improved algorithms for fully dynamic geometric spanners and geometric routing*. *Proc. 19th ACM-SIAM Sympos. Discrete Alg.* (SODA), 591–600, 2008.

[71] J. Gudmundsson and S. Wong. *Computing the yolk in spatial voting games without computing median lines*. *33th Conf. Artificial Intell.* (AAAI), DOI: 10.1609/aaai.v33i01.33012012.

[72] J. Gudmundsson and S. Wong. *Computing the yolk in spatial voting games without computing median lines*. *CoRR*, abs/1902.04735, 2019. arXiv: 1902.04735.

[73] Y. Guo and R. Greiner. *Optimistic active-learning using mutual information*. *Proc. 20th Int. Joint Conf. on AI* (IJCAI), 823–829, 2007.

[74] S. Har-Peled. *On the expected complexity of random convex hulls*. *CoRR*, abs/1111.5340, 2011.

[75] S. Har-Peled, M. Jones, and S. Rahul. *An animation of the greedy classification algorithm in 2D*. YouTube: https://www.youtube.com/watch?v=IZX0VQdIgNA. 2018.

[76] S. Har-Peled, N. Kumar, D. M. Mount, and B. Raichel. *Space exploration via proximity search*. *Discrete Comput. Geom.*, 56(2): 357–376, 2016. DOI: 10.1007/s00454-016-9801-7.

[77] S. Har-Peled and B. Lidicky. *Peeling the grid*. *SIAM J. Discrete Math.*, 27(2): 650–655, 2013. DOI: 10.1137/120892660.

[78] S. Har-Peled. *Geometric approximation algorithms*. Vol. 173. Math. Surveys & Monographs. Boston, MA, USA: Amer. Math. Soc., 2011. DOI: 10.1090/surv/173.

[79] S. Har-Peled and M. Jones. *Fast algorithms for geometric consensuses*. *Proc. 36th Int. Annu. Sympos. Comput. Geom.* (SoCG), 50:1–50:16, 2020. DOI: 10.4230/LIPIcs.SoCG.2020.50.

[80] S. Har-Peled and M. Jones. *Journey to the Center of the Point Set*. *ACM Trans. Algorithms*, 17(1), 2021. DOI: 10.1145/3431285.

[81] S. Har-Peled and M. Jones. *On separating points by lines*. *Discrete Comput. Geom.*, 63(3): 705–730, 2020. DOI: 10.1007/s00454-019-00103-z.

[82] S. Har-Peled and M. Jones. *Stabbing convex bodies with lines and flats*. *CoRR*, abs/2007.09874, 2020. arXiv: 2007.09874.

[83] S. Har-Peled and M. Jones. *Where is the center of Illinois?* Youtube: https://www.youtube.com/watch?v=NoSvqRGAYYY. 2019.

[84] S. Har-Peled, M. Jones, and S. Rahul. *Active-learning a convex body in low dimensions*. *Algorithmica*: 1–33, 2021. DOI: 10.1007/s00453-021-00807-w.

[85] S. Har-Peled and M. Mendel. *Fast construction of nets in low dimensional metrics, and their applications*. *SIAM J. Comput.*, 35(5): 1148–1184, 2006. DOI: 10.1137/S0097539704446281.

[86] S. Har-Peled and M. Sharir. *Relative (p, ε)-approximations in geometry*. *Discrete Comput. Geom.*, 45(3): 462–496, 2011. DOI: 10.1007/s00454-010-9248-1.

[87] S. Har-Peled and T. Zhou. *Improved Approximation Algorithms for Tverberg Partitions*. 2020. arXiv: 2007.08717.

[88] D. Haussler and E. Welzl. *ε-nets and simplex range queries*. *Discrete Comput. Geom.*, 2: 127–151, 1987. DOI: 10.1007/BF02187876.

[89] G. Hinton and R. Salakhutdinov. *Reducing the dimensionality of data with neural networks*. *Science*, 313(5786): 504–507, 2006. DOI: 10.1126/science.1127647.

[90] D. S. Hochbaum and W. Maass. *Approximation schemes for covering and packing problems in image processing and VLSI*. *J. Assoc. Comput. Mach.*, 32(1): 130–136, 1985. DOI: 10.1145/2455.214106.

[91] J. Holm, E. Rotenberg, and C. Wulff-Nilsen. *Faster fully-dynamic minimum spanning forest*. *Proc. 23rd Annu. Euro. Sympos. Alg.* (ESA), vol. 9294. 742–753, 2015. DOI: 10.1007/978-3-662-48350-3_62.

