

Geometric Computing over Uncertain Data

by

Wuzhou Zhang

Department of Computer Science
Duke University

Date: _____

Approved:

Pankaj K. Agarwal, Supervisor

Sayan Mukherjee

Kamesh Munagala

Jun Yang

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2015

ABSTRACT

Geometric Computing over Uncertain Data

by

Wuzhou Zhang

Department of Computer Science
Duke University

Date: _____

Approved:

Pankaj K. Agarwal, Supervisor

Sayan Mukherjee

Kamesh Munagala

Jun Yang

An abstract of a dissertation submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2015

Copyright © 2015 by Wuzhou Zhang
All rights reserved except the rights granted by the
Creative Commons Attribution-Noncommercial Licence

Abstract

Entering the era of big data, human beings are faced with an unprecedented amount of geometric data today. Many computational challenges arise in processing the new deluge of geometric data. A critical one is data uncertainty: the data is inherently noisy and inaccurate, and often lacks completeness. The past few decades have witnessed the influence of geometric algorithms in various fields including GIS, spatial databases, and computer vision, etc. Yet most of the existing geometric algorithms are built on the assumption of the data being precise and are incapable of properly handling data in the presence of uncertainty. This thesis explores a few algorithmic challenges in what we call *geometric computing over uncertain data*.

We study the nearest-neighbor searching problem, which returns the nearest neighbor of a query point in a set of points, in a probabilistic framework. This thesis investigates two different nearest-neighbor formulations: expected nearest neighbor (ENN), where we consider the expected distance between each input point and a query point, and probabilistic nearest neighbor (PNN), where we estimate the probability of each input point being the nearest neighbor of a query point.

For the ENN problem, we consider a probabilistic framework in which the location of each input point and/or query point is specified as a probability density function and the goal is to return the point that minimizes the expected distance. We present methods for computing an exact ENN or an ε -approximate ENN, for a given error parameter $0 < \varepsilon < 1$, under different distance functions. These methods build an

index of near-linear size and answer ENN queries in polylogarithmic or sublinear time, depending on the underlying function. As far as we know, these are the first nontrivial methods for answering exact or ε -approximate ENN queries with provable performance guarantees. Moreover, we extend our results to answer exact or ε -approximate k -ENN queries. Notably, when only the query points are uncertain, we obtain state-of-the-art results for top- k aggregate (group) nearest-neighbor queries in the L_1 metric using the weighted SUM operator.

For the PNN problem, we consider a probabilistic framework in which the location of each input point is specified as a probability distribution function. We present efficient algorithms for (i) computing all points that are nearest neighbors of a query point with nonzero probability; (ii) estimating, within a specified additive error, the probability of a point being the nearest neighbor of a query point; (iii) using it to return the point that maximizes the probability being the nearest neighbor, or all the points with probabilities greater than some threshold to be the nearest neighbor. We also present some experimental results to demonstrate the effectiveness of our approach.

We study the convex-hull problem, which asks for the smallest convex set that contains a given point set, in a probabilistic setting. In our framework, the uncertainty of each input point is described by a probability distribution over a finite number of possible locations including a *null* location to account for non-existence of the point. Our results include both exact and approximation algorithms for computing the probability of a query point lying inside the convex hull of the input, time-space tradeoffs for the membership queries, a connection between Tukey depth and membership queries, as well as a new notion of β -hull that may be a useful representation of uncertain hulls.

We study contour trees of terrains, which encode the topological changes of the level set of the height value ℓ as we raise ℓ from $-\infty$ to $+\infty$ on the terrains, in a

probabilistic setting. We consider a terrain that is defined by linearly interpolating each triangle of a triangulation. In our framework, the uncertainty lies in the height of each vertex in the triangulation, and we assume that it is described by a probability distribution. We first show that the probability of a vertex being a critical point, and the expected number of nodes (resp. edges) of the contour tree, can be computed exactly efficiently. Then we present efficient sampling-based methods for estimating, with high probability, (i) the probability that two points lie on an edge of the contour tree, within additive error; (ii) the expected distance of two points p, q and the probability that the distance of p, q is at least ℓ on the contour tree, within additive error and/or relative error, where the distance of p, q on a contour tree is defined to be the difference between the maximum height and the minimum height on the unique path from p to q on the contour tree.

Dedicated to the uncertainty of life

Contents

Abstract	iv
List of Tables	xii
List of Figures	xiii
1 Geometric Computing over Uncertain Data	
— An Introduction	1
1.1 Introduction	1
1.2 Uncertainty Models	3
1.2.1 Deterministic model	3
1.2.2 Existential model	4
1.2.3 Locational model	4
1.3 Prior Work	5
1.4 Contributions	7
2 Expected Nearest Neighbor	13
2.1 Introduction	13
2.2 Preliminaries	20
2.3 Squared Euclidean Distance	25
2.3.1 Uncertain data	25
2.3.2 Uncertain query	28
2.4 Rectilinear Metric	30

2.4.1	Uncertain data	31
2.4.2	Uncertain query	36
2.5	Euclidean Distance	42
2.5.1	Approximation of the expected Euclidean distance	42
2.5.2	Uncertain data, ε -ENN queries	47
2.5.3	Uncertain data, (k, ε) -ENN queries	53
2.5.4	Uncertain query	59
2.6	Conclusion	61
3	Aggregate Nearest Neighbor	62
3.1	Introduction	62
3.2	Top- k ANN Searching in the 1-D Space	66
3.3	Top- k ANN Searching in the Plane	70
3.3.1	The minimal points and the skyline	72
3.3.2	Computing the top- k ANN set $S_k(P^1)$	79
3.4	Top- k Aggregate Farthest Neighbor Searching	100
3.5	Conclusion	101
4	Probabilistic Nearest Neighbor	102
4.1	Introduction	102
4.1.1	Problem definition	102
4.1.2	Previous work	105
4.1.3	Our results	107
4.2	Nonzero Probabilistic Voronoi Diagram	109
4.2.1	Continuous case	109
4.2.2	Discrete case	122
4.3	Data Structures for $NN_{\neq 0}$ Queries	124

4.4	Quantification Probabilities	127
4.4.1	A Monte-Carlo algorithm	129
4.4.2	A spiral-search algorithm	133
4.5	Experimental Results	137
4.6	Conclusion	139
5	Probabilistic Convex Hull	140
5.1	Introduction	140
5.2	Membership Probability in the Plane	142
5.2.1	The unipoint model	143
5.2.2	The multipoint model	145
5.2.3	Dealing with degeneracies	147
5.3	Membership Probability in Higher Dimensions	148
5.3.1	The unipoint model	148
5.3.2	The multipoint model	152
5.4	Membership Queries	154
5.4.1	Probability map	154
5.4.2	A Monte-Carlo algorithm	159
5.5	Tukey Depth and Convex Hull	161
5.6	β -Hull	166
5.7	Conclusion	171
6	Probabilistic Contour Tree	172
6.1	Introduction	172
6.2	Probability of a Vertex Being a Critical Point	177
6.3	Probability of Two Points Lying on an Edge of the Contour Tree . . .	179
6.4	The Distance Statistics of Two Points	189

6.4.1	The expected distance of two points	190
6.4.2	Probability that the distance of two points is at least ℓ	200
6.5	Conclusion	203
	Bibliography	204
	Biography	214

List of Tables

2.1	Summary of our results.	18
-----	---------------------------------	----

List of Figures

2.1	An example EVD in \mathbb{R}^2 where $d(\cdot, \cdot)$ is the L_1 metric.	16
2.2	Euclidean Voronoi diagram as the minimization diagram.	21
2.3	A simple example point set, its quadtree and its compressed quadtree.	23
2.4	An example of $\text{Ed}(P_i, q)$ in the L_1 metric.	30
2.5	Lower bound construction for EVD.	31
2.6	(a) The set of rectangles \mathcal{B} . (b) The four quadrants of \square	37
2.7	Hollow circles have been visited; Squares are event points.	40
2.8	An illustration for the proof of Lemma 2.5.3.	44
2.9	An exponential grid composed of canonical squares.	45
2.10	(a) A single pair (A, B) in an α -WSPD. (b) D_1^B, \dots, D_t^B constructed for a pair (A, B) in a $(1/8)$ -WSPD.	50
2.11	(a) The minimization diagram for an exposed node of \mathbb{T} . (b) A portion of the ε -EVD.	51
3.1	Illustrating minimal points, dominance relationship, and the skyline.	72
3.2	Illustrating the algorithm in Lemma 3.3.4.	77
3.3	An example of the skylines π_1 and π_2	86
3.4	Illustrating the proof of Lemma 3.3.7.	89
4.1	(a) An example of P_i represented by a uniform distribution. (b) $g_{q,i}(x)$	103
4.2	Examples of $\Delta(q)$ and $\text{NN}_{\neq 0}(q)$ with disks being uncertainty regions.	110
4.3	An example of a breakpoint and an intersection point of γ_i 's.	112

4.4	An example of γ_1	113
4.5	(a) $\Omega(n^3)$ lower bound construction. (b) An illustration of the proof. .	114
4.6	$\Omega(n^3)$ lower bound construction using disks of same radius	116
4.7	An illustration for the proof of Lemma 4.2.8.	118
4.8	Any pair (P_i, P_j) satisfying $j - i \geq 2$ determines 2 vertices of $\mathcal{V}_{\neq 0}$. . .	121
4.9	An illustration of the proof of Lemma 5.4.1	127
4.10	Monte Carlo method.	138
4.11	Spiral search method.	138
5.1	(A) A witness edge. (B) Sites in radial order around q . (C) The set W_i . (D) The set W_u	145
5.2	A three-dimensional example of an p_i -escaping face f for q	149
5.3	A facet f_j projected to the orthogonal complement plane.	151
5.4	An example of probability map.	155
5.5	The cases to consider for computing the probability of C' from C . . .	157
5.6	The cases to consider for computing the probability of e from C	158
5.7	Illustration for the halfplanes h_{ℓ_1} and h_{ℓ_2}	164
5.8	An example of β -hull.	168
5.9	A $(1/r)$ -cutting Ξ for computing the upper hull.	169
6.1	(a) A terrain Σ ; (b) its height level map M_h ; (c) its contour tree T_h . .	174
6.2	A lower bound construction on the number of distinct contour trees. .	180

Geometric Computing over Uncertain Data — An Introduction

1.1. Introduction

Entering the era of *big data*, today human beings are faced with an unprecedented amount of geometric data, such as locational traces recorded by GPS devices, location information produced by smart phones (typically associated with photos or social network statuses), point clouds acquired by 3D scanners, and detailed Google/Bing Maps. Even non-geometric data, such as videos and images, are often processed by first reducing to geometric data through embedding into a multi-dimensional feature space. In fact, the past few decades have witnessed the influence of geometric algorithms in various fields including GIS, spatial databases, computer vision, computer graphics, and CAD/CAM, so there is an urgent call for efficient and practical geometric algorithms that are scalable and robust enough to handle *big* geometric data.

Many other computational challenges arise in exploring the new deluge of geometric data. A critical one is *data uncertainty*. There is a growing awareness of uncertainty in the geometric data we are collecting nowadays. The data is inherently noisy and inaccurate, and often lacks of completeness. For example, GPS devices only provide

certain level of accuracy; due to hardness reason, the geo-location of an IP address is only approximate and specified as a region; different sensors might produce different readings for the same target; for a moving object, its velocity is measured say, every 10 seconds, instead of continuously, to conserve energy; in privacy, noise is imposed into data through various means. Yet most of the existing geometric algorithms are built on the assumption of the data being precise and are incapable of properly processing data in the presence of uncertainty. In response to this computational challenge in handling geometric data, the focus of this thesis is what we call *geometric computing over uncertain data* — through a small subset of representative geometric problems together with a probabilistic framework used to model uncertainty, inspired by models in machine learning and Bayesian statistics.

Among numerous geometric problems that can be casted under uncertainty, we have devoted ourselves to three of the most fundamental ones, specifically, *nearest-neighbor searching*, *convex hull*, and *contour tree*, with an emphasis on geometric queries, combinatorial structures, and topological structures, respectively. All of these three problems, well-studied in the exact case, are often building blocks of many others, and have wide-ranging applications, including but not limited to, computer graphics, computer vision, geographic information systems, databases, image processing, pattern recognition, visualizations, robotics, combinatorics, and statistics. Therefore, we believe they are key problems that can help unveil the mask of uncertainty role in both practice and theory, and that offer first steps towards illustrating the computational challenges that arise when handling uncertain data. Theoretically, we have explored some algorithmic challenges in geometric computing over uncertain data: (i) how the input uncertainty affects the output quality, e.g. we may have to associate a confidence level or likelihood with each output, or simply compute certain statistics such as the expectation of the output; and (ii) how the input uncertainty impacts the computational efficiency of an algorithm to obtain the

output, e.g. it may become space-prohibiting or indeed computationally intractable. Empirically, we have conducted some preliminary experiments on real datasets (for nearest-neighbor searching) to validate the effectiveness of our approaches.

1.2. Uncertainty Models

Different models have been proposed for modeling uncertainty in geometric data: mainly classified into *deterministic models* and *probabilistic models*. Probabilistic models can be further classified into the *existential model* and the *locational model*, which are simple and mathematically tractable enough for algorithmic designs and analyses, and well capture the probabilistic nature of uncertainty, hence the focus of this thesis.

1.2.1. Deterministic model

In this model, each input point lies inside a region, which can be a square, a disk, a polygon, etc. An earlier model, ε -geometry, a special case of the deterministic model, was introduced by Salesin, Stolfi and Guibas in the late 1980s as a way to cope with computational errors in geometric algorithms [107]. In the ε -geometry model, each input point is assumed to be at most ε away from its true location; alternatively, each input point lies inside a disk of radius ε centered at its true location. The region associated with an input point represents the knowledge we have about that point. Consider the example that Walmart plans to open a new store in the Research Triangle Park, but the exact location has not been chosen. Then the location of the new store may be represented as the region of Research Triangle Park.

1.2.2. Existential model

In the existential uncertainty model, each *uncertain* point¹ p exists with independent probability π_p , and absent otherwise. A finite set $P = \{p_1, p_2, \dots, p_n\}$ of such points in \mathbb{R}^d is called a stochastic point set. For example, sensors may become inactive with some probability due to hardware failures, and post offices may be closed with some probability based on their operating hours, etc.

1.2.3. Locational model

In the locational uncertainty model, an uncertain point P in \mathbb{R}^d is represented as a continuous probability distribution defined by a probability density function (**pdf**) $f_P: \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$; f_P may be a parametric **pdf** such as a uniform distribution or a Gaussian distribution, or may be a non-parametric **pdf** such as a histogram. For example, a vehicle's location obtained using GPS can be represented as a Gaussian distribution centered at its reading with variance being the accuracy level of the GPS device. We also consider the case where P is represented as a discrete distribution defined by a finite set $P = \{p_1, \dots, p_s\} \subset \mathbb{R}^d$ along with a set of probabilities $\{w_1, \dots, w_s\} \subset [0, 1]$, where $w_i = \Pr[P \text{ is } p_i]$ and $\sum_{i=1}^s w_i = 1$. For instance, if an object's location is measured by multiple devices, and we have varying confidence levels towards different devices, then the location of that object can be described as a discrete distribution.

Note that the locational model generalizes the existential model if we allow $\sum_{i=1}^s w_i < 1$, and furthermore, it is a more general model than the deterministic model because of the probability density function imposed on the uncertainty region (the deterministic model can be thought of imposing a *uniform* distribution on the uncertainty region). In this thesis, we focus on both the existential model and the

¹If the location of data is precise, we call it *certain*.

locational model.

1.3. Prior Work

Uncertainty is ubiquitous. Many disciplines have been battling with, and will continue dealing with, the presence of uncertainty in the data. See the book [43] for uncertainty handling in databases, books [17, 122] in data mining, the book [78] in statistical science, the book [59] in remote sensing and GIS, among many others.

In computational geometry, most of the earlier methods assumed that the input points and the query points were precise. In many applications, such as sensor databases, location based services, face recognition, and mobile data, the location of data is imprecise. This has led to a flurry of activity on geometric computing over uncertain data. Below, we briefly survey some of the most recent work. More related work can be found in each individual chapter.

In the deterministic model, much work has been done in computational geometry, see e.g. [30, 31, 32, 66, 93, 107, 120] and Maarten Löffler's Ph.D. thesis [92] for geometric computing under this model. Many of them have focused on *extreme* values of geometric measures such as diameter and bounding box, etc. For example, it was shown that the largest (resp. smallest) *smallest enclosing circle* of n imprecise points in the deterministic model can be found in $O(n)$ time when the regions are all squares or all disks, which also holds for the largest *smallest bounding box* in [92]. And for the smallest *smallest bounding box*, it takes time $O(n)$ for squares while $O(n^2)$ for disks. In [30], Cabello studied the problem of spreading points, i.e., placing n points, each one inside its own prespecified disk, with the objective of maximizing the distance between the closest pair of them. He showed that the problem of spreading points is NP-hard and gave several approximation algorithms, e.g., in the L_∞ metric, a 2-approximation algorithm running in $O(n\sqrt{n}\log^2 n)$ time was given. In [31], Cabello and van Kreveld considered approximating the problem of aligning

points, i.e., aligning as many points as possible horizontally, vertically, or diagonally, when each point is allowed to be placed anywhere in its own given region.

In the existential model, researchers have investigated nearest-neighbor searching, closest pair, convex hulls, minimum spanning trees, etc [79, 80, 104, 115, 116]. Let \mathcal{P} be a set of n uncertain points in the existential model. Very recently, Suri and Verbeek [115] showed that the size of the most likely Voronoi diagram of \mathcal{P} in \mathbb{R}^1 is: (1) $\Theta(n^2)$ in the worst case, (2) $\Theta(kn)$ if the input has only k distinct probability values, (3) $O(n \log n)$ on average, and (4) $O(n\sqrt{n})$ under smoothed analysis. Pérez-Lantero [104] showed that for both area and perimeter of the convex hull of \mathcal{P} , it is NP-hard to compute the probability that the measure is at least a given bound w . For a parameter $\varepsilon \in (0, 1)$, Pérez-Lantero offered a deterministic algorithm with running time $O(n^9/\varepsilon)$ which returns a value that is between the probability that the area is at least w , and the probability that the area is at least $(1 - \varepsilon)w$. Finally, a Monte-Carlo algorithm was proposed to estimate such a probability within additive error ε with high probability.

In the locational model, numerous results have been obtained on nearest neighbor queries, top- k queries, convex hulls, skylines, range queries, clustering, among numerous results [7, 28, 44, 45, 50, 64, 73, 85, 91, 114, 118, 125, 127], in both the database (known as probabilistic databases) and the computational geometry community; see [17, 51] for surveys on uncertain data. Take uncertain skylines for an example. People have studied ρ -skyline and τ -skyline, for both of which we are given a set \mathcal{P} of n uncertain points in the plane, where each uncertain point P_i is described by a discrete probability distribution defined over k locations. For a parameter $\rho \in (0, 1]$, the ρ -skyline of \mathcal{P} consists of all the uncertain points of \mathcal{P} whose skyline probabilities are at least ρ , and it can be computed in $O(m^2)$ time in a straightforward manner, where $m = nk$. Atallah and Qi [22] devised the first sub-quadratic algorithm with running time $O(m^{5/3} \text{polylog}(n))$. Later, Afshani *et al.* [1] improved the running time

to $O(m^{3/2})$, and they also constructed a set of uncertain points suggesting that this bound might be optimal. For the case $k \ll n$, the running time can be further improved to $O(mk \log m)$. Alternatively, in [125], we proposed τ -skyline: given a parameter $\tau \in [0, 1]$, the τ -skyline region of \mathcal{P} is the set of points q in the plane such that the probability of any P_i dominating q is at most τ , and the τ -skyline of \mathcal{P} is the boundary of the τ -skyline region of \mathcal{P} . We showed that τ -skyline of \mathcal{P} can be computed in $O(m \log m)$ time using a very simple approach, where $m = nk$. The definition of the τ -skyline of P_i is closely related to the concept of K -skyband, proposed in [101]. Given a set P of n (exact) points and a parameter $K \leq n$, the K -skyband of P asks for the set of points which are dominated by at most K points of P . 0-skyband corresponds to the conventional skyline. As a product, we obtained the first algorithm for computing the weighted skyband of a set of weighted points [125].

From above, it can be seen that some problems become indeed intractable in the presence of uncertainty, and one has to turn to approximation. One more example is that it was shown to be NP-hard in [80] to compute the probability that the closest pair distance is less than a given value l , even in \mathbb{R}^2 and L_2 norm, and a linear-space data structure was given with $O(\log n)$ query time to compute the expected distance of a given query point to its $(1 + \varepsilon)$ -approximate nearest neighbor when the dimension d is a constant. For some other problems, it becomes impractical, though theoretically tractable. For example, we have shown that the expected Voronoi diagram for ENN or the most likely Voronoi diagram for PNN can have size quadratic, cubic or even worse [6, 9], far from satisfying compared with the linear-size Voronoi diagram in the exact case.

1.4. Contributions

Nearest-neighbor searching. Motivated by a wide range of applications, nearest-neighbor searching has been studied in many different fields including computational

geometry, database systems and information retrieval; see [23, 48, 103] for surveys on this topic. In its simplest form, it asks for preprocessing a set S of n points in \mathbb{R}^d into an index so that the nearest neighbor (NN) in S of a query point can be reported quickly. Voronoi diagrams are used to perform nearest-neighbor searching among a set of n input points in \mathbb{R}^2 , with $O(n \log n)$ preprocessing time, $O(n)$ space, and $O(\log n)$ query time [52]. Unfortunately, the size of the Voronoi diagram is $\Theta(n^{\lceil d/2 \rceil})$ in \mathbb{R}^d . The best known method for answering an NN query, requires $O((n/m^{\lceil d/2 \rceil})\text{polylog}(n))$ query time for an $O(m)$ -space structure, where $n < m < n^{\lceil d/2 \rceil}$ [13]. To obtain better performance, many researchers turned to approximate nearest-neighbor searching: given any $\varepsilon > 0$, a point p is an ε -approximate nearest neighbor of q if $d(q, p) \leq (1 + \varepsilon)d(q, p^*)$, where p^* is the actual nearest neighbor. Arya *et al.* [20] attained space-time trade-offs for approximate nearest-neighbor searching: given a tradeoff parameter γ , where $2 \leq \gamma \leq 1/\varepsilon$, there exists an index of space $O(n\gamma^{d-1} \log(1/\varepsilon))$ that can answer ε -NN queries in time $O(\log(n\gamma) + 1/(\varepsilon\gamma)^{(d-1)/2})$. There is also extensive work on answering approximate nearest neighbor queries using locality sensitive hashing, when d is not a constant and the goal is to have an algorithm whose query time is polynomial in d ; see e.g. [19, 70].