[92] P. Indyk and R. Motwani. *Approximate nearest neighbors: towards removing the curse of dimensionality*. *Proc. 30th ACM Sympos. Theory Comput.* (STOC), 604–613, 1998. DOI: 10.1145/276698.276876.

[93] S. Janson. *Tail bounds for sums of geometric and exponential variables*. *Stat. Prob. Letters*, 135: 1–6, 2018. DOI: https://doi.org/10.1016/j.spl.2017.11.017.

[94] I. Kamel and C. Faloutsos. *On packing R-trees*. *Proc. 2nd Intl. Conf. Info. Knowl. Mang.*, 490–499, 1993. DOI: 10.1145/170088.170403.

[95] D. M. Kane, S. Lovett, S. Moran, and J. Zhang. *Active classification with comparison queries*. *Proc. 58th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), 355–366, 2017. DOI: 10.1109/FOCS.2017.40.

[96] S. Kapoor and X. Li. *Efficient construction of spanners in d-dimensions*. *CoRR*, abs/1303.7217, 2013. arXiv: 1303.7217.

[97] K. Kedem, R. Livne, J. Pach, and M. Sharir. *On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles*. *Discrete Comput. Geom.*, 1(1): 59–71, 1986. DOI: 10.1007/BF02187683.

[98] C. Keller, S. Smorodinsky, and G. Tardos. *Improved bounds on the Hadwiger-Debrunner numbers*. the internet, Dec. 2015. arXiv: 1512.04026 [math.CO].

[99] C. Keller, S. Smorodinsky, and G. Tardos. *On Max-Clique for intersection graphs of sets and the hadwiger-debrunner numbers*. *Proc. 28th ACM-SIAM Sympos. Discrete Algs.* (SODA), 2254–2263, 2017. DOI: 10.1137/1.9781611974782.148.

[100] S. Kratsch, T. Masařík, I. Muzi, M. Pilipczuk, and M. Sorge. *Optimal discretization is fixed-parameter tractable*. *Proc. 32th ACM-SIAM Sympos. Discrete Algs.* (SODA), 1702–1719, 2021. DOI: 10.1137/1.9781611976465.103.

[101] A. Kupavskii. *The VC-dimension of k-vertex d-polytopes*. *CoRR*, abs/2004.04841, 2020. arXiv: 2004.04841.

[102] J. Kynčl. *Vapnik-Chervonenkis dimension of lines in the plane*. MathOverflow: http://mathoverflow.net/questions/98412/vapnik-chervonenkis-dimension-of-lines-in-the-plane. 2012.

[103] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006. DOI: 10.1017/CBO9780511546877.

[104] C. Levcopoulos, G. Narasimhan, and M. H. M. Smid. *Efficient algorithms for constructing fault-tolerant geometric spanners*. *Proc. 30th ACM Sympos. Theory Comput.* (STOC), 186–195, 1998. DOI: 10.1145/276698.276734.

[105] Y. Li, P. M. Long, and A. Srinivasan. *Improved bounds on the sample complexity of learning*. *J. Comput. Syst. Sci.*, 62(3): 516–527, 2001.

[106] S. Liao, M. A. López, and S. T. Leutenegger. *High dimensional similarity search with space filling curves*. *Proc. 17th Int. Conf. on Data Eng.* (ICDE), 615–622, 2001. DOI: 10.1109/ICDE.2001.914876.

[107] T. Lukovszki. *New results of fault tolerant geometric spanners*. *Proc. 9th Workshop Alg. Data Struct.* (WADS), vol. 1663. 193–204, 1999. DOI: 10.1007/3-540-48447-7_20.

[108] T. Lukovszki. *New results on geometric spanners and their applications*. PhD thesis. University of Paderborn, Germany, 1999.

[109] J. Matoušek. *Approximations and optimal geometric divide-an-conquer*. *J. Comput. Syst. Sci.*, 50(2): 203–208, 1995. DOI: 10.1006/jcss.1995.1018.

[110] J. Matoušek. *Efficient partition trees*. *Discrete & Computational Geometry*, 8: 315–334, 1992. DOI: 10.1007/BF02293051.

[111] J. Matoušek. *Geometric discrepancy: an illustrated guide*. Vol. 18. Algorithms and Combinatorics. Springer, 1999. DOI: 10.1007/978-3-642-03942-3.

[112] J. Matoušek. *Lectures on discrete geometry*. Vol. 212. Grad. Text in Math. Berlin, Heidelberg: Springer, 2002. DOI: 10.1007/978-1-4613-0039-7/.

[113] J. Matoušek. *Reporting points in halfspaces*. *Comput. Geom.*, 2: 169–186, 1992. DOI: 10.1016/0925-7721(92)90006-E.

[114] J. Matoušek and U. Wagner. *New constructions of weak epsilon-nets*. *Discrete Comput. Geom.*, 32(2): 195–206, 2004. DOI: 10.1007/s00454-004-1116-4.

[115] R. D. McKelvey. *Covering, dominance, and institution-free properties of social choice*. *American Journal of Political Science*, 30(2): 283–314, 1986. DOI: 10.2307/2111098.

[116] K. Mehlhorn and S. Näher. *Dynamic fractional cascading*. *Algorithmica*, 5(2): 215–241, 1990. DOI: 10.1007/BF01840386.

[117] S. Merrill III and B. Grofman. *A unified theory of voting: directional and proximity spatial models*. Cambridge University Press, 1999.

[118] F. Meunier, W. Mulzer, P. Sarrabezolles, and Y. Stein. *The rainbow at the end of the line—A PPAD formulation of the colorful carathéodory theorem with applications*. *Proc. 28th ACM-SIAM Sympos. Discrete Algs.* (SODA), 1342–1351, 2017. DOI: 10.1137/1.9781611974782.87.

[119] G. L. Miller and D. R. Sheehy. *Approximate centerpoints with proofs*. *Comput. Geom.*, 43(8): 647–654, 2010. DOI: 10.1016/j.comgeo.2010.04.006.

[120] M. Mitzenmacher and E. Upfal. *Probability and computing: randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[121] G. M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. Tech. rep. Ottawa, Ontario: IBM, Mar. 1966.

[122] K. Mulmuley. *Computational geometry: an introduction through randomized algorithms*. Prentice Hall, 1994.

[123] K. Mulmuley. *On levels in arrangements and voronoi diagrams*. *Discrete Comput. Geom.*: 307–338, 1991. DOI: 10.1007/BF02574692.

[124] W. Mulzer and D. Werner. *Approximating tverberg points in linear time for any fixed dimension*. *Discrete Comput. Geom.*, 50(2): 520–535, 2013. DOI: 10.1007/s00454-013-9528-7.

[125] N. H. Mustafa and S. Ray. *Weak ε-nets have basis of size $O(\varepsilon^{-1} \log \varepsilon^{-1})$ in any dimension*. *Comput. Geom. Theory Appl.*, 40(1): 84–91, 2008. DOI: 10.1016/j.comgeo.2007.02.006.

[126] N. H. Mustafa and K. Varadarajan. *Epsilon-approximations and epsilon-nets*. 2017. arXiv: 1702.03676.

[127] S. C. Nandy, T. Asano, and T. Harayama. *Shattering a set of objects in 2D*. *Disc. Appl. Math.*, 122(1-3): 183–194, 2002. DOI: 10.1016/S0166-218X(01)00315-8.

[128] E. Oh and H.-K. Ahn. *Computing the center region and its variants*. *CoRR*, abs/1910.12169, 2019. arXiv: 1910.12169.

[129] J. Pach and G. Tardos. *Tight lower bounds for the size of epsilon-nets*. *J. Amer. Math. Soc.*, 26: 645–658, 2013. DOI: 10.1090/S0894-0347-2012-00759-0.

[130] F. Panahi, A. Adler, A. F. van der Stappen, and K. Goldberg. *An efficient proximity probing algorithm for metrology*. *Int. Conf. on Automation Science and Engineering, CASE 2013*, 342–349, 2013. DOI: 10.1109/CoASE.2013.6653995.

[131] G. Peano. *Sur une courbe, qui remplit toute une aire plane*. *Mathematische Annalen*, 36(1): 157–160, 1890. DOI: 10.1007/BF01199438.

[132] F. P. Preparata and M. I. Shamos. *Computational geometry - an introduction*. Texts and Monographs in Computer Science. Springer, 1985. DOI: 10.1007/978-1-4612-1098-6.

[133] R. Rado. *A theorem on general measure*. *J. Lond. Math. Soc.*, 21: 291–300, 1947.