In the presence of uncertainty, it is even unclear how to formulate the nearest-neighbor searching problem. Given a query point, what is the nearest neighbor in a set of uncertain points? Can one efficiently return the (approximated) distribution of the nearest distance, or some simple statistics such as the expected nearest distance? We touch upon some of these questions in this thesis, and have obtained nontrivial results with two different problem formulations: *expected nearest neighbor* and *probabilistic nearest neighbor*.

In the formulation of expected nearest neighbor, one considers the expected distance from each data point to the query point, and the goal is to return the point that minimizes the expected distance, which we refer to as the expected nearest

neighbor (ENN). In [9], we presented methods for computing an exact ENN or an ε -approximate ENN, for a given error parameter $0 < \varepsilon < 1$, under different distance functions. These methods build an index of near-linear size and answer ENN queries in polylogarithmic or sublinear time, depending on the underlying function. As far as we know, these are the first nontrivial methods for answering exact or ε -approximate ENN queries with provable performance guarantees. This thesis also generalizes the original ENN results presented in [9] to k -ENN queries. In many applications, such as computer vision and pattern recognition, retrieving k points instead of one point is required. It turns out that some of the generalizations are nontrivial, and require new ideas. All these results are included in Chapter 2, based on the joint work with Pankaj K. Agarwal, Alon Efrat, and Swaminathan Sankararaman [9].

Moreover, we obtained improved results in the L_1 metric when the query point is uncertain and the input data points are certain. The results were also generalized to k -ENN queries. Notably, when only the query points are uncertain, we have obtained state-of-the-art results for top- k aggregate (group) nearest-neighbor queries in the L_1 metric using the weighted SUM operator. The improved results are included in Chapter 3, based on the joint work with Haitao Wang [124].

In the formulation of probabilistic nearest neighbor, one considers the probability of each data point being the NN of the query point, which we refer to as the probabilistic nearest neighbor (PNN). We obtained efficient algorithms for (i) computing all points that are nearest neighbors of a query point with nonzero probability; (ii) estimating, within a specified additive error, the probability of a point being the nearest neighbor of a query point; (iii) using it to return the point that maximizes the probability being the nearest neighbor, or all the points with probabilities greater than some threshold to be the NN. We also conducted some experimental results to demonstrate the effectiveness of our approach. The results for probabilistic nearest neighbor are presented in Chapter 4, based on the joint work with Pankaj K. Agarwal, Boris

Aronov, Sariel Har-Peled, Jeff M. Phillips, and Ke Yi [6].

Convex hull. Convex hull is a fundamental structure in mathematics and computational geometry. Given a set of points \mathcal{P} in d -space, the *convex hull* of \mathcal{P} is the minimal convex polytope that contains all points in \mathcal{P} . Convex hulls have applications in a variety of areas including but not limited to computer graphics, image processing, pattern recognition, robotics, combinatorics and statistics. Due to their importance in practice, the algorithms for computing convex hulls are well-studied. The convex hull of n points can be computed in $O(n \log n)$ time [61, 106] for $d = 2, 3$, and in $O(n^{\lfloor d/2 \rfloor})$ time [39] for $d > 3$. These bounds are worst-case optimal [25, 39], and output-sensitive algorithms are also known [34, 42, 83, 97, 109]. See the survey [110] for an overview of known results. In many applications, the location and sometimes even the existence of the data is uncertain, but statistical information can be used as a probability distribution guide for data. This raises the natural computational question: what is a robust and useful convex hull representation for such an uncertain input, and how well can we compute it? We explore this problem under two simple models in which both the location and the existence (presence) of each point is described probabilistically, and study basic questions such as what is the probability of a query point lying inside the convex hull, or what does the probability distribution of the convex hull over the space look like.

We studied the convex-hull problem in a probabilistic setting. In our framework, the uncertainty of each input point is described by a probability distribution over a finite number of possible locations including a *null* location to account for non-existence of the point. Our results include both exact and approximation algorithms for computing the probability of a query point lying inside the convex hull of the input, time-space tradeoffs for the membership queries, a connection between Tukey depth and membership queries, as well as a new notion of β -hull that may be a useful

representation of uncertain hulls. We present the results for convex hulls in Chapter 5, based on the joint work with Pankaj K. Agarwal, Sarel Har-Peled, Subhash Suri, and Hakan Yıldız [12].

Contour tree. Contour tree is a fundamental structure for topological analysis and data visualizations on large volume data sets, such as terrains² and images. Efficient algorithms have been devised for computing contour trees of terrains in memory [33, 117, 119], I/O-efficiently [5], and for maintaining contour trees of dynamic terrains [4], where terrains are defined as piecewise-linear height functions in \mathbb{R}^3 . Due to the inherent measurement errors, it is reasonable to assume that the height of each vertex of the underlying triangulation defining a terrain is described probabilistically. This triggers a natural question: what is the contour tree of such a resulting terrain? As there can be exponential number of contour tree instances, can we compute/estimate some statistics among these contour tree instances? For example, what is the probability of two points lying on an edge of the contour tree? What is the expected distance of two points p, q on the contour tree, where the distance of p, q on a contour tree is defined to be the difference between the maximum height and the minimum height on the unique path of from p to q on the contour tree, as defined in [24]? We look into some of the computational challenges related to contour trees of terrains imposed by the uncertainty on the vertex heights.

We studied contour trees of terrains, which encode the topological changes of the level set of the height value ℓ as we raise ℓ from $-\infty$ to $+\infty$ on the terrains, in a probabilistic setting. We consider a terrain that is defined by linearly interpolating each triangle of a triangulation. In our framework, the uncertainty lies in the height of each vertex in the triangulation, and we assume that it is described by a probability

²A terrain is the graph of a piecewise-linear height function in \mathbb{R}^3 , induced by a triangulation in \mathbb{R}^2 . See Chapter 6 for a more precise definition.

distribution. We showed that the probability of a vertex being a critical point, and the expected number of nodes (resp. edges) of the contour tree, can be computed exactly efficiently. Moreover, we obtained efficient sampling-based methods for estimating, with high probability, (i) the probability that two points lie on an edge of the contour tree, within additive error; (ii) the expected distance of two points p, q and the probability that the distance of p, q is at least ℓ on the contour tree, within additive error and/or relative error, where the distance of p, q on a contour tree is defined to be the difference between the maximum height and the minimum height on the unique path from p to q on the contour tree. We include our results for contour trees in Chapter 6, joint work with Pankaj K. Agarwal and Sayan Mukherjee.

Expected Nearest Neighbor

2.1. Introduction

Nearest-neighbor (NN) search, which returns the nearest neighbor of a query point in a set of points, is an important and widely studied problem in many fields, and it has wide range of applications. In many of them, such as sensor databases, location-based services, face recognition, and mobile data, the location of data is imprecise. In this chapter we are interested in answering NN queries over uncertain data—the location of input points or the query point is not precisely known, and we assume that it is given as a probability density function. The existing methods for answering NN queries on precise data cannot be applied directly to this setting and new methods are needed.

Problem statement. We assume that each uncertain point P follows from the locational uncertainty model defined in Section 1.2. Let $d(\cdot, \cdot)$ denote a distance function in \mathbb{R}^d ; we consider L_1, L_2, L_∞ -metrics or squared Euclidean distance¹. For a given $d(\cdot, \cdot)$, the *expected distance* between two independent uncertain points P and

¹ the squared Euclidean distance between two points $p, q \in \mathbb{R}^d$ is $\|p - q\|^2$ where $\|\cdot\|$ is the L_2 metric.

Q is defined as

$$\text{Ed}(P, Q) = \iint f_P(x) f_Q(y) d(x, y) dx dy.$$

If f_P and f_Q are discrete pmfs of size s each, then

$$\text{Ed}(P, Q) = \sum_{i=1}^s \sum_{j=1}^s w_i w'_j d(p_i, q_j),$$

where w_i, w'_j are the probabilities of P and Q being at p_i and q_j respectively. If Q is a (certain) point in \mathbb{R}^d , i.e., $Q = \{q\}$, then

$$\text{Ed}(P, q) = \sum_{i=1}^s w_i d(p_i, q).$$

We say that the *description complexity* of f_P is s if it can be represented using $O(s)$ parameters and certain basic primitive operations on f_P can be performed in $O(s)$ time. In particular, $\text{Ed}(P, x)$, for any $x \in \mathbb{R}^2$, can be computed in $O(s)$ time, and the expected location $\int x f_P(x) dx$ of P , also called the *centroid* of P , can be computed in $O(s)$ time. For example, a discrete pdf consisting of at most s points, and a piecewise-constant or piecewise-linear pdf consisting of at most s pieces have description complexity s . Gaussian (under certain distance functions) and inverse-distance distributions have constant description complexity.

Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n uncertain points in \mathbb{R}^d , each of which is independently chosen. For simplicity, let f_i denote f_{P_i} , the pdf of P_i . For an uncertain point Q , its *expected nearest neighbor* (ENN), denoted by $\varphi(\mathcal{P}, Q)$, is

$$\varphi(\mathcal{P}, Q) = \underset{P \in \mathcal{P}}{\text{argmin}} \text{Ed}(P, Q).$$

For $i \geq 1$, let $\varphi_i(\mathcal{P}, Q)$ be the i -th *expected NN* of Q in \mathcal{P} , i.e., $\text{Ed}(Q, \varphi_i(\mathcal{P}, Q))$ is the i -th smallest expected distance from Q to the points of \mathcal{P} . Note that $\varphi_1(\mathcal{P}, Q) = \varphi(\mathcal{P}, Q)$.

For $k \geq 1$, let $\Phi_k(\mathcal{P}, Q) = \{\varphi_i(\mathcal{P}, Q) \mid 1 \leq i \leq k\}$, and we refer to $\Phi_k(\mathcal{P}, Q)$ as the k -ENN of Q .

For a parameter $0 < \varepsilon < 1$, we call a point $P \in \mathcal{P}$ an ε -approximate ENN (or ε -ENN, for brevity) of Q if

$$\text{Ed}(P, Q) \leq (1 + \varepsilon)\text{Ed}(\varphi(\mathcal{P}, Q), Q).$$

We call a set $\mathcal{T} \subseteq \mathcal{P}$ an ε -approximate k -ENN (or (k, ε) -ENN, for brevity) of Q if for every $\varphi_i(\mathcal{P}, Q) \in \Phi_k(\mathcal{P}, Q)$, $1 \leq i \leq k$, there exists $P \in \mathcal{T}$, such that

$$(1 - \varepsilon)\text{Ed}(\varphi_i(\mathcal{P}, Q), Q) \leq \text{Ed}(P, Q) \leq (1 + \varepsilon)\text{Ed}(\varphi_i(\mathcal{P}, Q), Q).$$

Note that the size of \mathcal{T} may be smaller than k when there exists $P \in \mathcal{T}$ which ε -approximates more than one point in $\Phi_k(\mathcal{P}, Q)$. Alternatively, one may think of \mathcal{T} as a multiset of size k .

Next, we introduce the notion of the *expected Voronoi diagram* of \mathcal{P} . For $1 \leq i \leq n$, we define the *expected Voronoi cell* $\text{EVor}(P_i)$ as

$$\text{EVor}(P_i) = \{x \in \mathbb{R}^d \mid \text{Ed}(P_i, x) \leq \text{Ed}(P_j, x), \forall j\}.$$

The decomposition of \mathbb{R}^d into maximal connected regions induced by $\text{EVor}(P_i)$, $1 \leq i \leq n$, is called the *expected Voronoi diagram*, $\text{EVD}(\mathcal{P})$ of \mathcal{P} . See Figure 2.1 for an example EVD in \mathbb{R}^2 where $d(\cdot, \cdot)$ is the L_1 metric. A decomposition of \mathbb{R}^d into connected cells, each cell τ labeled with $\lambda(\tau) \in \mathcal{P}$, is called an ε -approximate EVD of \mathcal{P} (or ε -EVD(\mathcal{P}) for brevity) if for all $x \in \tau$, $\lambda(\tau)$ is an ε -ENN of x .

The above notion can be extended to the *order- k expected Voronoi diagram* of \mathcal{P} . For any subset $\mathcal{T} \subseteq \mathcal{P}$ of size k , the *order- k expected Voronoi cell* $\text{EVor}_k(\mathcal{T})$ is defined as

$$\text{EVor}_k(\mathcal{T}) = \{x \in \mathbb{R}^d \mid \text{Ed}(P_i, x) \leq \text{Ed}(P_j, x), \forall P_i \in \mathcal{T}, \forall P_j \in \mathcal{P} - \mathcal{T}\}.$$

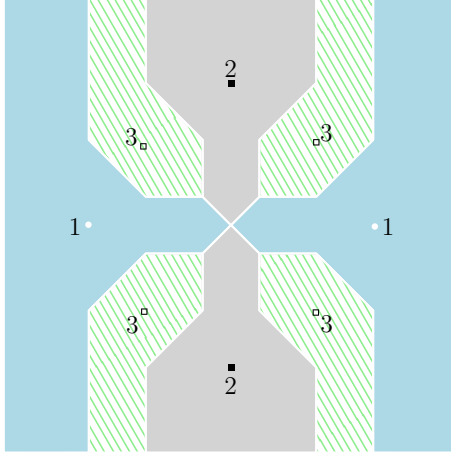


FIGURE 2.1. An example EVD in \mathbb{R}^2 where $d(\cdot, \cdot)$ is the L_1 metric. $\mathcal{P} = \{P_1, P_2, P_3\}$. P_1 has two locations: $(-5, 0)$ and $(5, 0)$, each with probability 0.5. P_2 has two locations: $(0, -5)$ and $(0, 5)$, each with probability 0.5. P_3 has four locations: $(-3, -3)$, $(-3, 3)$, $(3, -3)$ and $(3, 3)$, each with probability 0.25. Domain shown: $[-8, 8]^2$.

The decomposition of \mathbb{R}^d into maximal connected regions induced by $\text{EVD}_k(\mathcal{T})$, for all subsets $\mathcal{T} \subseteq \mathcal{P}$ of size k , is called the order- k expected Voronoi diagram of \mathcal{P} , and denoted by $\text{EVD}_k(\mathcal{P})$.

In this chapter, we study the problem of answering exact or approximate ENN and k -ENN queries when input points are uncertain or the query is an uncertain point. We also study the problem of computing $\text{EVD}(\mathcal{P})$, $\text{EVD}_k(\mathcal{P})$ and ε -EVD(\mathcal{P}).

Previous results. In the uncertainty setting, nearest neighbor searching has been studied in both the existential model and the locational model (see Section 1.2). In the existential model, Kamousi *et al.* [80] proposed a linear-space index with $O(\log n)$ query time to compute an ε -approximate value of the expected distance from a query point to its nearest neighbor when the dimension d is a constant.

In the locational model, when the data is uncertain but the query is exact, researchers have studied top- k probable nearest neighbor, probabilistic nearest neighbor, and superseding nearest neighbor [28, 44, 45, 85, 91, 118, 127]. Ljosa *et al.* [91] investigated the expected k -NN under L_1 metric using and obtained ε -approximation.

Cheng *et al.* [44] studied the probabilistic nearest neighbor query that returns those uncertain points whose probabilities of being the nearest neighbor are higher than some threshold, allowing some given error in the answers. All of these methods were based on heuristics and did not provide any guarantee on the query time in the worst case. Moreover, recent results that rely on Voronoi diagram for supporting nearest neighbor queries under uncertainty cannot be adapted to answer ENN (see [47, 76, 111]). We are not aware of any index that uses near-linear space and returns in sublinear time the expected nearest neighbor or a point that is the most likely nearest neighbor.

The problem of computing the expected nearest neighbor when the queries are uncertain but the input is exact is closely related to the *aggregate nearest neighbors* (ANN) problem. Given a set of points \mathcal{P} in a metric space X with a distance function d , the aggregate nearest neighbor to a set of query points Q is defined as $\text{ANN}(Q, \mathcal{P}) = \arg \min_{p \in \mathcal{P}} g(p, Q)$, where $g(p, Q)$ is some aggregation function of the distances from points of Q to p . The aggregation functions commonly considered are SUM, corresponding to the minimization of the summation of the individual distances, and MAX, corresponding to the minimization of the maximum distance. If the pdf is a uniform distribution, the ENN problem is the same as the ANN problem under the SUM aggregation function. Several heuristics are known for answering ANN queries [69, 87, 90, 94, 100, 112, 126]. Li *et al.* [86] provided a polynomial-time approximation scheme for ANN queries under the MAX aggregation function. Li *et al.* [88] presented approximation schemes under MAX and SUM functions for any metric space as long as an efficient nearest-neighbor algorithm is provided. For the SUM function, they provide a 3-approximation which, to the best of our knowledge, is the best known approximation factor. See also [90].

Our results. In this chapter, we focus on the locational model of uncertainty to

Table 2.1. Summary of our results for k -ENN queries under squared Euclidean distance and rectilinear distance, and for (k, ε) -ENN queries under Euclidean distance. Setting $k = 1$ gives corresponding results for ENN or ε -ENN queries. The \tilde{O} notation hides polylog factors of n , k , and ε .

Distances	Settings	Space	Preprocessing	Query
Squared Euclidean	Uncertain Data	$O(n)$	$O(n \log n + ns)$	$O(\log n + k)$
	Uncertain Query	$O(n)$	$O(n \log n)$	$O(\log n + s + k)$
Rectilinear Distance	Uncertain Data	$O(s^2 n \log^2 n)$	$O(s^2 n \log^3 n)$	$O(\log^3(sn) + k \log(\log(sn) + k))$
	Uncertain Query	$O(n \log^2 n)$	$O(n \log^3 n)$	$O(s^2 \log^2 n + k \log(s \log n + k))$
Euclidean Distance	Uncertain Data	$\tilde{O}(nk/\varepsilon^2)$	$\tilde{O}(nk/\varepsilon^2)$	$O(\log(n/\varepsilon) + k)$
	Uncertain Query	$O(n)$	$O(n \log n)$	$O((1/\varepsilon^2) \log(1/\varepsilon)(\log n + s) + k)$

study the problem of nearest neighbor searching. We present efficient algorithms for answering ENN and k -ENN queries under various distance functions. For simplicity, we state the results in \mathbb{R}^2 , i.e., $\mathcal{P} = \{P_1, \dots, P_n\}$ is a set of n uncertain points in \mathbb{R}^2 . We assume that the description complexity of the pdf of each P_i is s . Table 2.1 summarizes our main results.

Squared Euclidean distance (Section 2.3). If $d(\cdot, \cdot)$ is the squared Euclidean distance, then we show that a set \mathcal{P} of n uncertain points can be replaced by a set $\bar{\mathcal{P}}$ of n weighted points such that the weighted Voronoi diagram of $\bar{\mathcal{P}}$ under $d(\cdot, \cdot)$ (also called the *power diagram* of $\bar{\mathcal{P}}$ [23]) is the same as $\text{EVD}(\mathcal{P})$. In particular, $\text{EVD}(\mathcal{P})$ has linear size and can be computed in $O(n \log n + ns)$ time if the pdf of each P_i has description complexity s . Furthermore, $\text{EVD}(\mathcal{P})$ can be preprocessed in $O(n \log n)$ time into a linear size index so that an ENN query for a (certain) point can be answered in $O(\log n)$ time. For k -ENN queries, we show that \mathcal{P} can be preprocessed, in $O(n \log n + ns)$ expected time, where s is the description complexity of the pdf of each P_i , into a linear-size index such that a k -ENN query can be answered in $O(\log n + k)$ time. If the query is also an uncertain point Q , then we show that $\varphi(\mathcal{P}, Q)$ is the same as $\varphi(\mathcal{P}, \bar{q})$, where $\bar{q} = \int x f_Q(x) dx$ is the centroid of Q .

Rectilinear distance (Section 2.4). We assume that each pdf f_i is a discrete pdf

consisting of s points. We show that $\text{EVD}(\mathcal{P})$ has $O(n^2 s^2 \alpha(n))$ complexity, where $\alpha(n)$ is the inverse Ackermann function, and that it can be computed in the same time. We also show that there exists a set \mathcal{P} of n uncertain points with $s = 2$ such that $\text{EVD}(\mathcal{P})$ has $\Omega(n^2)$ vertices. We then describe an index of size $O(s^2 n \log^2 n)$ that can answer an ENN query in $O(\log^3(sn))$ time. The index can be built in $O(s^2 n \log^3 n)$ time. For k -ENN queries, the query time becomes $O(\log^3(sn) + k \log(\log(sn) + k))$, while the space and the preprocessing time remain the same. Next, we show that a set \mathcal{P} of n (certain) points in \mathbb{R}^2 can be stored in an index of size $O(n \log^2 n)$ so that for an uncertain point with discrete pdf consisting of s points, an ENN query can be answered in $O(s^2 \log^2 n)$ time. The index can be built in $O(n \log^2 n)$ time. For k -ENN queries, we show that \mathcal{P} can be preprocessed, in $O(n \log^3 n)$ time, into an index of size $O(n \log^2 n)$, so that a k -ENN query can be answered in $O(s^2 \log^2 n + k \log(s \log n + k))$ time. We note that L_1 and L_∞ metrics are closely related, so these results also hold for the L_∞ metric.