[134] L. Roditty. *Fully dynamic geometric spanners*. *Algorithmica*, 62(3-4): 1073–1087, 2012. DOI: 10.1007/s00453-011-9504-7.

[135] A. Rok and S. Smorodinsky. *Weak $1/r$-nets for moving points*. *Proc. 32nd Int. Annu. Sympos. Comput. Geom.* (SoCG), 59:1–59:13, 2016. DOI: 10.4230/LIPIcs.SoCG.2016.59.

[136] D. Rolnick and P. Soberón. *Algorithms for Tverberg's theorem via centerpoint theorems*. 2016. arXiv: 1601.03083.

[137] N. Rubin. *An improved bound for weak epsilon-nets in the plane*. *Proc. 59th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), 224–235, 2018. DOI: 10.1109/FOCS.2018.00030.

[138] N. Rubin. *Stronger bounds for weak epsilon-nets in higher dimensions*. *Proc. 53rd ACM Sympos. Theory Comput.* (STOC), to appear.

[139] A. Rubinstein. *A note about the "nowhere denseness" of societies having an equilibrium under majority rule*. *Econometrica*, 47(2): 511–514, 1979. DOI: 10.2307/1914198.

[140] H. Sagan. *Space-filling curves*. Universitext Series. Springer-Verlag, 1994. DOI: 10.1007/978-1-4612-0871-6.

[141] R. Seidel. *Small-dimensional linear programming and convex hulls made easy*. *Discrete Comput. Geom.*, 6: 423–434, 1991. DOI: 10.1007/BF02574699.

[142] B. Settles. *Active Learning Literature Survey*. Tech. rep. #1648. Computer Science, Univ. Wisconsin, Madison, Jan. 2009.

[143] M. I. Shamos. *Computational Geometry*. PhD thesis. New Haven, CT, USA, 1978.

[144] M. Sharir and P. K. Agarwal. *Davenport-Schinzel sequences and their geometric applications*. New York: Cambridge University Press, 1995.

[145] M. Sharir and E. Welzl. *A combinatorial bound for linear programming and related problems*. *9th Symp. on Theoretical Aspects of Comput. Sci.* (STACS), 569–579, 1992. DOI: 10.1007/3-540-55210-3_213.

[146] S. Solomon. *From hierarchical partitions to hierarchical covers: Optimal fault-tolerant spanners for doubling metrics*. *Proc. 46th ACM Sympos. Theory Comput.* (STOC), 363–372, 2014. DOI: 10.1145/2591796.2591864.

[147] W. Steiger and J. Zhao. *Generalized ham-sandwich cuts*. *Discrete Comput. Geom.*, 44(3): 535–545, 2010. DOI: 10.1007/s00454-009-9225-8.

[148] R. E. Stone and C. A. Tovey. *Limiting median lines do not suffice to determine the yolk*. *Social Choice and Welfare*, 9(1): 33–35, 1992. DOI: 10.1007/BF00177668.

[149] S.-H. Teng. *Points, Spheres, and Separators: A Unified Geometric Approach to Graph Partitioning*. PhD thesis. Pittsburgh, PA 15213-3890: School of Computer Science, Carnegie Mellon University, Aug. 1991.

[150] C. A. Tovey. *A polynomial-time algorithm for computing the yolk in fixed dimension*. *Math. Program.*, 57: 259–277, 1992. DOI: 10.1007/BF01581084.

[151] V. N. Vapnik and A. Y. Chervonenkis. *On the uniform convergence of relative frequencies of events to their probabilities*. *Theory Probab. Appl.*, 16: 264–280, 1971.

[152] J.-L. Verger-Gaugry. *Covering a ball with smaller equal balls in $\mathbb{R}^n$*. *Discrete Comput. Geom.*, 33(1): 143–155, 2005. DOI: 10.1007/s00454-004-2916-2.

[153] *Random Polytopes, Convex Bodies, and Approximation*. *Stochastic Geometry: Lectures given at the C.I.M.E. Summer School held in Martina Franca, Italy, September 13–18, 2004*. Ed. by W. Weil. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 77–118. DOI: 10.1007/978-3-540-38175-4_2.

[154] A. Wuffle, S. L. Feld, G. Owen, and B. Grofman. *Finagle's law and the finagle point, a new solution concept for two-candidate competition in spatial voting games without a core*. *American Journal of Political Science*, 33(2): 348–375, 1989. DOI: 10.2307/2111151.