Euclidean distance (Section 2.5). Since the expected distance function under Euclidean distance is algebraically quite complex even for discrete pdfs, we focus on answering ε -ENN and (k, ε) -ENN queries. First, we show that the expected distance to an uncertain point P can be approximated by two parts: (i) a piecewise-constant function defined inside a square, consisting of $O((1/\varepsilon^2) \log(1/\varepsilon))$ pieces; (ii) the distance to the centroid of P outside the square. Using this result, we construct, in $O((n/\varepsilon^2) \log^2(n) \log(n/\varepsilon) \log(1/\varepsilon))$ time, an ε -EVD of \mathcal{P} of size $O((n/\varepsilon^2) \log(1/\varepsilon))$; each face of the subdivision is the region lying between two nested squares—it can be partitioned into at most four rectangles, so that the ε -EVD is a rectangular subdivision. Moreover, for any query point, we can return its ε -ENN in $O(\log(n/\varepsilon))$ time. For (k, ε) -ENN queries, an index of size $O((nk/\varepsilon^2) \log k \log(1/\varepsilon))$ can be constructed in $O((nk/\varepsilon^2) \log(k) \log^2(n) \log(n/\varepsilon) \log(1/\varepsilon))$ time, so that a (k, ε) -ENN query can be

answered in $O(\log(n/\varepsilon) + k)$ time.

Finally, we show that a set \mathcal{P} of n (certain) points in \mathbb{R}^2 can be stored in an index of linear size so that, for an uncertain point with pdf of description complexity s , an ENN query can be answered in $O((1/\varepsilon^2) \log(1/\varepsilon)(\log n + s))$ time. The index can be built in $O(n \log n)$ time. For (k, ε) -ENN queries, an index of linear size can be built on \mathcal{P} in $O(n \log n)$ expected time so that a query can be answered in $O((1/\varepsilon^2) \log(1/\varepsilon)(\log n + s) + k)$ time. These results can be extended to any L_p metric.

We remark that most of our algorithms extend to higher dimensions, but the query time increases exponentially with d ; we mention specific results in the appropriate sections.

Outline of this chapter. We begin in Section 2.2 by describing a few geometric concepts that will be useful. Section 2.3 describes our algorithms for the squared Euclidean distance function, Section 2.4 for the L_1 and L_∞ metrics, and Section 2.5 for the Euclidean distance. We conclude by making a few final remarks in Section 2.6.

2.2. Preliminaries

In this section, we describe a few geometric concepts and data structures that we need.

Lower envelopes and Voronoi diagrams. Let $F = \{f_1, \dots, f_n\}$ be a set of n bivariate functions. The lower envelope of F is defined as

$$\mathbb{L}_F(x) = \min_{1 \leq i \leq n} f_i(x),$$

and the *minimization diagram* of F , denoted by $\mathbb{M}(F)$, is the projection of the graph of \mathbb{L}_F on \mathbb{R}^2 . $\mathbb{M}(F)$ is a planar subdivision in which the same function appears on the lower envelope for all points inside a cell. The (combinatorial) complexity of \mathbb{L}_F

and $\mathbb{M}(F)$ is the number of vertices, edges, and faces in $\mathbb{M}(F)$. If we define $f_i(x)$ to be $\text{Ed}(P_i, x)$, then $\text{EVD}(\mathcal{P})$ is the minimization diagram of the resulting functions. Figure 2.2 shows the Voronoi diagram of a set of points as the minimization diagram of its distance functions.

The notion of lower envelope and minimization diagram can be extended to partially defined functions: f_i is defined over a region $V_i \subseteq \mathbb{R}^2$, then $\mathbb{L}_F(x)$ is the minimum over all functions f_i of F that are defined at x , i.e., $x \in V_i$. Let \mathcal{R} be a set of polygons, each consisting of a constant number of vertices (e.g. triangles, rectangles) in \mathbb{R}^3 . By viewing each of them as the graph of a partially-defined linear function, we can define the lower envelope $\mathbb{L}_{\mathcal{R}}$ and minimization diagram $\mathbb{M}(\mathcal{R})$ of \mathcal{R} . It is known that the complexity of $\mathbb{M}(\mathcal{R})$ is $\Theta(n^2\alpha(n))$ and that it can be computed in $O(n^2\alpha(n))$ time [113], where $\alpha(n)$ is the inverse Ackermann function.

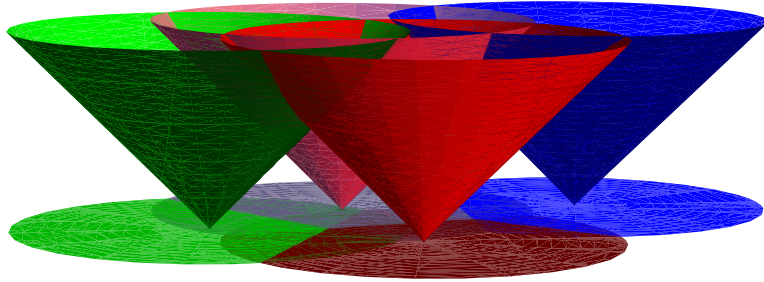


FIGURE 2.2. Euclidean Voronoi diagram of (certain) points as the minimization diagram of their distance functions.

Compressed quadtree. A square in \mathbb{R}^2 is called *canonical* if its side length is 2^l for an integer l and its bottom-left corner is $(2^l a, 2^l b)$ for some integers a, b . Note that two canonical squares are either disjoint or one of them is contained in the other.

A *quadtree* on a canonical square H is a 4-way tree T , each of whose nodes v is associated with a canonical square $\square_v \subseteq H$. The root of T is associated with H itself. The squares associated with the children of a node v are obtained by dividing each side of \square_v into two halves, thereby dividing \square_v into four congruent canonical squares.

If the side length of H is 2^L , then the nodes of T at depth δ induce a $2^\delta \times 2^\delta$ uniform grid inside H ; each grid cell has side length $2^{L-\delta}$. See Figure 2.3 for an example.

Let $\mathcal{B} = \{B_1, \dots, B_m\}$ be a set of m canonical squares inside H . A *compressed quadtree* \mathbb{T} on (\mathcal{B}, H) is constructed as follows: Let T be the quadtree on H as described above. A square $B \in \mathcal{B}$ is stored at a node v if $\square_v = B$. The leaves of \mathbb{T} are the lowest nodes that store a square of \mathcal{B} . They induce a subdivision of H into canonical squares, none of them contains any square of \mathcal{B} in its interior. If a node $v \in T$ does not store a square of \mathcal{B} and both v and $p(v)$, the parent of v , have degree one, we delete v and the child of v becomes the child of $p(v)$. We repeat this step until no such node is left. The size of \mathbb{T} is $O(m)$, and it can be constructed directly, without constructing T , in $O(m \log m)$ time [70]. See Figure 2.3.

We call a node v of \mathbb{T} *exposed* if its degree is at most one. We associate a region R_v with each exposed node v . If v is a leaf, then $R_v = \square_v$. Otherwise, v has one child w and we set $R_v = \square_v \setminus \square_w$. For a point $x \in R_v$, v is the lowest node such that $x \in \square_v$. The regions R_v of the exposed nodes induce a partition $\mathbb{E}(\mathcal{B}, H)$ of H of size $O(m)$. Each face of $\mathbb{E}(\mathcal{B}, H)$ is a canonical square or the difference between two canonical squares, and none of the faces contains a square of \mathcal{B} in its interior. The depth of \mathbb{T} is $\Theta(m)$ in the worst case. Nevertheless, using standard tree-decomposition schemes, for a point $x \in H$, the lowest node of \mathbb{T} such that $x \in \square_v$ can be computed in $O(\log m)$ time [70]. We thus have the following.

Lemma 2.2.1. *Let H be a canonical square, and let \mathcal{B} be a set of m canonical squares in H . A compressed quadtree \mathbb{T} on (\mathcal{B}, H) of size $O(m)$ can be constructed in $O(m \log m)$ time, and it can be preprocessed in additional $O(m \log m)$ time, so that for a point $q \in H$, the lowest node v of \mathbb{T} such that $q \in \square_v$ can be reported in $O(\log m)$ time.*

\mathbb{T} can also be used to store a set $S = \{p_1, \dots, p_n\}$ of points in H . We again build

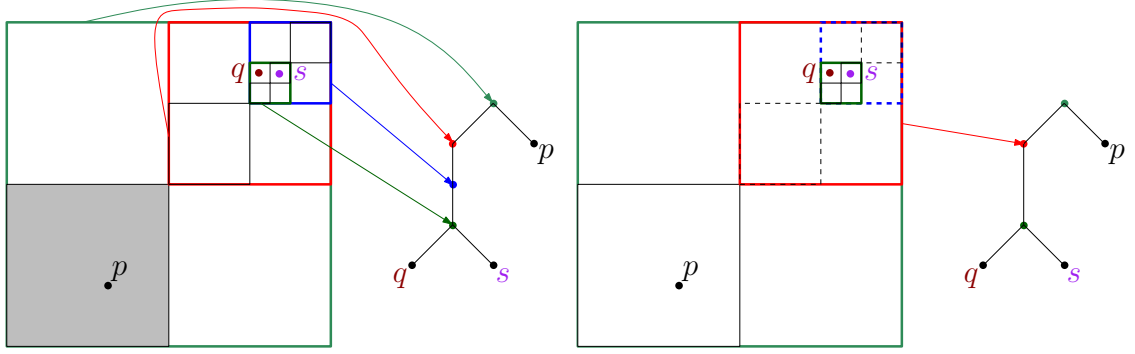


FIGURE 2.3. A simple example point set, its quadtree and its compressed quadtree.

a quadtree T on H . A node $v \in T$ is a leaf if $|S \cap \square_v| \leq 1$. We compress the nodes as above and define the partition $\mathbb{E}(S, H)$ as earlier. Again, using tree-decomposition schemes, we can now determine in $O(\log n)$ time whether $\sigma \cap S \neq \emptyset$ for a canonical square σ . If the answer is yes, we can also return all the points in $S \cap \sigma$ in $O(\log n + t)$ time, where $t = |S \cap \sigma|$. Hence, we have the following.

Lemma 2.2.2. *Let H be a canonical square, and let S be a set of n points in H . A compressed quadtree \mathbb{T} on (S, H) of size $O(n)$ can be constructed in $O(n \log n)$ time, and it can be preprocessed in additional $O(n \log n)$ time, so that for a canonical square σ , we can return all the points in $S \cap \sigma$, if $S \cap \sigma \neq \emptyset$, in $O(\log n + t)$ time, where $t = |S \cap \sigma|$.*

Approximate order- k Voronoi diagrams. Given a set $S = \{p_1, \dots, p_n\}$ of points and any point q in \mathbb{R}^2 , we say that a point $p \in S$ is an ε -approximate i -th NN of q if

$$(1 - \varepsilon)d(q, \text{NN}_i(q)) \leq d(q, p) \leq (1 + \varepsilon)d(q, \text{NN}_i(q)),$$

where $\text{NN}_i(q)$ denotes the exact i -th NN of q in S . We say that a subset $S' \subseteq S$ of size k is a set of ε -approximate k NNs, or (k, ε) -NN for brevity, of q , if for any i , $1 \leq i \leq k$, there exists an ε -approximate i -th NN in S' . An ε -approximate i -th Voronoi diagram of S is a subdivision of \mathbb{R}^2 such that the same input point is an

ε -approximate i -th NN for all points within each cell. An ε -approximate order- k Voronoi diagram, denoted by (k, ε) -VD, of S is a subdivision of \mathbb{R}^2 such that one can store a set $\Phi_c \subseteq S$ of k points with each cell c of the subdivision that is a (k, ε) -NN of all points in c . A (k, ε) -VD can be built as follows.

For each $i \leq k$, we build an ε -approximate i -th Voronoi diagram of S using the algorithm by [71]. Their algorithm constructs a compressed quadtree \mathbb{T}_i such that the subdivision induced by the exposed nodes of \mathbb{T}_i is an ε -approximate i -th Voronoi diagram of \mathcal{P} . \mathbb{T}_i has size $O((n/(i\varepsilon^2)) \log(1/\varepsilon))$, it can be built in $O(n \log n + (n/(i\varepsilon^2))(1/\varepsilon + \log n) \log(1/\varepsilon))$ time, and an ε -approximate i -th NN query can be answered in $O(\log(n/(i\varepsilon)))$ time. We can obtain a compressed quadtree \mathbb{T} by overlaying all the k compressed quadtrees $\mathbb{T}_1, \dots, \mathbb{T}_k$. The exposed nodes of \mathbb{T} induce a subdivision \mathbb{M} , which is the desired (k, ε) -VD of S .

Note that we do not store the (k, ε) -NN for each cell of \mathbb{M} explicitly, otherwise the size of the (k, ε) -VD increases by a factor of k . We observe that for two adjacent cells of \mathbb{M} , if their common edge belongs to the subdivisions of t \mathbb{T}_i 's, then their (k, ε) -NN's differ by at most t points. Therefore, using a persistent data structure [54], we can store the (k, ε) -NN's for all cells using $O((n/\varepsilon^2) \log(k) \log(1/\varepsilon))$ space, so that for any cell, its (k, ε) -NN can be retrieved in $O(\log(n/\varepsilon) + k)$ time. Furthermore, the (k, ε) -VD of S can be built in $O(nk \log n + (n/\varepsilon^2)(1/\varepsilon + \log n) \log(k) \log(1/\varepsilon))$ time. Therefore, we obtain the following.

Lemma 2.2.3. *Given a set of $S = \{p_1, \dots, p_n\}$ of points in \mathbb{R}^2 , a (k, ε) -VD of S of size $O((n/\varepsilon^2) \log(k) \log(1/\varepsilon))$ can be built in $O(nk \log n + (n/\varepsilon^2)(1/\varepsilon + \log n) \log(k) \log(1/\varepsilon))$ time, such that a (k, ε) -NN query can be answered in $O(\log(n/\varepsilon) + k)$ time.*

Remarks. We prove in Section 2.5.3 that the subdivision induced by \mathbb{T}_1 itself is a (k, ε) -VD of S , but unlike \mathbb{M} , the (k, ε) -NN's stored at two adjacent cells may

be completely different, so if we store them for each cell, the total size will be $O((kn/\varepsilon^2) \log(1/\varepsilon))$, even though the size of the subdivision is $O((n/\varepsilon^2) \log(1/\varepsilon))$.

2.3. Squared Euclidean Distance

In this section, for two points $a, b \in \mathbb{R}^2$, $d(a, b) = \|a - b\|^2$. We first show how to compute the EVD of a set of uncertain points, and then show how to answer an ENN query with an uncertain point.

2.3.1. Uncertain data

Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n uncertain points in \mathbb{R}^2 . The following lemma, well known in mathematics, suggests how to replace \mathcal{P} with a set of weighted points. We provide a proof for the sake of completeness.

Lemma 2.3.1. *Let P be an uncertain point in \mathbb{R}^2 , let f be its pdf, let \bar{p} be its centroid, and let $\sigma^2 = \int_{\mathbb{R}^2} \|x - \bar{p}\|^2 f(x) dx$. Then for any point $q \in \mathbb{R}^2$,*

$$\text{Ed}(P, q) = \|q - \bar{p}\|^2 + \sigma^2.$$

Proof. Let $\langle p, q \rangle$ denote the inner product of p and q , and $\|\cdot\|$ denote the Euclidean metric. Using the fact that $\int_{\mathbb{R}^2} f(x) dx = 1$, we obtain

$$\begin{aligned}
\text{Ed}(P, q) &= \int_{\mathbb{R}^2} \|q - x\|^2 f(x) dx \\
&= \|q\|^2 - 2\langle q, \int_{\mathbb{R}^2} x f(x) dx \rangle + \int_{\mathbb{R}^2} \|x\|^2 f(x) dx \\
&= \|q - \bar{p}\|^2 - \|\bar{p}\|^2 + \int_{\mathbb{R}^2} \|x\|^2 f(x) dx \\
&= \|q - \bar{p}\|^2 - 2\|\bar{p}\|^2 + \int_{\mathbb{R}^2} (\|x - \bar{p}\|^2 + 2\langle x, \bar{p} \rangle) f(x) dx \\
&= \|q - \bar{p}\|^2 - 2\|\bar{p}\|^2 + \sigma^2 + 2\langle \bar{p}, \bar{p} \rangle \\
&= \|q - \bar{p}\|^2 + \sigma^2.
\end{aligned}$$

□

Let p be a weighted point in \mathbb{R}^2 with weight w_p . For a point $q \in \mathbb{R}^2$, we define the (weighted) distance from q to p as

$$\delta(q, p) = \|q - p\|^2 + w_p.$$

If we replace each point in $P_i \in \mathcal{P}$ by a weighted point \bar{p}_i whose weight is $\sigma_i^2 = \int_{\mathbb{R}^2} \|x - \bar{p}_i\|^2 f_i(x) dx$, then by the above lemma $\delta(q, \bar{p}_i) = \text{Ed}(P_i, q)$. Set $\bar{\mathcal{P}} = \{\bar{p}_1, \dots, \bar{p}_n\}$. $\text{EVD}(\mathcal{P})$ is the same as the Voronoi diagram of $\bar{\mathcal{P}}$ under the distance function $\delta(\cdot, \cdot)$. We now show how to compute the Voronoi diagram of $\bar{\mathcal{P}}$.

For each $1 \leq i \leq n$, we define a linear function $h_i : \mathbb{R}^2 \rightarrow \mathbb{R}$ as

$$h_i(x) = 2\langle \bar{p}_i, x \rangle - \|\bar{p}_i\|^2 - \sigma_i^2.$$

We will use h_i to denote the graph of this linear function, which is a plane in \mathbb{R}^3 .

The proof of the following lemma is straightforward (see e.g., [52]).

Lemma 2.3.2. *For any $q \in \mathbb{R}^2$,*

$$\arg \min_{1 \leq i \leq n} \delta_i(q, \bar{p}_i) = \arg \max_{1 \leq i \leq n} h_i(q).$$

Let $h_i^+ = \{x \in \mathbb{R}^3 \mid h_i(x) \geq 0\}$ be the halfspace in \mathbb{R}^3 lying above the plane h_i . Set $H^+ = \{h_i^+ \mid 1 \leq i \leq n\}$. By Lemma 2.3.2, the minimization diagram of functions $\{\delta(x, \bar{p}_i) \mid 1 \leq i \leq n\}$ is the same as the xy -projection of $\bigcap_{h^+ \in H^+} h^+$. Since the intersection of n halfspaces in \mathbb{R}^3 has linear size and can be computed in $O(n \log n)$ time [52], we conclude that the Voronoi diagram of $\bar{\mathcal{P}}$, under $\delta(\cdot, \cdot)$ as the distance function, can be computed in $O(n \log n)$ time, and thus $\text{EVD}(\mathcal{P})$ can be computed in $O(n \log n + ns)$ time, where the extra $O(ns)$ time is required for computing $\bar{\mathcal{P}}$. Furthermore, by preprocessing $\text{EVD}(\mathcal{P})$ into a linear-size index for point-location queries, an ENN query for a point $q \in \mathbb{R}^2$ can be answered in $O(\log n)$ time. We thus obtain the following.

Theorem 2.3.3. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 . $\text{EVD}(\mathcal{P})$ under the squared Euclidean distance has $O(n)$ size. If the description complexity of the pdf of every point in \mathcal{P} is s , then $\text{EVD}(\mathcal{P})$ can be computed in $O(n \log n + ns)$ time. Furthermore, $\text{EVD}(\mathcal{P})$ can be preprocessed in a linear-size index so that for a point $q \in \mathbb{R}^2$, an ENN query can be answered in $O(\log n)$ time.*

Remarks. This can be extended to higher dimensions. In \mathbb{R}^d , $d \geq 3$, the space becomes $O(n^{\lfloor d/2 \rfloor})$, and the preprocessing time becomes $O(n^{\lfloor d/2 \rfloor} \log n + ns)$. The query time remains the same.

Answering k -ENN queries. We now generalize this to k -ENN queries. Set $H = \{h_i \mid 1 \leq i \leq n\}$, and denote $\mathcal{A}(H)$ as the arrangement of H . A point p is said to be at level k with respect to H if there are exactly k hyperplanes in H lying strictly below p . The k -level of $\mathcal{A}(H)$ is the closure of all faces of $\mathcal{A}(H)$ whose interior points have level k with respect to H . It is not hard to see that $\text{EVD}_k(\mathcal{P})$ is the projection of the $(k - 1)$ -level of $\mathcal{A}(H)$. The projection of the 0-level is $\text{EVD}(\mathcal{P})$. However, the

²The arrangement $\mathcal{A}(H)$ of H is the subdivision of \mathbb{R}^3 induced by H that consists of 0-, 1-, 2-, and 3-dimensional faces.

complexity of $\text{EVD}_k(\mathcal{P})$ is $O(nk^{5/3})$ [53], so we describe a different approach, without constructing $\text{EVD}_k(\mathcal{P})$ explicitly, that uses $O(n)$ space. For $k \geq 0$, let $\mathcal{A}_{\leq k}(H)$ denote the set of points which have level at most k with respect to H .

A result by Chan [35, Lemma 3.1], which adapts an earlier construction by Matoušek [96], shows that $\mathcal{A}_{\leq k}(H)$ can be covered by $O(n/k)$ pairwise-disjoint cells, each intersecting $O(k)$ planes of H . Furthermore, each cell is a triangular prism unbounded from below whose top face is a triangle. The top faces of these cells form a concave surface. The cells and the list of planes intersecting each cell can be constructed in $O(n \log n)$ expected time. Let Ξ be the set of these cells, and let Ξ^* be the set of xy -projections of these cells; Ξ^* is a set of pairwise-disjoint triangles that triangulate \mathbb{R}^2 . For each prism $\tau \in \Xi$, let $H_\tau \subseteq H$ be the set of planes that intersect τ , and for the xy -projected triangle τ^* of τ , let $\bar{P}_\tau = \{\bar{p}_i \mid h_i \in H_\tau\}$. By construction, $|\bar{P}_\tau| = O(k)$, and for any point $q \in \tau^*$, the k nearest points of q in \bar{P} belong to \bar{P}_τ . We preprocess Ξ^* in $O((n/k) \log(n/k))$ time for point-location queries, and store \bar{P}_τ for each $\tau^* \in \Xi^*$. The total size of the index is $\sum_{\tau \in \Xi} |\bar{P}_\tau| = O(n)$, and the expected preprocessing time is $O(n \log n)$.

Let $q \in \mathbb{R}^2$ be a query point for which we wish to compute $\Phi_k(\mathcal{P}, q)$. We first find the triangle $\tau^* \in \Xi^*$ that contains q . Next, we compute $\delta(q, \bar{p}_i)$ for every $\bar{p}_i \in \bar{P}_\tau$ and then return the k points with the smallest k values of $\delta(q, \bar{p}_i)$. The total query time is $O(\log n + k)$. We thus obtain the following.

Theorem 2.3.4. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 . If the description complexity of the pdf of every point in \mathcal{P} is s , then \mathcal{P} can be preprocessed into an index of size $O(n)$ in $O(n \log n + ns)$ expected time so that for a point $q \in \mathbb{R}^2$, a k -ENN query can be answered in $O(\log n + k)$ time.*

2.3.2. Uncertain query

Let Q be an uncertain query point in \mathbb{R}^2 represented as a pdf f_Q .

Lemma 2.3.5. For an uncertain point Q with a pdf f_Q ,

$$\varphi(\mathcal{P}, Q) = \arg \min_{p \in \mathcal{P}} \|\bar{q} - p\|^2,$$

where \bar{q} is the centroid of Q .

Proof. For a point $p \in \mathcal{P}$, we have

$$\begin{aligned} \text{Ed}(p, Q) &= \int_{\mathbb{R}^d} \|p - x\|^2 f_Q(x) dx \\ &= \|p\|^2 + \int_{\mathbb{R}^d} \|x\|^2 f_Q(x) dx - 2\langle p, \bar{q} \rangle. \end{aligned} \quad (2.1)$$

Observing that the second term in RHS of (2.1) is independent of p , we obtain

$$\begin{aligned} \arg \min_{p \in \mathcal{P}} \text{Ed}(p, Q) &= \arg \min_{p \in \mathcal{P}} \|p\|^2 - 2\langle p, \bar{q} \rangle \\ &= \arg \min_{p \in \mathcal{P}} \|p - \bar{q}\|^2, \end{aligned}$$

as claimed. □

The preprocessing step is to compute the Voronoi diagram $\text{VD}(\mathcal{P})$ of the points \mathcal{P} in time $O(n \log n)$. Once a query Q with a pdf of description complexity s is given, we compute its centroid \bar{q} in $O(s)$ time and find the nearest neighbor $\text{NN}(\mathcal{P}, \bar{q}) = \arg \min_{p \in \mathcal{P}} \|\bar{q} - p\|^2$ in $O(\log n)$ time by querying $\text{VD}(\mathcal{P})$ with q .

Theorem 2.3.6. Let \mathcal{P} be a set of n points in \mathbb{R}^2 . \mathcal{P} can be preprocessed in $O(n \log n)$ time into an index of size $O(n)$ so that, for a query point Q with a pdf of description complexity s , $\varphi(\mathcal{P}, Q)$, under the squared Euclidean distance, can be computed in $O(\log n + s)$ time.

Remarks. The algorithm extends to higher dimensions but the query time becomes $O(n^{1-1/\lceil d/2 \rceil + \delta} + s)$ for any $\delta > 0$ [13]. Alternatively, a query can be answered in $O(\log n + s)$ time using $O(n^{\lceil d/2 \rceil})$ space.

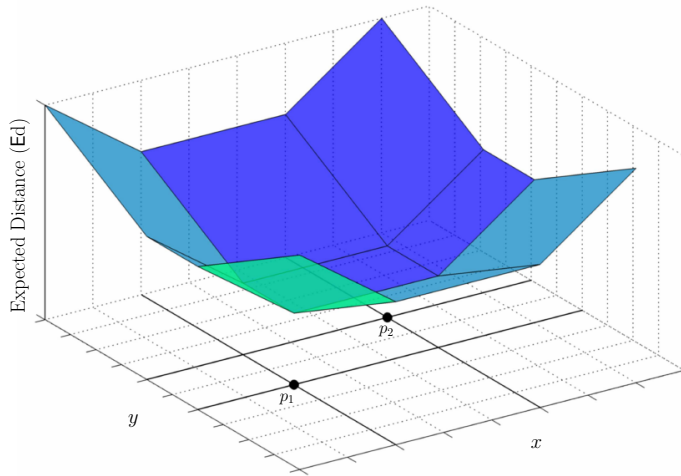


FIGURE 2.4. $\text{Ed}(P_i, q)$ when the uncertain point P_i is composed of two points p_1 and p_2 with probabilities 0.5 each. The grid induced by H_i^- and H_i^+ is shown below and $\text{Ed}(P_i, q)$ is linear within each rectangle of B_i .

Answering k -ENN queries. To generalize to k -ENN queries, we preprocess \mathcal{P} into an index Γ of size $O(n)$ in $O(n \log n)$ expected time for answering k nearest neighbor queries in $O(\log n + k)$ time, as in [2]. Once a query Q with a pdf of description complexity s is given, we compute its centroid \bar{q} in $O(s)$ time and find its k nearest neighbors in $O(\log n + k)$ time by querying Γ with \bar{q} . We conclude the following.

Theorem 2.3.7. *Let \mathcal{P} be a set of n points in \mathbb{R}^2 . \mathcal{P} can be preprocessed in $O(n \log n)$ expected time into an index of size $O(n)$ so that, for an uncertain point with description complexity s , a k -ENN query under the squared Euclidean distance, can be computed in $O(\log n + s + k)$ time.*

2.4. Rectilinear Metric

In this section we assume the distance function to be the L_1 metric. That is, for any two points $p = (p_x, p_y)$ and $q = (q_x, q_y)$,

$$d(p, q) = |p_x - q_x| + |p_y - q_y|.$$

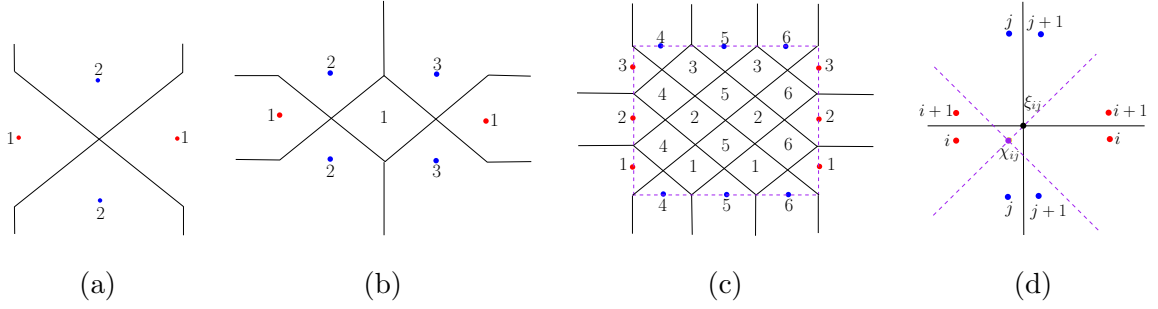


FIGURE 2.5. Lower bound construction for EVD. EVD of (a) 2 points, (b) 3 points, (c) 6 points on the boundary of a square σ . (d) Bisectors of (P_i, P_j) , (P_i, P_{i+1}) , and (P_j, P_{j+1}) .

The results in this section also hold for the L_∞ metric, i.e., when $d(p, q) = \max\{|p_x - q_x|, |p_y - q_y|\}$. We first consider the case when the input is a set of n uncertain points in \mathbb{R}^2 , each with a discrete pdf, and the query is a certain point, and then consider the case when the input is a set of certain points and the query is an uncertain point with a discrete pdf.

2.4.1. Uncertain data

Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n uncertain points in \mathbb{R}^2 , each with a discrete pdf of size s as described above. We first prove a lower bound on the complexity of $\text{EVD}(\mathcal{P})$ and then present a near-linear-size index to answer ENN queries.

Expected Voronoi diagram. Fix a point $P_i = \{p_{i,1}, p_{i,2}, \dots, p_{i,s}\}$ of \mathcal{P} . Let H_i^- (resp. H_i^+) be the set of s horizontal (resp. vertical) lines in \mathbb{R}^2 passing through the points of P_i . Let \mathcal{B}_i be the set of $O(s^2)$ rectangles in the grid induced by the lines in $H_i^- \cup H_i^+$. It can be checked that $\text{Ed}(P_i, q)$ is a linear function f_\square within each rectangle \square of \mathcal{B}_i ; see Figure 2.4. For each $\square \in \mathcal{B}_i$, let \square^\uparrow be the rectangle in \mathbb{R}^3 formed by restricting the graph of f_\square with \square , i.e.,

$$\square^\uparrow = \{(x, y, f_\square(x, y)) \mid (x, y) \in \square\}.$$

Let $\mathcal{B}_i^\uparrow = \{\square^\uparrow \mid \square \in \mathcal{B}_i\}$. By definition, the rectangles in \mathcal{B}_i^\uparrow form the graph of the

function $\text{Ed}(P_i, q)$. Set $\mathcal{B} = \bigcup_{i=1}^n \mathcal{B}_i$ and $\mathcal{B}^\dagger = \bigcup_{i=1}^n \mathcal{B}_i^\dagger$. By construction and the discussion in Section 2, $\text{EVD}(\mathcal{P})$ is the minimization diagram $\mathbb{M}(\mathcal{B}^\dagger)$ of \mathcal{B}^\dagger . We prove an almost tight bound on the complexity of $\text{EVD}(\mathcal{P})$.

Theorem 2.4.1. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 , each with a discrete pdf consisting of s points, and let $d(\cdot, \cdot)$ be the L_1 metric. Then the complexity of $\text{EVD}(\mathcal{P})$ is $O(s^2 n^2 \alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function. Moreover, there is a set \mathcal{P} of n uncertain points in \mathbb{R}^2 with $s = 2$ such that $\text{EVD}(\mathcal{P})$ has $\Omega(n^2)$ size.*

Proof. We first prove the upper bound. Set $H^- = \bigcup_{i=1}^n H_i^-$ and $H^\dagger = \bigcup_{i=1}^n H_i^\dagger$; $|H^-| = |H^\dagger| = ns$. We sort the lines in H^- by their y values and choose a subset G^- of s lines by selecting every n th line. Let G^\dagger be a similar subset of H^\dagger . Let \mathcal{R} be the set of rectangles in the non-uniform grid formed by the lines in $G^- \cup G^\dagger$. For each rectangle $R \in \mathcal{R}$, let $\mathcal{B}_R \subseteq \mathcal{B}$ be the set of rectangles whose boundaries intersect R , and let $\overline{\mathcal{B}}_R \subseteq \mathcal{B}$ be the set of rectangles that contain R . Since R lies between two adjacent lines of G^- and G^\dagger , at most $2n$ lines of $H^- \cup H^\dagger$ intersect R , implying that $|\mathcal{B}_R| \leq 2n$. $|\overline{\mathcal{B}}_R| \leq n$ because at most one rectangle of \mathcal{B}_i can contain R for any $1 \leq i \leq n$. Set $\chi_R = \{\square^\dagger \mid \square \in \mathcal{B}_R \cup \overline{\mathcal{B}}_R\}$. We note that $\mathbb{M}(\mathcal{B}^\dagger) \cap R = \mathbb{M}(\chi_R) \cap R$. Since $|\chi_R| \leq 3n$, the complexity of $\mathbb{M}(\chi_R)$ is $O(n^2 \alpha(n))$ (cf. Section 2.2). The $O(s^2)$ rectangles in \mathcal{R} tile the entire plane, therefore the complexity of $\mathbb{M}(\mathcal{B}^\dagger)$, and thus of $\text{EVD}(\mathcal{P})$, is $O(s^2 n^2 \alpha(n))$.

Next, we show that there exists a set \mathcal{P} of n uncertain points in \mathbb{R}^2 with $s = 2$ such that $\text{EVD}(\mathcal{P})$ has $\Omega(n^2)$ size. Assume that $n = 2m$ for some positive integer m . Each point $P_i \in \mathcal{P}$ has two possible locations p_{i1} and p_{i2} , each with probability 0.5. All the points lie on the boundary of the square $\sigma = [0, 2m]^2$ (see Figure 2.5). More specifically, for $1 \leq i \leq m$, the two possible locations of P_i are $p_{i1} = (0, 2i - 1)$ and $p_{i2} = (2m, 2i - 1)$. For $1 \leq j \leq m$, the two possible locations of P_{m+j} are $p_{m+j,1} = (2j - 1, 0)$ and $p_{m+j,2} = (2j - 1, 2m)$. That is, the two sites of each point in

P_1, \dots, P_m lie on the left and right edges of σ , and the two sites of each of P_{m+1}, \dots, P_n lie on the top and bottom edges of σ .

We claim that $\text{EVD}(\mathcal{P})$ has $\Omega(n^2)$ size. Notice that for any pair $1 \leq i \leq m < j \leq 2m$, the bisector of P_i and P_j inside the square σ consists of two lines: $y = x + 2(m + i - j)$ and $y = -x + 2(i + j - m - 1)$ (see Figure 2.5(d), dashed lines). Let χ_{ij} be the intersection point of these two lines. We observe that $\text{Ed}(P_i, \chi_{ij}) = \text{Ed}(P_j, \chi_{ij}) = m$ and $\text{Ed}(P_k, \chi_{ij}) > m$ for all $k \notin \{i, j\}$, implying that χ_{ij} bounds $\text{EVor}(P_i)$ and $\text{EVor}(P_j)$ on $\text{EVD}(\mathcal{P})$. Since χ_{ij} is the common vertex of two bisectors of $\text{EVor}(P_i)$ and $\text{EVor}(P_j)$, hence χ_{ij} is a vertex of $\text{EVD}(\mathcal{P})$. Similarly, for all $1 \leq i < m$, the bisector of P_i and P_{i+1} is the line $y = 2i$, and for all $m < j < 2m$, the bisector of P_j and P_{j+1} is the line $x = 2j - 2m$ (see Figure 2.5(d), solid lines). The intersection point of these two bisectors, $\xi_{ij} = (2j - 2m, 2i)$, is also a vertex of $\text{EVD}(\mathcal{P})$: $\text{Ed}(P_i, \xi_{ij}) = \text{Ed}(P_{i+1}, \xi_{ij}) = \text{Ed}(P_j, \xi_{ij}) = \text{Ed}(P_{j+1}, \xi_{ij}) = m + 0.5 < \text{Ed}(P_k, \xi_{ij})$, for all $k \notin \{i, i + 1, j, j + 1\}$, implying that ξ_{ij} is a common vertex of $\text{EVor}(P_i)$, $\text{EVor}(P_{i+1})$, $\text{EVor}(P_j)$ and $\text{EVor}(P_{j+1})$. Hence $\text{EVD}(\mathcal{P})$ has $\Omega(n^2)$ vertices inside σ . \square

Remarks. By preprocessing $\text{EVD}(\mathcal{P})$ for point-location queries [52, 108], an ENN query can be answered in $O(\log n)$ time using $O(s^2 n^2 \alpha(n))$ space. For higher dimensions, the complexity of $\text{EVD}(\mathcal{P})$ is $O(s^d n^d \alpha(n))$.

Near-linear size index. Next we show that despite the size of $\text{EVD}(\mathcal{P})$ being $\Omega(n^2)$, an index of size $O(s^2 n \log^2 n)$ can be constructed so that an ENN query can be answered in $O(\log^3(sn))$ time. Roughly speaking, we follow the same construction as to prove the upper bound on $\text{EVD}(\mathcal{P})$, but we proceed in a hierarchical manner to keep the size of the index under control.

For a query point $q \in \mathbb{R}^2$, let l_q be the line parallel to the z -axis passing through q , and oriented in the $(+z)$ -direction. Then $\varphi(\mathcal{P}, q)$ is P_i if the first rectangle of \mathcal{B}^\uparrow that l_q intersects belongs to \mathcal{B}_i^\uparrow . We label each rectangle \square^\uparrow in \mathcal{B}_i^\uparrow with i and

build an index on \mathcal{B}^\dagger so that the first rectangle intersected by a line parallel to the z -axis can be reported quickly. The index works in two stages. In the first stage, it builds a family $\mathcal{F} = \{C_1, C_2, \dots, C_u\}$ of *canonical subsets* of \mathcal{B} , i.e., for each $1 \leq i \leq u$, $C_i \subseteq \mathcal{B}$, so that for a query point $q \in \mathbb{R}^2$, the subset $\mathcal{B}_q \subseteq \mathcal{B}$ of rectangles containing q can be represented as the union of $O(\log^2(sn))$ canonical subsets of \mathcal{F} . That is, there exists a subset $\mathcal{F}_q \subseteq \mathcal{F}$ of size $O(\log^2 n)$ such that $\mathcal{B}_q = \bigcup_{C \in \mathcal{F}_q} C$. Furthermore, $\sum_{i \geq 1} |C_i| = O(s^2 n \log^2 n)$ [7]. Next, for a rectangle $\square \in \mathcal{B}$, let γ_\square be the plane containing the rectangle \square^\dagger , i.e., the graph of the linear function f_\square . For each $1 \leq i \leq u$, set $T_i = \{\gamma_\square \mid \square \in C_i\}$. We compute the minimization diagram (cf. Section 2.2) of T_i , which is a convex planar subdivision of size $O(|T_i|)$ and preprocess it for point-location queries [52].

Given a query point $q \in \mathbb{R}^2$, we first compute \mathcal{F}_q in $O(\log^2(sn))$ time, then for each canonical subset C_i in \mathcal{F}_q , we perform a point-location query using q on the minimization diagram of T_i to find out the first plane of T_i intersected by a vertical line l_q in $O(\log n)$ time, and finally we choose the best candidate among them. The total query time is $O(\log^3(sn))$. We conclude the following.

Theorem 2.4.2. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 , each with a discrete pdf consisting of s points, and let $d(\cdot, \cdot)$ be the L_1 or L_∞ metric. \mathcal{P} can be stored in an index of size $O(s^2 n \log^2 n)$, so that an ENN query can be answered in $O(\log^3(sn))$ time. The index can be built in $O(s^2 n \log^3 n)$ time.*

Remarks. The algorithm extends to higher dimension. Namely, we use a higher-dimensional range tree so that the set \mathcal{F}_q can be represented as a union of $O(\log^d(sn))$ canonical subsets. For each canonical subset C_i , the first hyperplane of T_i intersected by a vertical line can be computed in $O(n^{1-1/\lceil d/2 \rceil} \log^{O(d)} n)$ time using $O(n)$ space [13]. Putting everything together and omitting further details, we conclude

the following. For $d \geq 3$, the size of the index is $O(s^d n \log^d n)$ and the query time is $O(n^{1-1/\lceil d/2 \rceil} \log^{O(d)}(sn))$.

Answering k -ENN queries. The above index can be generalized to k -ENN queries. The only part we need to modify is the second stage, where we build an index of linear size on T_i to report the first k planes, instead of the first one, of T_i intersected by a vertical line l_q in $O(\log n + k)$ time [2]. By querying each canonical subset in \mathcal{F}_q with this structure, we compute a set of $O(k \log^2(sn))$ points of \mathcal{P} that contain the k -ENN of q . We extract the k -ENN from this set in an additional $O(k \log^2(sn))$ time. So a k -ENN query takes $O(\log^3(sn) + k \log^2(sn))$ time. We sketch below how the query time can be improved to $O(\log^3(sn) + k \log(s \log n + k))$.

We note that the data structure described in [2] reports the k planes in batches: first 2^0 plane, next 2^1 planes, next 2^2 planes, \dots , next $2^{\lceil \log k \rceil}$ planes. Instead of retrieving all the k planes for each canonical subset of \mathcal{F}_q at once, we retrieve the planes according to the batches as needed. Initially, we only retrieve the first plane for each canonical subset. We store all the possible candidate planes retrieved in a priority queue \mathcal{Q} , with $f_{\square}(q)$ as the key for each plane γ_{\square} . At each step, the algorithm performs the DELETETEMIN operation on \mathcal{Q} , which reports (and removes from \mathcal{Q}) the lowest plane intersected by l_q among all the candidate planes in \mathcal{Q} . If the DELETETEMIN operation on \mathcal{Q} returns a plane from T_i for the canonical subset C_i and \mathcal{Q} does not contain any more planes of T_i , we retrieve the next batch of planes from the Afshani-Chan data structure built on T_i and add them to \mathcal{Q} . The retrieval process stops as soon as k planes have been reported or \mathcal{Q} becomes empty.

Recall that $\mathcal{F}_q = O(\log^2(sn))$. Our algorithm ensures that if it reports k_i planes from T_i for the canonical subset $C_i \in \mathcal{F}_q$, then it fetches at most $2k_{i+1}$ planes from T_i through the Afshani-Chan data structure built on T_i . Hence, the total number of planes retrieved over all canonical subsets is $O(k + \log^2(sn))$. Since

each DELETEMIN operation takes $O(\log(\log(sn) + k))$ time, the overall query time is $O(\log^3(sn) + k \log(\log(sn) + k))$. Therefore, we conclude the following.

Theorem 2.4.3. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 , each with a discrete pdf consisting of s points, and let $d(\cdot, \cdot)$ be the L_1 or L_∞ metric. \mathcal{P} can be stored in an index of size $O(s^2 n \log^2 n)$ so that a k -ENN query can be answered in $O(\log^3(sn) + k \log(\log(sn) + k))$ time. The index can be built in $O(s^2 n \log^3 n)$ time.*

Remarks. The algorithm extends to \mathbb{R}^d , $d \geq 3$. The size of the index will be $O(s^d n \log^d(sn))$ and the query time will be $O(n^{1-1/\lceil d/2 \rceil} \log^{O(d)} n + k \log^d(sn))$ [13]. We note that we write a weaker bound for the query time and do not use the improved procedure, just described, for reporting the points from different canonical subsets.

2.4.2. Uncertain query

Let $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$ be a set of n certain input data points in \mathbb{R}^2 . We first build an index such that the ENN of an uncertain query point Q , which is represented as a discrete pdf of s points, can be found quickly.

Given an uncertain query Q , which has a discrete pdf of s points $\{q_1, \dots, q_s\}$ with associated probabilities $\{w_1, \dots, w_s\}$, let H^- (resp. $H^|$) be the set of s horizontal (resp. vertical) lines in \mathbb{R}^2 passing through the points of Q . Let \mathcal{B} be the set of $O(s^2)$ rectangles in the grid induced by the lines in $H^- \cup H^|$ (see Figure 2.6(a)).

Fix a rectangle $\square \in \mathcal{B}$, let $\mathcal{P}_\square = \mathcal{P} \cap \square$. Let w_{nw} denote the sum of the probabilities of points of Q which are above and to the left of \square . We similarly define w_{ne} , w_{sw} , w_{se} for points of Q which are at top-right, bottom-left and bottom-right of \square . We call these regions the quadrants of \square ; see Figure 2.6(b).

Lemma 2.4.4. *For every point $p = (x_p, y_p) \in \square$,*

$$\text{Ed}(p, Q) = w_x x_p + w_y y_p + c,$$

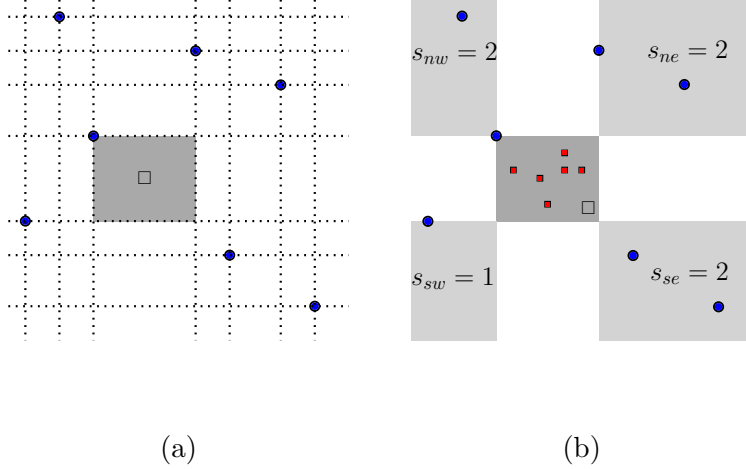


FIGURE 2.6. (a) The set of rectangles \mathcal{B} induced by horizontal lines H^- and vertical lines H^+ through the points of Q . A single rectangle \square is also shown. (b) The four quadrants of \square are shown along with the number of points of Q in each. The points \mathcal{P}_\square in \square are shown as red squares.

where $w_x = w_{nw} + w_{sw} - w_{ne} - w_{se}$, $w_y = w_{sw} + w_{se} - w_{nw} - w_{ne}$ and c is independent of p .

Proof. Note that no point of Q lies vertically above or below, or horizontally to the left or right of \square . Let $v_{nw} = (x_{nw}, y_{nw})$ (resp. v_{ne}, v_{sw}, v_{se}) denote the top-left (resp. top-right, bottom-left, bottom-right) corner of \square . Let Q_{nw}, Q_{ne}, Q_{sw} and Q_{se} denote the points of Q that lie in the top-left, top-right, bottom-left and bottom-right quadrants of \square respectively, and let $q \in Q_{nw}$. Then $d(p, q) = d(p, v_{nw}) + d(v_{nw}, q)$. Thus the total contributions of points of Q in this quadrant to $\text{Ed}(p, Q)$ is

$$\sum_{q \in Q_{nw}} d(q, v_{nw}) + w_{nw} d(v_{nw}, p) = \sum_{q \in Q_{nw}} d(q, v_{nw}) + w_{nw}(x_p - x_{nw}) + w_{nw}(y_{nw} - y_p).$$

Similar expressions hold for the remaining quadrants. Thus, by summing over all quadrants, the lemma follows. □

Lemma 2.4.5. *Let*

$$p^* = \arg \min_{p \in \mathcal{P}_\square} \text{Ed}(p, Q).$$

Then p^ is a vertex of the convex hull $\text{conv}(\mathcal{P}_\square)$ of \mathcal{P}_\square .*

Proof. By Lemma 2.4.4, p^* is an extreme point of \mathcal{P}_\square minimizing a linear function of $p \in \mathcal{P}_\square$. Thus, without loss of generality, it realizes its minimum when p^* is a vertex of $\text{conv}(\mathcal{P}_\square)$. \square

Preprocessing step. Our index is simply a two dimensional range-tree on the points in \mathcal{P} [52] with a single modification to enable efficient ENN queries. The range-tree consists of two levels. We first construct a balanced binary tree \mathcal{T} on the x -coordinates of the points of \mathcal{P} . We call this the *primary tree*. Its leaves store the points of \mathcal{P} in sorted x -order from left to right, and internal nodes store splitting values to guide the search. Each node v in \mathcal{T} is associated with the subset $S_v \subseteq \mathcal{P}$ of points that are stored at the leaves in the subtree rooted at v . For each node v in the tree, a similar balanced binary tree \mathcal{T}_v is constructed on the y -coordinates of S_v . We call these *secondary trees*. Each node u in a secondary tree \mathcal{T}_v corresponding to a node v in \mathcal{T} is associated with the subset $S_{uv} \subseteq S_v$ of points stored at leaves in the subtree of \mathcal{T}_v rooted at u . All such subsets are called *canonical subsets*. Given a query rectangle, the points of \mathcal{P} in the rectangle are reported as the disjoint union of $O(\log^2 n)$ canonical subsets. See [52] for more details on range-trees.

We make the following modification to the range-tree structure. For any canonical subset \mathcal{P}_u corresponding to a node u in a secondary tree \mathcal{T}_v , we store the convex hull $\text{conv}(\mathcal{P}_u)$ of the points of \mathcal{P}_u . For any secondary tree \mathcal{T}_v , the convex hull of the canonical subsets may be computed by performing a bottom-up traversal while merging the convex hulls of the children at any internal node. Thus, if there are m nodes in \mathcal{T}_v , the total time for constructing the convex hulls is $O(m \log m)$. Finally, we build a fractional-cascading data structure [52] on the convex hulls to expedite the search among them. The total preprocessing time and the space required for the index are $O(n \log^2 n)$.

Query step. When a query Q is given, we construct \mathcal{B} as above, and compute,

for each rectangle $\square \in \mathcal{B}$, the values w_{ne} , w_{nw} , w_{se} , and w_{sw} . Next, we perform a range query in \mathcal{T} , to find the points of \mathcal{P}_\square as the union of $O(\log^2 n)$ canonical subsets of \mathcal{P} . We find the point p^* by performing a binary search on the points on the convex hulls of each subset, using fractional cascading, and picking the minimum over all subsets. By Lemma 2.4.5, the point p^* must be among these points. The total time is $O(s^2 \log^2 n)$.

Theorem 2.4.6. *Let \mathcal{P} be a set of n points in \mathbb{R}^2 and let $d(\cdot, \cdot)$ be the L_1 or L_∞ metric. \mathcal{P} can be preprocessed, in $O(n \log^2 n)$ time, in an index of size $O(n \log^2 n)$, so that for an uncertain query Q as a discrete pdf with s points, its ENN under the L_1 or L_∞ metric can be reported in $O(s^2 \log^2 n)$ time.*

Remarks. (i) Theorem 2.4.6 has been improved by [124], where they showed that it is sufficient to check $O(s)$ rectangles, instead of all the $O(s^2)$ rectangles, of \mathcal{B} . As a result, a factor of $O(s)$ got shaved in the query time. They further showed that by using the compact interval tree [65], instead of building the convex hull explicitly, for each canonical subset, one can obtain an index of $O(n \log n \log \log n)$ size and preprocessing time such that an ENN query can be answered in $O(s \log^2 n)$ time.

(ii) This can be extended to \mathbb{R}^d by constructing a d -dimensional range tree and by preprocessing the convex hull of each canonical subset for point-location queries [52]. The space and preprocessing time become $O(n^{\lfloor d/2 \rfloor} \log^d n)$, and the query time becomes $O(s^d \log^{d+1} n)$. Alternatively, one can construct a linear-size data structure and answer a query in $O(s^d n^{1 - \frac{1}{\lfloor d/2 \rfloor} + \delta})$ time for any $\delta > 0$.

Answering k -ENN queries. We first make a further modification to the range-tree structure, as follows. For a point set X , the *convex layers* of X , denoted by $\mathcal{L}(X)$, is a partition of X into a sequence X_1, \dots, X_r of subsets with $\text{conv}(X_i) \subset \text{int conv}(X_{i-1})$, defined as follows. Set $Y_0 = X$, and for $i \geq 1$, $X_i = Y_{i-1} \cap \partial \text{conv}(Y_{i-1})$, and

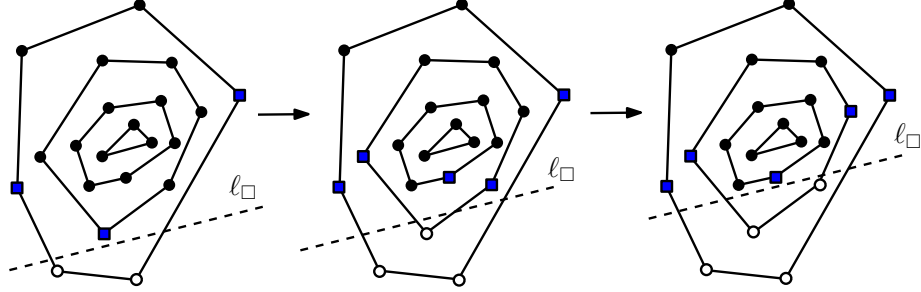


FIGURE 2.7. Hollow circles have been visited; Squares are event points.

$Y_i = Y_{i-1} \setminus X_i$. That is, $\mathcal{L}(X)$ is obtained by repeatedly peeling away the points that lie on the boundary of the convex hull. See Figure 2.7. $\mathcal{L}(X)$ can be constructed in $O(|X| \log |X|)$ time [36]. For each canonical subset in the 2D range tree, we construct its convex layers. We also preprocess all of these convex layers for fractional cascading [40]. The total size of the index remains $O(n \log^2 n)$, but the preprocessing time becomes $O(n \log^3 n)$.

We now describe the query procedure. Let \mathcal{B} be as above. We query the range tree for each rectangle $\square \in \mathcal{B}$. Let $f_{\square}(x, y) = w_x x + w_y y + c$ be the linear function as described in Lemma 2.4.4. Let \mathcal{F}_{\square} be the family of canonical subsets reported for $\mathcal{P} \cap \square$. Fix a canonical subset $\mathcal{P}_u \in \mathcal{F}_{\square}$. We describe how to report the points of \mathcal{P}_u in the increasing order of their expected distances from Q , i.e., in increasing order of the values $\{f_{\square}(p) \mid p \in \mathcal{P}_u\}$.

We achieve this by sweeping a line ℓ_{\square} in direction (w_x, w_y) from $-\infty$ to $+\infty$. We note that the i -th point of \mathcal{P}_u encountered during the sweep is $\varphi_i(\mathcal{P}_u, Q)$, i -th ENN of Q in \mathcal{P}_u . It thus suffices to show how to perform the sweep efficiently without explicitly sorting the points of \mathcal{P}_u in direction (w_x, w_y) . This is where $\mathcal{L}(\mathcal{P}_u)$ plays an important role. Suppose the sweep line ℓ_{\square} is currently intersecting the first i layers of \mathcal{P}_u . We define the *event points* to be the endpoints of the two edges of each layer that ℓ_{\square} is intersecting but not encountered so far plus the minimal point of layer $i + 1$ in direction (w_x, w_y) (squares in Figure 2.7). We store these $2i + 1$ event points in a

priority queue \mathcal{Q} , with $\langle p, (w_x, w_y) \rangle$, i.e., $f_{\square}(p) - c$, as the key of a point p . It can be checked that the next point of \mathcal{P}_u encountered by ℓ_{\square} is one of the event points.

At each step, the algorithm performs the DELETEMIN operation on \mathcal{Q} , which reports the event point p with the minimum expected distance from Q . We report p . If p is not the last point of its convex layer, say, layer i , we add the point(s) adjacent to p in that layer to \mathcal{Q} — if p is not the first point of layer i encountered by ℓ_{\square} , then only one point is added to \mathcal{Q} , otherwise two points are added. If p is the first point of layer i and layer $i + 1$ exists, then we also add the minimal point of layer $i + 1$ in direction (w_x, w_y) to \mathcal{Q} .

Returning to our overall query procedure, we perform the above sweep simultaneously at all canonical subsets in \mathcal{F}_{\square} , and for all $\square \in \mathcal{B}$. We store all event points, of all canonical subsets, in a single global priority queue \mathcal{Q} . If the DELETEMIN operation on \mathcal{Q} returns a point from a canonical subset \mathcal{P}_u of \mathcal{F}_{\square} , we advance the sweep over \mathcal{P}_u as described above. The sweep stops as soon as k points have been reported or \mathcal{Q} becomes empty.

Recall that there are a total of $O(s^2 \log^2 n)$ canonical subsets that are reported by the query procedure. The total number of layers swept by the sweep-line procedure over all canonical subsets is $O(k + s^2 \log^2 n)$. Hence, each DELETEMIN operation takes $O(\log(s \log n + k))$ time, implying that the overall query time is $O(s^2 \log^2 n + k \log(s \log n + k))$.

Theorem 2.4.7. *Let \mathcal{P} be a set of n points in \mathbb{R}^2 and let $d(\cdot, \cdot)$ be the L_1 or L_{∞} metric. \mathcal{P} can be preprocessed, in $O(n \log^3 n)$ time, in an index of size $O(n \log^2 n)$, so that for an uncertain query Q as a discrete pdf with s points, the k -ENN of Q can be reported in $O(s^2 \log^2 n + k \log(s \log n + k))$ time.*

2.5. Euclidean Distance

We now consider the case when $d(\cdot, \cdot)$ is the Euclidean distance. For any two points $a, b \in \mathbb{R}^2$, we use $\|a - b\|$ to denote the Euclidean distance $d(a, b)$. Unfortunately, the expected distance under Euclidean metric is an expression containing the sum of radicals (square roots), so the algebraic complexity of a bisector, and thus of a Voronoi cell, is quite large (see e.g. [29]). We thus focus on answering ε -ENN queries in this section. We first describe an algorithm for computing a function that approximates the expected distance from a fixed uncertain point to any point in \mathbb{R}^2 . The construction is similar to the one given in [11]. We use this algorithm to answer ε -ENN queries: first the input being a set of uncertain points but the query a fixed point, and then, the input being a set of points in \mathbb{R}^2 but the query an uncertain point. In the former case, we construct an approximate expected Voronoi diagram of \mathcal{P} .

2.5.1. Approximation of the expected Euclidean distance

Let P be an uncertain point in \mathbb{R}^2 , and let $f_P : \mathbb{R}^2 \rightarrow \mathbb{R}_{\geq 0}$ be its pdf³. Let the description complexity of f_P be s . We construct a function $g_P : \mathbb{R}^2 \rightarrow \mathbb{R}_{\geq 0}$ of description complexity $O((1/\varepsilon^2) \log(1/\varepsilon))$ such that for any $x \in \mathbb{R}^2$,

$$\text{Ed}(P, x) \leq g_P(x) \leq (1 + \varepsilon)\text{Ed}(P, x).$$

Let \bar{p} be the centroid of P . The following two lemmas follow from the triangle inequality.

Lemma 2.5.1. *For any two points $a, b \in \mathbb{R}^2$,*

$$|\text{Ed}(P, a) - \text{Ed}(P, b)| \leq \|a - b\|.$$

³For simplicity, we focus on P having a continuous pdf; the argument holds for a discrete pdf as well.

Proof.

$$\begin{aligned}
|\text{Ed}(P, a) - \text{Ed}(P, b)| &= \int_{\mathbb{R}^2} f_P(x) \left| \|x - a\| - \|x - b\| \right| dx \\
&\leq \int_{\mathbb{R}^2} f_P(x) \|a - b\| dx \\
&= \|a - b\|.
\end{aligned}$$

□

Lemma 2.5.2. $\text{Ed}(P, \bar{p}) \leq 2 \min_{x \in \mathbb{R}^2} \text{Ed}(P, x)$, where \bar{p} is the centroid of P .

Proof. Let $p_{\min} = \arg \min_{x \in \mathbb{R}^2} \text{Ed}(P, x)$. By Lemma 2.5.1,

$$\begin{aligned}
|\text{Ed}(P, \bar{p}) - \text{Ed}(P, p_{\min})| &\leq \|\bar{p} - p_{\min}\| \\
\text{have, } &= \left\| p_{\min} - \int_{\mathbb{R}^2} x f_P(x) dx \right\| \\
&= \left\| \int_{\mathbb{R}^2} f_P(x) (x - p_{\min}) dx \right\| \\
&\leq \int_{\mathbb{R}^2} f_P(x) \|x - p_{\min}\| dx \\
&= \text{Ed}(P, p_{\min}).
\end{aligned}$$

The lemma now follows. □

Lemma 2.5.3. Let $0 < \varepsilon < 1$ be a parameter, let $\bar{p} = \text{Ed}(P, \bar{p})$, and for any $x \in \mathbb{R}^2$, let

$$g(x) = \|x - \bar{p}\| + \bar{p}.$$

Then for any point $q \in \mathbb{R}^2$ with $\|q - \bar{p}\| > 8\bar{p}/\varepsilon$,

$$\text{Ed}(P, q) \leq g(q) \leq (1 + \varepsilon)\text{Ed}(P, q).$$

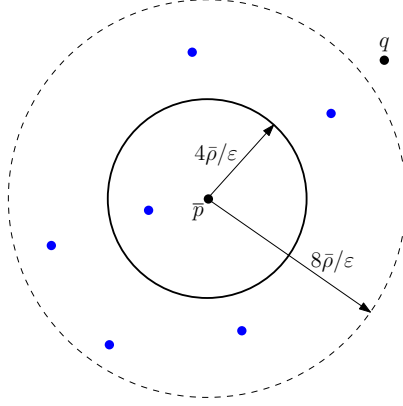


FIGURE 2.8. An illustration for the proof of Lemma 2.5.3.

Proof. Let $q \in \mathbb{R}^2$ be a point with $\|q - \bar{p}\| > 8\bar{\rho}/\varepsilon$. See Figure 2.8. By Lemma 2.5.1,

$$\text{Ed}(P, q) \leq \text{Ed}(P, \bar{p}) + \|q - \bar{p}\| = \bar{\rho} + \|q - \bar{p}\| \leq g(q).$$

Similarly,

$$\text{Ed}(P, q) \geq \|q - \bar{p}\| - \text{Ed}(P, \bar{p}) = \|q - \bar{p}\| - \bar{\rho}.$$

Therefore

$$g(q) = \|q - \bar{p}\| + \bar{\rho} \leq \text{Ed}(P, q) + 2\bar{\rho}. \quad (2.2)$$

Let D be the disk of radius $4\bar{\rho}/\varepsilon$ centered at \bar{p} . For any point $x \notin D$, $\|x - \bar{p}\| > 4\bar{\rho}/\varepsilon$.

Hence,

$$\begin{aligned} \bar{\rho} &= \int_{\mathbb{R}^2} \|x - \bar{p}\| f_P(x) dx \geq \int_{\mathbb{R}^2 \setminus D} \|x - \bar{p}\| f_P(x) dx \\ &\geq \frac{4\bar{\rho}}{\varepsilon} \int_{\mathbb{R}^2 \setminus D} f_P(x) dx, \end{aligned}$$

implying that

$$\int_{\mathbb{R}^2 \setminus D} f_P(x) dx \leq \varepsilon/4 \quad \text{and} \quad \int_D f_P(x) dx \geq 1 - \varepsilon/4.$$

On the other hand, for any point $x \in D$, $\|x - q\| > 4\bar{\rho}/\varepsilon$. Therefore

$$\begin{aligned} \text{Ed}(P, q) &= \int_{\mathbb{R}^2} \|x - q\| f_P(x) dx \geq \int_D \|x - q\| f_P(x) dx \\ &\geq \frac{4\bar{\rho}}{\varepsilon} \int_D f_P(x) dx \geq \frac{4\bar{\rho}}{\varepsilon} (1 - \varepsilon/4) \geq \frac{2\bar{\rho}}{\varepsilon}, \end{aligned}$$

which implies that $\bar{\rho} \leq \varepsilon \text{Ed}(P, q)/2$. Substituting this in (2.2),

$$g(q) \leq (1 + \varepsilon) \text{Ed}(P, q).$$

□

We are now ready to describe the function g_P . $\bar{\rho} = \text{Ed}(P, \bar{p})$. For a point $x \in \mathbb{R}^2$ and a value $r \geq 0$, let $B(x, r)$ denote the square of side length $2r$ centered at x . Let $l = \lceil \log_2(8/\varepsilon) \rceil$. For $0 \leq i \leq l$, set $B_i = B(\bar{p}, \bar{\rho}2^i)$; set $B_{-1} = \emptyset$. Finally, set $\rho_i = \varepsilon 2^i \bar{\rho}/8$.

We cover B_l by at most four congruent canonical squares C_1, \dots, C_4 of side length at most $2\rho_l = \bar{\rho}2^{l+1}$. The union of C_1, \dots, C_4 is also a square C ; see Figure 2.9. We set

$$g_P(x) = \|x - \bar{p}\| + \bar{\rho}, \quad \forall x \notin C.$$

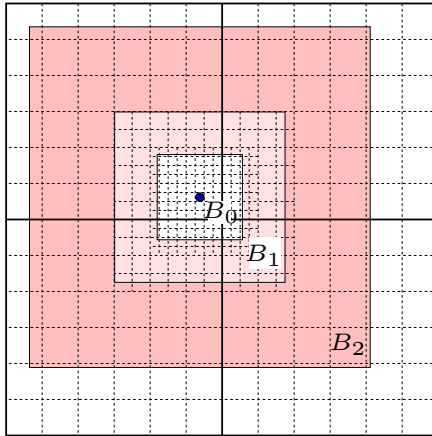


FIGURE 2.9. Covering B_l with four canonical squares and drawing an exponential grid composed of canonical squares.

For $1 \leq j \leq 4$ and $0 \leq i \leq l$, we cover $C_j \cap (B_i \setminus B_{i-1})$ with canonical squares of size 2^{Δ_i} where $\Delta_i = \lfloor \log_2 \rho_i \rfloor$; see Figure 2.9. For each such square \square , let a_\square be its center and set

$$\delta_\square = \text{Ed}(P, a_\square) + 4 \cdot 2^{\Delta_i}.$$

Finally, we also cover $C \setminus B_l$ with canonical squares of size 2^{Δ_l} and set δ_\square as above. By construction, any point $x \in C$ is covered with some canonical square, which is further associated with a constant value. Let \mathcal{B} be the resulting set of $O((1/\varepsilon^2) \log(1/\varepsilon))$ canonical squares. We construct a compressed quadtree \mathcal{T} on (\mathcal{B}, C) as described in Section 2.2. It can be checked that each exposed node on \mathcal{T} is a leaf and therefore the rectilinear subdivision of C induced by $\mathbb{E} = \mathbb{E}(\mathcal{B}, C)$ is composed of canonical squares. If a square σ in \mathbb{E} lies in multiple squares of \mathcal{B} , we set $\delta_\sigma = \delta_\square$ where \square is the smallest square of \mathcal{B} containing σ . Finally, for every $\sigma \in \mathbb{E}$, we set

$$g_P(x) = \delta_\sigma, \quad \forall x \in \sigma.$$

Lemma 2.5.4. *Let P be an uncertain point in \mathbb{R}^2 with a pdf of description complexity s , and let $0 < \varepsilon < 1$ be a parameter. A function $g_P : \mathbb{R}^2 \rightarrow \mathbb{R}_{\geq 0}$ can be constructed in $O((s/\varepsilon^2) \log(1/\varepsilon))$ time such that*

- (i) g_P is piecewise constant inside a square C , which is the union of four canonical squares.
- (ii) Each piece of g_P inside C is defined over a canonical square, and the number of pieces is $O((1/\varepsilon^2) \log(1/\varepsilon))$.
- (iii) $C \supseteq B[\bar{p}, 8\text{Ed}(P, \bar{p})/\varepsilon]$ and $g_P(x) = \|x - \bar{p}\| + \text{Ed}(P, \bar{p})$ for $x \notin C$.
- (iv) $\text{Ed}(P, x) \leq g_P(x) \leq (1 + \varepsilon)\text{Ed}(P, x)$ for all $x \in \mathbb{R}^2$.

Proof. (i) and (ii) follow from the construction, and (iii) follows from Lemma 2.5.3, so we only need to prove (iv). We describe the proof for the case when $x \in B_0$, a

similar argument holds when $x \in B_i \setminus B_{i-1}$, for $i \geq 1$. Suppose x lies in a grid cell τ of B_0 . Then, using Lemma 2.5.1,

$$\begin{aligned} g_P(x) &= \text{Ed}(P, a_\tau) + 4 \cdot 2^{\Delta_0} \\ &\geq \text{Ed}(P, x) - \|x - a_\tau\| + 2\rho_0 \\ &\geq \text{Ed}(P, x). \end{aligned}$$

On the other hand,

$$\begin{aligned} g_P(x) &\leq \text{Ed}(P, x) + \|x - a_\tau\| + 4 \cdot 2^{\Delta_0} \\ &\leq \text{Ed}(P, x) + 2\rho_0 + 4\rho_0 \\ &\leq \text{Ed}(P, x) + \frac{3\varepsilon}{4}\bar{\rho} \\ &\leq (1 + \varepsilon)\text{Ed}(P, x). \end{aligned}$$

□

Remarks. We remark that a similar function can be constructed that approximates $\text{Ed}(P, x)$ even if $d(\cdot, \cdot)$ is an L_p metric for $p \neq 2$.

2.5.2. Uncertain data, ε -ENN queries

Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n uncertain points in \mathbb{R}^2 , each with a pdf of description complexity s , and let $\varepsilon \in (0, 1)$ be a parameter. We describe a method for computing an ε -ENN of a query point $q \in \mathbb{R}^2$ in \mathcal{P} . For each $1 \leq i \leq n$, we construct the function $g_i : \mathbb{R}^2 \rightarrow \mathbb{R}_{\geq 0}$, using Lemma 2.5.4 with $\varepsilon/3$ as the approximation factor (instead of ε), so that

$$\text{Ed}(P_i, q) \leq g_i(q) \leq (1 + \varepsilon/3)\text{Ed}(P_i, q),$$

for all $q \in \mathbb{R}^2$. Let C_i be the canonical square inside which g_i is piecewise constant. Let $G = \{g_1, \dots, g_n\}$. By definition, the minimization diagram $\mathbb{M}(G)$ of G is an

ε -EVD(\mathcal{P}). Hence, it suffices to construct $\mathbb{M}(G)$ and build an index on $\mathbb{M}(G)$ for point-location queries. The difficulty with this approach is that we do not have a near-linear upper bound on the complexity of $\mathbb{M}(G)$ even in \mathbb{R}^2 . Moreover, the complexity of $\mathbb{M}(G)$ is $\Omega(n^{\lceil d/2 \rceil})$ in higher dimensions, so this approach will not be practical for $d \geq 3$. We circumvent this problem by using the ideas from Arya *et al.* [20] and constructing a different ε -EVD(\mathcal{P}) of near-linear size.

Here is the outline of the algorithm. We construct two sets \mathcal{B}_{in} and \mathcal{B}_{out} of canonical squares. Set $\mathcal{B} = \mathcal{B}_{\text{in}} \cup \mathcal{B}_{\text{out}}$. The size of \mathcal{B} , denoted by m , will be $O((n/\varepsilon^2) \log(1/\varepsilon))$, and it can be constructed in $O(n \log n + (n/\varepsilon^2) \log(1/\varepsilon))$ time. We build, in $O(m \log m)$ time, a compressed quadtree \mathbb{T} of size $O(m)$ on \mathcal{B} , and preprocess it in additional $O(m)$ time so that for a point $q \in \mathbb{R}^2$, the exposed node of \mathbb{T} containing q can be computed in $O(\log m)$ time. Let \mathbb{E} be the planar subdivision induced by the exposed nodes of \mathbb{T} . We refine each cell of \mathbb{E} into $O(1)$ faces to construct an ε -EVD of \mathcal{P} . More precisely, for a point $x \in \mathbb{R}^2$, let $\mathcal{P}_{\text{in}}[x] = \{P_i \mid x \in C_i\}$ and $\mathcal{P}_{\text{out}}[x] = \{P_i \mid x \notin C_i\}$. \mathbb{T} has the property that for every exposed node v , $\mathcal{P}_{\text{in}}[x]$ and $\mathcal{P}_{\text{out}}[x]$ are the same for all points in the region R_v . We denote these sets by $\mathcal{P}_{\text{in}}[v]$ and $\mathcal{P}_{\text{out}}[v]$.

We associate two representative points $P_v^{\text{in}} \in \mathcal{P}_{\text{in}}[v]$, $P_v^{\text{out}} \in \mathcal{P}_{\text{out}}[v]$ as follows: we choose an arbitrary point $z_v \in R_v$. P_v^{in} is an ε -ENN of z_v in $\mathcal{P}_{\text{in}}[v]$ and P_v^{out} is an ε -ENN of z_v in $\mathcal{P}_{\text{out}}[v]$. If $P_v^{\text{in}} = P_i$, we store the canonical square \square_v of the function g_i that contains R_v , and if $P_v^{\text{out}} = P_j$, we also store the centroid \bar{p}_j of P_j at v . For all $x \in R_v$, $g_i(x)$ is constant and $g_j(x) = \|x - \bar{p}_j\| + \text{Ed}(P_j, \bar{p}_j)$. The minimization diagram of g_i and g_j within R_v , denoted by Σ_v , has $O(1)$ size; see Figure 2.11(a). We compute Σ_v for all exposed nodes of \mathbb{T} . Σ_v 's induce a planar subdivision, denoted by Σ , which is a refinement of \mathbb{E} . We show that Σ is the desired ε -EVD of \mathcal{P} ; Figure 2.11(b) shows a section of such a planar subdivision.

We first describe the computation of \mathcal{B}_{in} and \mathcal{B}_{out} followed by the computation

of the representative points P_v^{in} and P_v^{out} for each exposed node v in \mathbb{T} . Then we describe how to construct an ε -EVD using the representative points. Finally, we prove that Σ is an ε -EVD of \mathcal{P} .

Constructing \mathcal{B}_{in} . For $1 \leq i \leq n$, let \mathcal{B}_i be the set of canonical squares that define the pieces of the piecewise-constant portion of g_i . Set $\mathcal{B}_{\text{in}} = \bigcup_{i=1}^n \mathcal{B}_i$. For each $\square \in \mathcal{B}_i$, we associate a value δ_\square with \square , which is $g_i(x)$ for any $x \in \square$. If a square \square appears in multiple \mathcal{B}_i 's, we keep only one copy of \square in \mathcal{B}_{in} and δ_\square is the minimum of the values associated with the different copies of \square . For each square $\square \in \mathcal{B}_i$, we set $P_\square = P_i$.

Constructing \mathcal{B}_{out} . \mathcal{B}_{out} is constructed using the algorithm by Arya *et al.* [20] for computing an ε -VD of a set S of N (certain) points. We therefore sketch their algorithm in \mathbb{R}^2 . Two point sets $A, B \subset \mathbb{R}^2$ are called α -well-separated if A and B can be contained in disks D_A and D_B respectively, whose centers are at distance ℓ and whose radii are at most $\alpha\ell$; see Figure 2.10(a). A partition of $S \times S - \{(p, p) \mid p \in S\}$ into a family $Z = \{(A_1, B_1), \dots, (A_M, B_M)\}$ of α -well-separated pairs is called an α -well-separated pair decomposition (α -WSPD) of S . It is well known that an α -WSPD of S with $O(N/\varepsilon^2)$ pairs can be computed in $O(N \log N + N/\varepsilon^2)$ time [70].

Arya *et al.* [20] first compute a $(1/8)$ -WSPD Z of S . Let (A_i, B_i) be a pair in Z . Without loss of generality, assume that A_i and B_i are contained in disks of radii $\ell/8$ centered at a_i, b_i , respectively, and the centers of these disks are at distance ℓ . Let $t = \lceil \log_2(1/\varepsilon) \rceil + 4$. They construct a family of $2t$ disks $D_1^{A_i}, D_1^{B_i}, \dots, D_t^{A_i}, D_t^{B_i}$ centered at a_i, b_i where the radius of $D_j^{A_i}, D_j^{B_i}$ is $r_j = 2^{j-3}\ell$. They cover $D_j^{A_i}, D_j^{B_i}$ with a set \mathcal{C}_j of canonical squares of the largest size but not exceeding $r_j \cdot \varepsilon/32$; $|\mathcal{C}_j| = O(1/\varepsilon^2)$. See Figure 2.10(b). Set $\tilde{\mathcal{B}}_i = \bigcup_{1 \leq j \leq t} \mathcal{C}_j$. The above procedure is repeated for each pair in Z . Let $\tilde{\mathcal{B}}$ be the overall set of canonical squares constructed; $|\tilde{\mathcal{B}}| = O((N/\varepsilon^2) \log(1/\varepsilon^2))$. $\tilde{\mathcal{B}}$ can be constructed in $O((N/\varepsilon^2) \log(1/\varepsilon) + N \log N)$

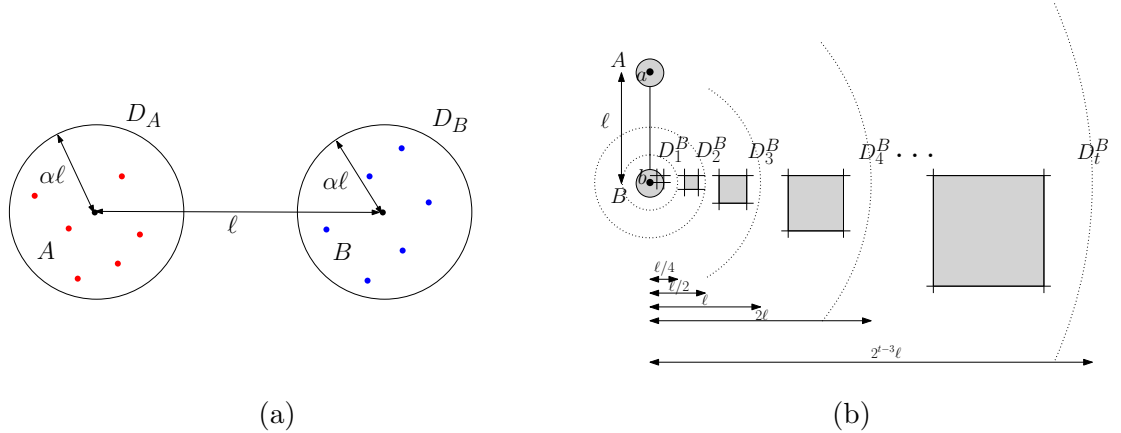


FIGURE 2.10. (a) A single pair (A, B) in an α -WSPD. (b) D_1^B, \dots, D_t^B centered at b , where $t = \lceil \log_2(1/\varepsilon) \rceil + 4$, constructed for a pair (A, B) in a $(1/8)$ -WSPD. D_j^B has radius $2^{j-3}\ell$ and is covered by canonical squares. A similar family of disks is constructed around a .

time. Next, they store $\tilde{\mathcal{B}}$ into a compressed quadtree and argue that the subdivision induced by the exposed nodes of the quadtree is an ε -VD of S .

Using the above procedure, we construct \mathcal{B}_{out} as follows: For $1 \leq i \leq n$, let \bar{p}_i denote the centroid of P_i . Set $\bar{P} = \{\bar{p}_i \mid 1 \leq i \leq n\}$. We run the above procedure on \bar{P} to construct an $(\varepsilon/3)$ -VD of \bar{P} . The set $\tilde{\mathcal{B}}$ of canonical squares constructed by the procedure is the desired \mathcal{B}_{out} . $|\mathcal{B}_{\text{out}}| = O((n/\varepsilon^2) \log(1/\varepsilon))$ and it is constructed in $O(n \log n + |\mathcal{B}_{\text{out}}|)$ time.

Computing the representative points. To facilitate the computation of representative points, we build an index, which is removed after the preprocessing step and is *not* part of our overall index. For a point $x \in \mathbb{R}^2$, let $\bar{P}_{\text{out}}[x] = \{\bar{p}_i \mid P_i \in \mathcal{P}_{\text{out}}[x]\}$. Similar to the index in Section 2.4.1, we construct an index for answering stabbing queries that can find, for a query point q , which squares in $\mathcal{C} = \bigcup_{i=1}^n C_i$ do not contain q and thus, find $\bar{P}_{\text{out}}[q]$. This index stores a family of canonical subsets of \bar{P} such that for any query point q , $\bar{P}_{\text{out}}[q]$ can be represented as the union of $O(\log^2 n)$ canonical subsets. For each canonical subset $A \subseteq \bar{P}$, we store an $(\varepsilon/12)$ -VD of A using the algorithm by Arya *et al.* [20]. The total space required for the index is

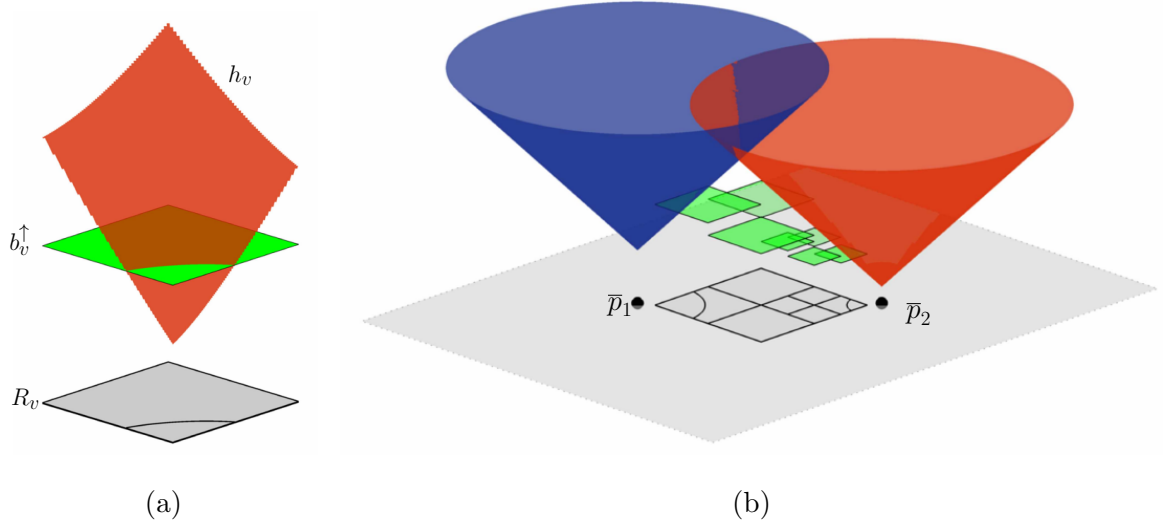


FIGURE 2.11. (a) The minimization diagram, Σ_v (shown below) of b_v^\uparrow and h_v for an exposed node v of \mathbb{T} . The complexity of Σ_v is $O(1)$. (b) A portion of the ε -EVD (shown below) obtained by replacing each cell of \mathbb{M} by Σ_v . The corresponding b_v^\uparrow 's (raised squares) and h_v 's (cones) are also shown.

$O((n/\varepsilon^2) \log^2(n) \log(1/\varepsilon))$ and it takes the same time to construct the index. For a query point q , we can now compute, in $O(\log^2(n) \log(n/\varepsilon))$ time, an $(\varepsilon/12)$ -NN of q in $\overline{P}_{\text{out}}[q]$.

We build a compressed quadtree \mathbb{T} on $\mathcal{B} = \mathcal{B}_{\text{in}} \cup \mathcal{B}_{\text{out}}$ as mentioned above. Let v be an exposed node of \mathbb{T} . If none of the ancestors of v (including v itself) stores a square of \mathcal{B}_{in} , P_v^{in} is undefined. Otherwise, among the squares \square of \mathcal{B}_{in} stored at the ancestors of v , let $\overline{\square}$ be the one with the smallest value of δ_\square . We set $P_v^{\text{in}} = P_{\overline{\square}}$, $b_v = \overline{\square}$, and b_v^\uparrow to the square in \mathbb{R}^3 obtained by lifting $\overline{\square}$ to the height $\delta_{\overline{\square}}$.

Next, we pick a point $z_v \in R_v$ and compute an $(\varepsilon/12)$ -NN of z_v in $\overline{P}_{\text{out}}[z_v]$, say \overline{p}_i . We set $P_v^{\text{out}} = P_i$ and $\overline{p}_v = \overline{p}_i$. Let $h_v(x) = \|x - \overline{p}_i\| + \text{Ed}(P_i, \overline{p}_i)$.

Finally, we compute the minimization diagram Σ_v of b_v^\uparrow and h_v within R_v ; see Figure 2.11(a). By replacing each cell R_v of \mathbb{E} with Σ_v , we obtain the desired ε -EVD of \mathcal{P} , whose size is $O(m) = O((n/\varepsilon^2) \log(1/\varepsilon))$; Figure 2.11(b) shows a

portion of such an ε -EVD. The total time spent in constructing this ε -EVD is $O((n/\varepsilon^2) \log^2(n) \log(n/\varepsilon) \log(1/\varepsilon))$.

Proof of correctness. The correctness of the algorithm follows from the following.

Lemma 2.5.5. *Let q be a point lying in the region R_v of an exposed node v of \mathbb{T} . Let P_q^{out} and P_q^{in} be the ENN of q in $\mathcal{P}_{\text{out}}[q]$ and $\mathcal{P}_{\text{in}}[q]$, respectively. Then,*

$$(i) \text{Ed}(P_v^{\text{in}}, q) \leq (1 + \varepsilon)\text{Ed}(P_q^{\text{in}}, q).$$

$$(ii) \text{Ed}(P_v^{\text{out}}, q) \leq (1 + \varepsilon)\text{Ed}(P_q^{\text{out}}, q).$$

Proof. (i) Let $\mathcal{B}_q = \{\square \in \mathcal{B}_i \mid P_i \in \mathcal{P}_{\text{in}}[q] \wedge q \in \square\}$. Since each square in \mathcal{B}_q contains q , it is stored at an ancestor of v in \mathbb{T} . Hence, $P_v^{\text{in}} = \arg \min_{P_i \in \mathcal{P}_{\text{in}}[q]} g_i(q)$. Furthermore, $P_q^{\text{in}} \in \mathcal{P}_{\text{in}}[q]$. Now, (i) follows from Lemma 2.5.4.

(ii) By construction, the set $\{C_i \mid x \notin C_i\}$ is the same for all $x \in R_v$. Therefore, $\mathcal{P}_{\text{out}}[q] = \mathcal{P}_{\text{out}}[v]$ and $P_v^{\text{out}} \in \mathcal{P}_{\text{out}}[q]$. Let \bar{p}_v and \bar{p}_q be the centroids of P_v^{out} and P_q^{out} respectively. The argument in Arya *et al.* [20] implies that

$$\|\bar{p}_v - q\| \leq (1 + \varepsilon/3)\|\bar{p}_q - q\|. \quad (2.3)$$

Since $P_v^{\text{out}} \in \mathcal{P}_{\text{out}}[q]$,

$$\|\bar{p}_v - q\| \geq 8\text{Ed}(P_v^{\text{out}}, \bar{p}_v)/(\varepsilon/3) \geq (24/\varepsilon)\text{Ed}(P_v^{\text{out}}, \bar{p}_v). \quad (2.4)$$

Therefore,

$$\begin{aligned} \text{Ed}(P_v^{\text{out}}, q) &\leq \|\bar{p}_v - q\| + \text{Ed}(P_v^{\text{out}}, \bar{p}_v) && \text{(triangle inequality)} \\ &\leq \|\bar{p}_v - q\| + (\varepsilon/24)\|\bar{p}_v - q\| && \text{(by (2.4))} \\ &\leq (1 + \varepsilon/24)(1 + \varepsilon/3)\|\bar{p}_q - q\| && \text{(by (2.3))} \\ &\leq (1 + \varepsilon/24)(1 + \varepsilon/3)g_{P_q^{\text{out}}}(q) && \text{(Lemma 2.5.4(iii))} \\ &\leq (1 + \varepsilon/24)(1 + \varepsilon/3)(1 + \varepsilon/3)\text{Ed}(P_q^{\text{out}}, q) && \text{(Lemma 2.5.4(iv))} \\ &\leq (1 + \varepsilon)\text{Ed}(P_q^{\text{out}}, q). \end{aligned}$$

□

Putting everything together, we conclude the following.

Theorem 2.5.6. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 , each with a pdf of description complexity s , and let $0 < \varepsilon < 1$ be a parameter. A Euclidean ε -EVD of \mathcal{P} of size $O((n/\varepsilon^2) \log(1/\varepsilon))$ can be constructed in $O((n/\varepsilon^2) \log^2(n) \log(n/\varepsilon) \log(1/\varepsilon))$ time. It can be processed in additional $O((n/\varepsilon^2) \log(1/\varepsilon))$ time into an index of size $O((n/\varepsilon^2) \log(1/\varepsilon))$ so that an ε -ENN query can be answered in $O(\log(n/\varepsilon))$ time.*

Noting that for an uncertain point P , the function g_P that approximates $\text{Ed}(P, x)$ under any L_p metric can be constructed in the same time, we also obtain the following.

Theorem 2.5.7. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 , each with a pdf of description complexity s , let $0 < \varepsilon < 1$ be a parameter. For any L_p metric, an ε -EVD of \mathcal{P} of size $O((n/\varepsilon^2) \log(1/\varepsilon))$ can be constructed in $O((n/\varepsilon^2) \log^2 n \log(n/\varepsilon) \log(1/\varepsilon))$ time. It can be processed in $O((n/\varepsilon^2) \log(1/\varepsilon))$ additional time into an index of size $O((n/\varepsilon^2) \log(1/\varepsilon))$ so that an ε -ENN query under the L_p metric can be answered in $O(\log(n/\varepsilon))$ time.*

Remarks. (i) Note that it is not necessary to construct the minimization diagram Σ_v for each exposed node $v \in \mathbb{T}$ if we simply want to answer ε -ENN queries. We can simply use $P_v^{\text{in}}, P_v^{\text{out}}, b_v^\uparrow$ and h_v stored at v to compute an ε -ENN of a query point.

(ii) The algorithm can be extended to higher dimensions. The size of the index becomes $O((n/\varepsilon^d) \log(1/\varepsilon))$, the preprocessing time become $O((n/\varepsilon^d) \text{polylog}(n/\varepsilon))$, and the query time remains the same.

2.5.3. Uncertain data, (k, ε) -ENN queries

We now generalize the above algorithm to (k, ε) -ENN queries. We again construct the sets \mathcal{B}_{in} and \mathcal{B}_{out} of canonical squares. For the set \mathcal{B}_{in} , if a square \square appears

in multiple \mathcal{B}_i 's, we keep up to k copies of \square in \mathcal{B}_{in} and δ_{\square} 's are the minimum of the values associated with the different copies of \square . For each square $\square \in \mathcal{B}_i$, we set $P_{\square} = P_i$. \mathcal{B}_{out} remains the same as for the ENN queries. As before, we build a compressed quadtree \mathbb{T} on $\mathcal{B} = \mathcal{B}_{\text{in}} \cup \mathcal{B}_{\text{out}}$. For each exposed node v of \mathbb{T} , we associate two sets of representative points $\mathcal{P}_v^{\text{kin}} \subset \mathcal{P}_{\text{in}}[v]$, $\mathcal{P}_v^{\text{kout}} \subset \mathcal{P}_{\text{out}}[v]$, defined as follows.

For an exposed node v of \mathbb{T} , if none of the ancestors of v (including v itself) stores a square of \mathcal{B}_{in} , $\mathcal{P}_v^{\text{kin}}$ is undefined. Otherwise, among the squares \square of \mathcal{B}_{in} stored at the ancestors of v , let $\bar{\square}$'s be the (at most k) ones with the smallest values of δ_{\square} . We set $\mathcal{P}_v^{\text{kin}}$ be the set of $P_{\bar{\square}}$'s. Next, we pick a representative point $z_v \in R_v$ and compute a $(k, \varepsilon/12)$ -NN of z_v in $\bar{P}_{\text{out}}[z_v]$, say \bar{p}_i 's. We set $\mathcal{P}_v^{\text{kout}}$ to be the set of P_i 's. For each $P_i \in \mathcal{P}_v^{\text{kin}}$, we store the canonical square \square_v of the function g_i that contains R_v , and for each $P_j \in \mathcal{P}_v^{\text{kout}}$, we also store the centroid \bar{p}_j of P_j at v . For all $x \in R_v$, $g_i(x)$ is constant and $g_j(x) = \|x - \bar{p}_j\| + \text{Ed}(P_j, \bar{p}_j)$.

Computing $\mathcal{P}_v^{\text{kin}}$ is straightforward but computing $\mathcal{P}_v^{\text{kout}}$ efficiently is more challenging. We build an index that stores a family of canonical subsets of \bar{P} such that for any query point q , $\bar{P}_{\text{out}}[q]$ can be represented as the union of $O(\log^2 n)$ canonical subsets. For each canonical subset X , we store a $(k, \varepsilon/12)$ -VD of X using Lemma 2.2.3. The total space required for the index is $O((n/\varepsilon^2) \log^2(n) \log(k) \log(1/\varepsilon))$ and it takes the same time to construct. For a point q , a $(k, \varepsilon/12)$ -NN of q in $\bar{P}_{\text{out}}[q]$ can be computed in $O((\log(n/\varepsilon) + k) \log^2 n)$ time. Again, this index is only needed for computing $\mathcal{P}_v^{\text{kout}}$ for all nodes, and is discarded after they have been computed.

Given a query point q , we find the exposed node v of \mathbb{T} containing q . Among the points in $\mathcal{P}_v^{\text{kin}} \cup \mathcal{P}_v^{\text{kout}}$, we return the k -ENN of $\mathcal{P}_v^{\text{kin}} \cup \mathcal{P}_v^{\text{kout}}$.

This completes the description of our index, and we now prove its correctness.

Proof of correctness. Let v be the exposed node of \mathbb{T} that contains a query point $q \in \mathbb{R}^2$. By construction, $\mathcal{P}_v^{\text{kin}}$ is obviously a (k, ε) -ENN of q in $\mathcal{P}_{\text{in}}[v] = \mathcal{P}_{\text{in}}[q]$.

It therefore suffices to prove that $\mathcal{P}_v^{\text{kout}}$ is a (k, ε) -ENN of q in $\mathcal{P}_{\text{out}}[v] = \mathcal{P}_{\text{out}}[q]$. Let $\bar{X}_v \subseteq \bar{P}_{\text{out}}[v]$ be the set of centroids of points in $\mathcal{P}_v^{\text{kout}}$. We need the following lemma.

Lemma 2.5.8. *\bar{X}_v is a $(k, \varepsilon/3)$ -NN of q in $\bar{P}_{\text{out}}[v]$.*

Before proving this lemma, we show how it implies $\mathcal{P}_v^{\text{kout}}$ to be a (k, ε) -ENN of q in $\mathcal{P}_{\text{out}}[v]$. For $i \leq k$, let $X_i \subseteq \bar{X}_v$ be the $(i, \varepsilon/12)$ -NN associated with the representative point z_v in $\bar{P}_{\text{out}}[v]$, and let $\mathcal{X}_i = \{P_j \mid \bar{p}_j \in X_i\}$ be the set of corresponding (uncertain) points of $\mathcal{P}_v^{\text{kout}}$; let $\mathcal{Y}_i = \{Y_1, \dots, Y_i\}$ denote the set of i ENNs of q in $\mathcal{P}_{\text{out}}[v]$ with $\text{Ed}(q, Y_1) \leq \dots \leq \text{Ed}(q, Y_i)$, and let $\bar{Y}_i = \{\bar{y}_j \mid Y_j \in \mathcal{Y}_i\}$ be the set of corresponding centroids of (uncertain) points in \mathcal{Y}_i . By induction on i , we prove that \mathcal{X}_i is an (i, ε) -ENN of q in $\mathcal{P}_{\text{out}}[v]$. For $i = 1$, the claim follows from the proof of correctness for the ENN queries. Assume that \mathcal{X}_i is an (i, ε) -ENN of q for all $i < k$, and now assume that $i = k$.

Let $X^* = \arg \max_{P \in \mathcal{X}_k} \text{Ed}(q, P)$. Since $|\mathcal{X}_k| = k$ and $\mathcal{X}_k \subseteq \mathcal{P}_{\text{out}}[v]$, $\text{Ed}(q, X^*) \geq \text{Ed}(q, Y_k)$. It thus suffices to show that $\text{Ed}(q, X^*) \leq (1 + \varepsilon)\text{Ed}(q, Y_k)$, because then it would imply that \mathcal{X}_k is a (k, ε) -ENN of q . Let $\bar{x}^* \in \bar{X}_v$ be the centroid of X^* . Let $\bar{y}^* = \arg \max_{\bar{y}_j \in \bar{Y}_k} \|q\bar{y}_j\|$, and let Y^* be the corresponding uncertain point of \bar{y}^* . Note that

$$\text{Ed}(q, Y^*) \leq \text{Ed}(q, Y_k). \quad (2.5)$$

Let y_k denote the k -th NN of q in $\bar{P}_{\text{out}}[v]$. Since $|\bar{Y}_k| = k$ and $\bar{Y}_k \subseteq \bar{P}_{\text{out}}[v]$, we have $\|qy_k\| \leq \|q\bar{y}^*\|$. Since \bar{X}_v is a $(k, \varepsilon/3)$ -NN of q in $\bar{P}_{\text{out}}[v]$,

$$\|q\bar{x}^*\| \leq (1 + \varepsilon/3)\|qy_k\| \leq (1 + \varepsilon/3)\|q\bar{y}^*\|. \quad (2.6)$$

Following the same argument as in the second part of the proof of Lemma 2.5.5, (2.5) and (2.6) imply that

$$\text{Ed}(q, X^*) \leq (1 + \varepsilon)\text{Ed}(q, Y^*) \leq (1 + \varepsilon)\text{Ed}(q, Y_k),$$

as desired. Hence $\mathcal{P}_v^{\text{kout}}$ is a (k, ε) -ENN of q in $\mathcal{P}_{\text{out}}[v]$. \square

We now prove Lemma 2.5.8, by generalizing the argument in [20] for ε -NN queries.

Proof of Lemma 2.5.8. Let x_1, \dots, x_k be the k points of \overline{X}_v in a non-decreasing order of their distances from the representative point z_v of R_v . We prove by induction on i that $X_i = \{x_1, \dots, x_i\}$ is an $(i, \varepsilon/3)$ -NN of q . For $i = 1$, this is true by the argument of Arya *et al.* [20]. Suppose X_i is an $(i, \varepsilon/3)$ -NN of q in $\overline{P}_{\text{out}}[v]$ for all $i < k$, and now assume $i = k$. Let y_1, \dots, y_k be the k -NN of q in $\overline{P}_{\text{out}}[v]$ with $\|qy_1\| \leq \dots \leq \|qy_k\|$. We show that there is a point $x^* \in X_k$ such that

$$\|qy_k\| \leq \|qx^*\| \leq (1 + \varepsilon/3)\|qy_k\|. \quad (2.7)$$

We choose $x^* = \arg \max_{x \in X_k} \|qx\|$. If $x^* = y_k$, then (2.7) obviously holds, so assume $x^* \neq y_k$. Since $|X_k| = k$ and $X_k \subseteq \overline{P}_{\text{out}}[v]$, $\|qy_k\| \leq \|qx^*\|$. So it suffices to prove that $\|qx^*\| \leq (1 + \varepsilon/3)\|qy_k\|$.

If $x^* \neq x_k$, then by induction hypothesis, $\|qx^*\| \leq (1 + \varepsilon/3)\|qy_{k-1}\| \leq (1 + \varepsilon/3)\|qy_k\|$, as desired. Next assume $x^* = x_k$. Since $\overline{P}_{\text{out}}[v] \subseteq \overline{P}$ and $x^* = x_k \neq y_k$, there is a pair (X, Y) in the WSPD of \overline{P} such that $x_k \in X$ and $y_k \in Y$. Let x', y' be the centers of the smallest disks D_X, D_Y containing X and Y , respectively, and let $\ell = \|x'y'\|$. Then by the WSPD property, the radii of D_X, D_Y is at most $\ell/8$. As in [20], we consider three cases.

Case 1: $\|qy'\| \geq 6\ell/\varepsilon$.

By the triangle inequality,

$$\|qx_k\| \leq \|qy'\| + \|y'x'\| + \|x'x_k\| \leq \|qy'\| + 9\ell/8 \leq (1 + 3\varepsilon/16)\|qy'\|. \quad (2.8)$$

On the other hand,

$$\|qy_k\| \geq \|qy'\| - \|y'y_k\| \geq \|qy'\| - \ell/8 \geq (1 - \varepsilon/48)\|qy'\|. \quad (2.9)$$

Combining (2.8) and (2.9) and using $\varepsilon \in (0, 1)$, we obtain

$$\|qx_k\| \leq \frac{1 + 3\varepsilon/16}{1 - \varepsilon/48} \|qy_k\| \leq (1 + \varepsilon/3) \|qy_k\|.$$

Case 2: $\ell/4 \leq \|qy'\| < 6\ell/\varepsilon$.

Since $\|qy'\| < 6\ell/\varepsilon$, the preprocessing algorithm constructed a disk of radius at most $2\|qy'\|$ centered at y' that contains q , and that it generated a quadtree box $B \in \mathcal{B}_{\text{out}}$ of side length $s_B \leq 2\|qy'\| \cdot (\varepsilon/3)/32 = (\varepsilon/48)\|qy'\|$ that contains q . Consequently $R_v \subseteq B$ and thus $\|z_v q\| \leq 2 \cdot s_B \leq (\varepsilon/24)\|qy'\|$. But

$$\|qy_k\| \geq \|qy'\| - \|y'y_k\| \geq \|qy'\| - \ell/8 \geq (1/2)\|qy'\|.$$

Therefore,

$$\|z_v q\| \leq (\varepsilon/12) \|qy_k\|. \quad (2.10)$$

Let $y^* = \arg \max_{1 \leq i \leq k} \|z_v y_i\|$, and let $\text{NN}_k(z_v)$ denote the k -th NN of z_v in $\overline{P}_{\text{out}}[z_v]$.

Note that $\|qy^*\| \leq \|qy_k\|$ and $\|z_v \text{NN}_k(z_v)\| \leq \|z_v y^*\|$. Then $\|z_v x_k\| \leq (1 + \varepsilon/12) \|z_v \text{NN}_k(z_v)\| \leq (1 + \varepsilon/12) \|z_v y^*\|$. Using these inequalities and (2.10), we obtain

$$\begin{aligned} \|qx_k\| &\leq \|qz_v\| + \|z_v x_k\| \\ &\leq \|qz_v\| + (1 + \varepsilon/12) \|z_v y^*\| \\ &\leq \|qz_v\| + (1 + \varepsilon/12) (\|z_v q\| + \|qy^*\|) \\ &\leq (1 + \varepsilon/12) \|qy_k\| + (2 + \varepsilon/12) (\varepsilon/12) \|qy_k\| \\ &\leq (1 + \varepsilon/4 + \varepsilon^2/144) \|qy_k\| \\ &\leq (1 + \varepsilon/3) \|qy_k\|. \end{aligned}$$

The last inequality follows from the fact that $\varepsilon \in (0, 1)$.

Case 3: $\|qy'\| < \ell/4$.

By construction, there is a quadtree box $B \in \mathcal{B}_{\text{out}}$ of side length $s_B \leq (\ell/4) \cdot (\varepsilon/3)/32$ that contains q and $R_v \subseteq B$. Hence $\|z_v q\| \leq 2 \cdot s_B \leq \ell\varepsilon/192$.

As in Case 2, let $y^* = \arg \max_{1 \leq i \leq k} \|z_v y_i\|$, and let $\text{NN}_k(z_v)$ denote the k -th NN of z_v in $\overline{P}_{\text{out}}[z_v]$. Note that $\|qy^*\| \leq \|qy_k\|$ and $\|z_v \text{NN}_k(z_v)\| \leq \|z_v y^*\|$. We have

$$\begin{aligned} \|z_v \text{NN}_k(z_v)\| &\leq \|z_v y^*\| \leq \|qy^*\| + \|qz_v\| \leq \|qy_k\| + \|qz_v\| \\ &\leq \|qy'\| + \|y'y_k\| + \|qz_v\| \\ &\leq \ell/4 + \ell/8 + \ell\varepsilon/192 = (3/8 + \varepsilon/192)\ell. \end{aligned}$$

On the other hand,

$$\begin{aligned} \|z_v x_k\| &\geq \|x'y'\| - \|y'q\| - \|x'x_k\| - \|z_v q\| \\ &\geq \ell - \ell/4 - \ell/8 - \ell\varepsilon/192 = (5/8 - \varepsilon/192)\ell. \end{aligned}$$

Consequently,

$$\frac{\|z_v x_k\|}{\|z_v \text{NN}_k(z_v)\|} \geq \frac{5/8 - \varepsilon/192}{3/8 + \varepsilon/192} \geq \frac{5 - \varepsilon/24}{3 + \varepsilon/24} \geq \frac{119}{73},$$

which contradicts that x_k is an $(\varepsilon/12)$ -appropriate k -th NN of z_v in $\overline{P}_{\text{out}}[v]$, for $\varepsilon \in (0, 1)$. Hence, this case is impossible.

Putting everything together, we conclude that \overline{X}_v is a $(k, \varepsilon/3)$ -NN of q in $\overline{P}_{\text{out}}[v]$. \square

We obtain the following.

Theorem 2.5.9. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 , each with a pdf of description complexity s , let $0 < \varepsilon < 1$ be a parameter and let $d(\cdot, \cdot)$ be the Euclidean distance. An index of size $O((nk/\varepsilon^2) \log(k) \log(1/\varepsilon))$ can be constructed in $O((nk/\varepsilon^2) \log(k) \log^2(n) \log(n/\varepsilon) \log(1/\varepsilon))$ time, such that a (k, ε) -ENN query can be answered in $O(\log(n/\varepsilon) + k)$ time.*

Remarks. (i) The algorithm can be extended to higher dimensions. The size of the index becomes $O((nk/\varepsilon^d) \log(k) \log(1/\varepsilon))$, the preprocessing time become $O((nk/\varepsilon^d) \log(k) \log^d(n) \log(n/\varepsilon) \log(1/\varepsilon))$, and the query time remains the same.

(ii) Instead of storing $\mathcal{P}_v^{\text{kin}}$, $\mathcal{P}_v^{\text{kout}}$ at each node of the compressed quadtree, we can build a separate index that can compute them as part of the query procedure. The space (resp. preprocessing time) used is shaved by a factor k (resp. $\log^2 n$), but the query time increases by a factor of $\log^2 n$. We omit the details of this index.

(iii) We conjecture that one may be able to shave the factor k from the space used in Theorem 2.5.9.

2.5.4. Uncertain query

Let $\mathcal{P} = \{p_1, \dots, p_n\}$ be a set of (certain) points in \mathbb{R}^2 . For an uncertain query point Q of description complexity s and a parameter $0 < \varepsilon < 1$, we wish to compute its ε -ENN in \mathcal{P} . We preprocess \mathcal{P} into a compressed quadtree \mathbb{T} as described in Section 2.2. We also preprocess \mathcal{P} for answering NN queries, by constructing its Voronoi diagram and preprocessing it for point-location queries. The size of the index is $O(n)$ and it can be built in $O(n \log n)$ time [52].

To answer a given query Q , we construct the function $g_Q : \mathbb{R}^2 \rightarrow \mathbb{R}_{\geq 0}$ using Lemma 2.5.4. Let \mathcal{B} be the set of canonical squares defining g_Q . For each $\square \in \mathcal{B}$, we query \mathbb{T} and report a point $p_\square \in \square \cap \mathcal{P}$ if there exists one. Among all the points reported, we return the point $p^* = \arg \min_{p_\square} g_Q(p_\square)$. If no point is reported, then we return the point of \mathcal{P} that is closest to \bar{q} , the centroid of Q . The correctness of the algorithm follows from the Lemma 2.5.4. Querying each $\square \in \mathcal{B}$ takes $O(\log n)$ time, by Lemma 2.2.2, and the NN of \bar{q} can be computed in $O(\log n)$ time, so we conclude the following:

Theorem 2.5.10. *Let \mathcal{P} be a set of n (certain) points in \mathbb{R}^2 . An index of size $O(n)$ can be built on \mathcal{P} in $O(n \log n)$ time so that for an uncertain query point Q with a pdf of description complexity s and for a parameter $0 < \varepsilon < 1$, an ε -ENN of Q can be computed in $O((1/\varepsilon^2) \log(1/\varepsilon)(\log n + s))$ time.*

Remarks. (i) The algorithm can be extended to \mathbb{R}^d by building a d -dimensional compressed quadtree. The size and the preprocessing time remain the same, but the query time increases to $O((1/\varepsilon^d) \log(1/\varepsilon)(\log n + s))$.

(ii) All pieces of the function g_Q need not be computed in the beginning itself. They can be constructed hierarchically while querying the compressed quadtree on \mathcal{P} . This does not affect the worst-case running time but it is more efficient in practice.

Answering k -ENN queries. We preprocess \mathcal{P} into a compressed quadtree \mathbb{T} as described in Section 2.2 (cf. Lemma 2.2.2). Next, we preprocess \mathcal{P} into an index $\Gamma(\mathcal{P})$ of size $O(n)$ in $O(n \log n)$ expected time for answering k -NN queries in $O(\log n + k)$ time, as in [2].

Given a query Q , we construct the function $g_Q : \mathbb{R}^2 \rightarrow \mathbb{R}_{\geq 0}$ using Lemma 2.5.4. Let \mathcal{B} be the set of squares defining g_Q . We sort the squares \square of \mathcal{B} in increasing order of their associated values δ_{\square} . For each $\square \in \mathcal{B}$ in the sorted order, we query \mathbb{T} and report the points of $\square \cap \mathcal{P}$ one by one. We stop the query process once we have reported k points. If we fail to report k points over all squares of \mathcal{B} , we query the index $\Gamma(\mathcal{P})$ to find the k -NN of \bar{q} in \mathcal{P} . Among the at most $2k$ points found, we pick the best k points. Omitting the details, we conclude the following.

Theorem 2.5.11. *Let \mathcal{P} be a set of n (certain) points in \mathbb{R}^2 . An index of size $O(n)$ can be built on \mathcal{P} in $O(n \log n)$ expected time so that for an uncertain query point Q with a pdf of description complexity s and for a parameter $0 < \varepsilon < 1$, a (k, ε) -ENN of Q can be computed in $O((1/\varepsilon^2) \log(1/\varepsilon)(\log n + s) + k)$ time.*

Remarks. The algorithm can be extended to \mathbb{R}^d , $d \geq 3$. The modifications lie in that: (1) We build a d -dimensional compressed quadtree; (2) We preprocess \mathcal{P} for answering $(k, \varepsilon/3)$ -NN queries by building a $(k, \varepsilon/3)$ -VD of \mathcal{P} in Lemma 2.2.3; (3) The function $g_Q : \mathbb{R}^2 \rightarrow \mathbb{R}_{\geq 0}$ is constructed by applying Lemma 2.5.4 using $\varepsilon/3$ (instead of ε). The size of the index becomes $O((n/\varepsilon^d) \log(k) \log(1/\varepsilon))$, and the preprocessing

time becomes $O(nk \log n + (n/\varepsilon^d)(1/\varepsilon^{d-1} + \log n) \log(k) \log(1/\varepsilon))$. A query takes $O((1/\varepsilon^d) \log(1/\varepsilon)(\log n + s) + k)$ time.

2.6. Conclusion

In this chapter we considered the problem of answering NN queries under uncertainty. We used a probabilistic framework to model the uncertainty in the location of input data or query point, and presented indexing schemes of linear or near-linear size that answer exact or ε -approximate ENN and k -ENN queries in \mathbb{R}^2 in $\text{polylog}(n)$ time under squared Euclidean, L_1 , L_2 , and L_∞ distance functions. As far as we know, these are the first methods to obtain such bounds. We conclude by mentioning a few open problems:

- (i) What is the combinatorial complexity of $\text{EVD}(\mathcal{P})$ when $d(\cdot, \cdot)$ is the Euclidean distance? Can a quadratic upper bound be proved? Although the algebraic complexity of a bisector is large, the combinatorial complexity, i.e., the number of vertices, can be small.
- (ii) The expected distance is not a reliable indicator when the variance of the pdf's is not small. In this case, one is interested in computing a point that is the nearest neighbor with highest probability or the points that are the nearest neighbors with probability higher than a given threshold. Is there a linear-size index to answer these queries in sublinear time in the worst case? This problem seems hard even for very simple pdf's such as Gaussians. A few initial results were obtained in Chapter 4, but the problem largely remains unsolved.

Aggregate Nearest Neighbor

3.1. Introduction

In this chapter, we present improved results over Chapter 2 for expected nearest neighbor in the L_1 metric with uncertain queries. We choose to present this chapter in the setting as for top- k weighted SUM aggregate nearest neighbor in the L_1 metric to emphasize the improvement and importance for the widely-studied aggregate nearest neighbor problem.

Top- k nearest neighbor searching has been well-studied, e.g., see [23,48] for surveys. For a set P of points in the d -D space \mathbb{R}^d , the problem asks for a data structure to quickly report the k nearest neighbors in P for any query point. Aggregate nearest neighbor (ANN) searching [9, 86, 87, 88, 90, 94, 100, 102, 112, 126], also known as *group* nearest neighbor searching, is a generalization of the basic problem, where each query consists of a set of (weighted) points and the result of the query is based on applying *aggregate* operators, such as (weighted) SUM and MAX, on all the points in the query. In this chapter, we study top- k ANN queries using the weighted SUM operator under the L_1 metric in the plane.

Problem statement and our results. For any two points p and q in the plane, denote by $d(p, q)$ the distance of p and q . Let Q be a set of points and each point $q \in Q$ has a weight $w(q) > 0$. Throughout the chapter, we use m to denote the size of Q for any given Q (note that m is not a fixed value). For any point p in the plane, the *aggregate distance* from p to Q , denoted by $\text{Ad}(p, Q)$, is defined to be $\text{Ad}(p, Q) = \sum_{q \in Q} w(q)d(p, q)$.

Let P be a set of n points in the plane. Given a query consisting of a set Q of weighted points and an integer k , $1 \leq k \leq n$, the *top- k aggregate nearest neighbors* (top- k ANNs) of Q in P are the k points of P whose aggregate distances to Q are the smallest among all points in P ; we denote by $S_k(P, Q)$ the set of the top- k ANNs. Our goal is to design a data structure to quickly report the set $S_k(P, Q)$ for any weighted point set Q and integer k .

For any point p , denote by $x(p)$ the x -coordinate of p and by $y(p)$ the y -coordinate of p .

In this chapter, we consider the L_1 metric. Specifically, for two points p and q in the plane, their L_1 distance is $d(p, q) = |x(p) - x(q)| + |y(p) - y(q)|$. We build an $O(n \log n \log \log n)$ -size data structure in $O(n \log n \log \log n)$ time that can answer each query in $O(m \log m + (k + m) \log^2 n)$ time. With trade-off between preprocessing and query time, we also build two other data structures: the first one has $O(n \log n)$ preprocessing time and space with $O(m \log m + (k + m) \log^2 n \log \log n)$ query time; the second one has $O(n \log n \log^* n)$ preprocessing time and space with $O(m \log m + (k + m) \log^2 n \log^* n)$ query time.

For the 1-D version, we build an $O(n)$ -size data structure in $O(n \log n)$ time with $O(\min\{k, \log m\} \cdot m + k + \log n)$ query time, and the query time can be reduced to $O(k + m + \log n)$ time if the points of Q are given in sorted order.

Further, we extend our techniques to solve the top- k *aggregate farthest neighbor* (AFN) searching problem, with the same bounds as above.

Related work. Previously, only approximation and heuristic results were given for the top- k ANN query problem [91]. For the special case where $k = 1$, with $O(n \log^2 n)$ time and space preprocessing, Agarwal *et al.* [9] can answer each top-1 ANN query in $O(m^2 \log^3 n)$ time. Hence, even for $k = 1$, our results are better than that in [9] in all three aspects: preprocessing time, space, and query time. Recently, Ahn *et al.* [18] studied the *unweighted version* of the problem, and they gave two data structures under the assumption that the maximum value of $|Q|$ is known in advance as m for all queries, with the following time bounds: the first one is built in $O(m^2 n \log^2 n)$ time and space with $O(m^2 \log n + k(\log \log n + \log m))$ query time; the second one is built in $O(m^2 n \log n)$ time and $O(m^2 n)$ space with $O(m^2 \log n + (k + m) \log^2 n)$ query time. Clearly, our results, albeit on the weighted version and do not require the assumption, are generally better than the results in [18] for most cases. (e.g., if $m = O(1)$, their second result is better than ours on the unweighted version with the assumption).

For the L_2 metric, only heuristic and approximation algorithms are known for even top-1 ANN queries [9, 87, 88, 94, 100, 102, 112, 126]. The best heuristic method for the top-1 ANN queries is based on R-tree [102], where each query takes $O(n)$ time in the worst case. Li *et al.* [88] gave a data structure with 3-approximation queries for the top-1 ANN. Agarwal *et al.* [9] gave a data structure with a polynomial-time approximation scheme for the top-1 ANN.

If MAX is used to define the aggregate distance, i.e., $\text{Ad}(p, Q) = \max_{q \in Q} w(q) d(p, q)$, to the best of our knowledge, we are not aware of any previous work on the general weighted top- k , even for $k = 1$. For the unweighted version, heuristic and approximation algorithms are given for the L_2 top-1 query [86, 102], and Wang [123] gave an $O(m\sqrt{n} \log^{O(1)} n)$ time exact algorithm. For the L_1 metric, Wang [123] built a data structure of $O(n)$ size in $O(n \log n)$ time that can answer each (unweighted) L_1 top- k ANN-MAX query in $O(m + k \log n)$ time.

In addition, Li et al. [88] proposed the *flexible* top- k ANN queries, which extend the ANN queries, and provided approximation algorithms for both unweighted SUM and MAX operators in any metric space and any fixed dimension.

Note that the weighted ANN queries studied in this chapter can be used to solve the *expected nearest neighbor* (ENN) queries for uncertain query points. In each ENN query, an uncertain point Q is given with m different locations and each location q is associated with a probability $w(q)$ of being the true location of Q , and the query asks for the point in P that has the smallest expected distance to Q . Agarwal *et al.* [9] gave the first nontrivial methods for answering exact or approximate ENN queries under L_1 , L_2 , and the squared Euclidean distance, with provable performance guarantees. We have mentioned their exact top-1 query algorithm on L_1 metric earlier. Other formulations on nearest neighbor queries over uncertain data have also been studied in [9] and elsewhere, e.g., [6, 28, 44, 47, 91, 127].

For the top- k AFN queries, we are not aware of any previous work on the weighted queries. For unweighted queries, Gao *et al.* [60] gave heuristic algorithms using R-trees for the L_2 metric; for the L_1 metric, Ahn *et al.* [18] also extended their techniques to top- k AFN queries with the same bounds, assuming that the maximum value of m is known for all queries. For $k = 1$, farthest Voronoi diagrams [23] can be used for answering top-1 AFN queries.

Below, in Section 3.2, we present our results in the 1-D space, which are generalized to the 2-D space in Section 3.3. One may view Section 3.2 as a “warm-up” for Section 3.3. Section 3.4 extends our techniques to solve the AFN queries. For simplicity of discussion, we make a general position assumption that no two points in $P \cup Q$ have the same x - or y -coordinate for any Q ; we also assume no two points of P have the same aggregate distance to Q . Our techniques can be extended to the general case without these assumptions, although the discussion would be more tedious.

To simplify the notation, we will write $\text{Ad}(p)$ for $\text{Ad}(p, Q)$, and $S_k(P)$ for $S_k(P, Q)$.

When we say “the ANN”, we mean the top-1 ANN. For any $P' \subseteq P$, denote by $S_k(P')$ the set of the top- k ANNs of Q in P' . Let $W = \sum_{q \in Q} w(q)$. We assume $m < n$ always holds since otherwise we could compute $S_k(P)$ in $O((m+n) \log m) = O(m \log m)$ time by directly computing the aggregate distances for all points in P , and we omit the details.

3.2. Top- k ANN Searching in the 1-D Space

In the 1-D space, all points of P lie on a real line L . We assume L is the x -axis. Consider any $Q = \{q_1, \dots, q_m\}$ on L . For any point p on L , the aggregate distance from p to Q is $\text{Ad}(p) = \sum_{q \in Q} w(q) d(p, q)$, where $d(p, q) = |x(p) - x(q)|$. Given any Q and k , our goal is to compute $S_k(P)$, i.e., the set of the top- k ANNs of Q in P .

A point p on L is called a *global minimum point* if it minimizes the aggregate distance $\text{Ad}(p)$ among all points on L . Such a global minimum point may not be unique. The global minimum point is also known as weighted Fermat-Weber point [55], and as shown below, it is very easy to compute in our problem setting.

To find $S_k(P)$, we do the following. First, we find a global minimum point q^* on L . Second, q^* partitions P into two subsets P_l and P_r , for which we compute $S_k(P_l)$ and $S_k(P_r)$. Finally, $S_k(P)$ is obtained by taking the first k points after merging $S_k(P_l)$ and $S_k(P_r)$.

Note that the points in Q may not be given sorted on L . Let q^* be the point in Q such that $\sum_{x(q) < x(q^*), q \in Q} w(q) < W/2$ and $w(q^*) + \sum_{x(q) < x(q^*), q \in Q} w(q) \geq W/2$. If we view $w(q)$ as the weight of $x(q)$, then q^* is the *weighted median* of the set $\{x(q) \mid q \in Q\}$ [49]. We claim that q^* is a global minimum point on L . To prove the claim, we first present Lemma 3.2.1. We say a function $f(x)$ is *monotonically increasing* (resp., *decreasing*) if $f(x_1) \leq f(x_2)$ for any $x_1 \leq x_2$ (resp., $x_1 \geq x_2$).

Lemma 3.2.1. *For any point p on L and $p \neq q^*$, if we move p on L towards q^* , the*

aggregate distance $\text{Ad}(p)$ is monotonically decreasing.

Proof. Without loss of generality, assume p is on the left side of q^* and we move p on L to the right towards q^* . The case where p is on the right side of q^* can be analyzed similarly. At any moment during the movement of p , let $Q_L = \{q \in Q \mid x(q) \leq x(p)\}$ and $Q_R = Q \setminus Q_L$. According to the definition of $\text{Ad}(p)$, we have

$$\begin{aligned} \text{Ad}(p) &= \sum_{q \in Q} |x(p) - x(q)| = \sum_{q \in Q_L} w(q) \cdot [x(p) - x(q)] + \sum_{q \in Q_R} w(q) \cdot [x(q) - x(p)] \\ &= \left[\sum_{q \in Q_L} w(q) - \sum_{q \in Q_R} w(q) \right] \cdot x(p) - \sum_{q \in Q_L} w(q) \cdot x(q) + \sum_{q \in Q_R} w(q) \cdot x(q). \end{aligned} \tag{3.1}$$

Because p is to the left of q^* , according to the definition of q^* , $\sum_{q \in Q_L} w(q) \leq W/2 \leq \sum_{q \in Q_R} w(q)$ holds. Further, as p moves to the right towards q^* , the value $x(p)$ is monotonically increasing.

Suppose p is between two points q_i and q_j of Q such that $x(q_i) \leq x(p) < x(q_j)$ and there are no other points of Q between q_i and q_j . Note that it is possible that such a point q_i does not exist (i.e., no point of Q is on the left side of p), in which case we let $x(q_i) = -\infty$.

If p moves in the interval $[x(q_i), x(q_j))$ to the right, then both sets Q_L and Q_R stay the same, and thus, the value $[\sum_{q \in Q_L} w(q) - \sum_{q \in Q_R} w(q)] \cdot x(p)$ is monotonically decreasing and neither $\sum_{q \in Q_L} w(q) \cdot x(q)$ nor $\sum_{q \in Q_R} w(q) \cdot x(q)$ changes. Therefore, if p moves in the interval $[x(q_i), x(q_j))$ to the right, $\text{Ad}(p)$ is monotonically decreasing. We claim that for any p in $[x(q_i), x(q_j))$, it always holds that $\text{Ad}(p) \geq \text{Ad}(q_j)$, which leads to the lemma. Indeed, it can be verified that $\text{Ad}(q_j) - \text{Ad}(p) = (x(q_j) - x(p)) \cdot [\sum_{q \in Q_L} w(q) - \sum_{q \in Q_R} w(q)]$. Since $x(q_j) - x(p) > 0$ and $[\sum_{q \in Q_L} w(q) - \sum_{q \in Q_R} w(q)] \leq 0$, we obtain that $\text{Ad}(q_j) - \text{Ad}(p) \leq 0$.

The lemma thus follows. \square

Lemma 3.2.1 implies that $\text{Ad}(p)$ attains a global minimum at $p = q^*$. Hence, the point q^* is a global minimum point on L . Let $P_l = \{p \in P \mid x(p) \leq x(q^*)\}$ and $P_r = P \setminus P_l$. We find the set $S_k(P_r)$ of top- k ANNs of Q in P_r by scanning the sorted list of P_r from left to right and reporting the first k scanned points. $S_k(P_l)$ can be obtained similarly. Among the $2k$ points obtained above, we report the set of k points with the smallest aggregate distances to Q as $S_k(P)$. Therefore, we obtain the following.

Theorem 3.2.2. *Given a set P of n points on the real line L , with $O(n \log n)$ time and $O(n)$ space preprocessing, the top- k ANNs can be found in $O(\min\{k, \log m\} \cdot m + k + \log n)$ time for any weighted point set Q and integer k ; if the points of Q are given sorted on L , then the query time is $O(k + m + \log n)$.*

Proof. The only preprocessing is to sort the points in P from left to right, which takes $O(n \log n)$ time and $O(n)$ space.

Given any query Q and any k , we first compute the point q^* , in $O(m)$ time by the weighted selection algorithm [49]. The sorted lists of P_l and P_r can be obtained implicitly in $O(\log n)$ time by determining the two neighboring points of q^* in the sorted list P . If $k = 1$, it is sufficient to consider the two neighboring points of q^* in P : we compute, in $O(m)$ time, their aggregate distances to Q , and return the point with smaller aggregate distance. Hence, the total time for finding ANN is $O(m + \log n)$. Below, we compute $S_k(P)$ for general k .

For simplicity of discussion, we assume $|P_l| \geq k$ and $|P_r| \geq k$. We first compute the set $S_k(P_r)$ of top- k ANNs of Q in P_r by scanning the sorted list of P_r from left to right and report the first k points. $S_k(P_l)$ can be obtained similarly. Among the found $2k$ points in $S_k(P_l) \cup S_k(P_r)$, we report the set of k points with the smallest aggregate distances to Q as $S_k(P)$. Let $\Psi_l = \{\text{Ad}(p) \mid p \in S_k(P_l)\}$ denote the set of k aggregate distance values $\text{Ad}(p)$ of all points p in $S_k(P_l)$. Similarly, we define Ψ_r . Set

$\Psi = \Psi_l \cup \Psi_r$. If we know Ψ , the final step can be done easily in $O(k)$ time. Ψ can be computed in $O(mk)$ time in a straightforward way, leading to $O(mk + \log n)$ overall query time. In the following, we show that Ψ can be computed in $O(m \log m + k)$ time, which leads to $O(m \log m + k + \log n)$ overall query time, and further, if Q is given sorted, Ψ can be computed in $O(m + k)$ time.

The m points in Q partition L into $m + 1$ intervals and an easy observation is that the aggregate distance $\text{Ad}(p)$ changes linearly as p changes in each interval. Specifically, consider computing $\text{Ad}(p)$ for any given point p : if we know the four values $\sum_{q \in Q_L} w(q)$, $\sum_{q \in Q_R} w(q)$, $\sum_{q \in Q_L} w(q)x(q)$, and $\sum_{q \in Q_R} w(q)x(q)$ in Eq. (3.1) in the proof of Lemma 3.2.1, then $\text{Ad}(p)$ can be computed in constant time. In order to utilize this, we preprocess Q as follows.

We sort the points of Q from left to right and assume the sorted list is q_1, q_2, \dots, q_m . For each $1 \leq j \leq m$, we compute the four values $\sum_{i=1}^j w(q_i)$, $\sum_{i=1}^j w(q_i)x(q_i)$, $\sum_{i=j}^m w(q_i)$, and $\sum_{i=j}^m w(q_i)x(q_i)$. Note that all these $4m$ values can be computed in $O(m)$ time (after Q is sorted). Then, given any point p , if we know the index i such that $x(q_i) \leq x(p) < x(q_{i+1})$, then $\text{Ad}(p)$ can be computed in constant time.

Now, we compute Ψ . Let $Q_l = \{q \in Q \mid x(q) \leq x(q^*)\}$ and $Q_r = Q \setminus Q_l$. After having q^* , Q_l and Q_r can be obtained implicitly in $O(\log m)$ time by binary search on the sorted list of Q . Recall that we scan the points in P_r from left to right to find $S_k(P_r)$. If we scan both P_r and Q_r simultaneously, then at the moment of scanning any point, say p , we already know the index i such that $x(q_i) \leq x(p) < x(q_{i+1})$ and thus $\text{Ad}(p)$ can be computed in constant time. Hence, Ψ_r can be computed in $O(m + k)$ time, so can Ψ_l . In this way, the total time for computing Ψ is $O(k + m \log m)$. Note that if Q is given sorted on L , the query time becomes $O(k + m)$.

The theorem thus follows. □

3.3. Top- k ANN Searching in the Plane

In this section, both P and Q are in the plane. We generalize the techniques in Section 3.2. For any Q , we first find a global minimum q^* in the plane. Then, for each quadrant R of the four quadrants with respect to q^* (i.e., the four quadrants partitioned by the vertical line and the horizontal line through q^*), we find the top- k ANNs of Q in $P \cap R$ (i.e., $S_k(P \cap R)$) and compute the aggregate distance values $\text{Ad}(p)$ for all $p \in S_k(P \cap R)$; among the found $4k$ points, we report the set of k points with smallest aggregate distances to Q as $S_k(P)$. Note that we view each quadrant as a closed region including its two bounding half-lines (with the common endpoint q^*).

In the sequel, we only describe our algorithm for the first quadrant, and the other three quadrants can be treated in a similar manner. Let $P^1 \subseteq P$ be the set of points lying in the first quadrant, i.e., $P^1 = \{p \in P \mid x(p) \geq x(q^*), y(p) \geq y(q^*)\}$. Our goal is to find $S_k(P^1)$, the set of top- k ANNs of Q in P^1 . Let z_i denote the i -th ANN of Q in P^1 . Our algorithm computes $S_k(P^1)$ in the order of z_1, z_2, \dots, z_k .

The problem is more difficult than the 1-D case. For example, in 1-D, for all $i \leq k$, the i -th ANN in P_l or P_r can be easily found by scanning a sorted list. Here in 2-D, instead, by proving a monotonicity property, we show that the i -th ANN z_i in P^1 must be on a “skyline” (to be defined later) and we need to somehow search the “skyline”. After z_i is found, we determine a new skyline without considering z_1, \dots, z_i , and then find z_{i+1} by searching the new skyline. This procedure continues until z_k is obtained. The details are given below.

Consider any $Q = \{q_1, q_2, \dots, q_m\}$ and any k . Our goal is to find the top- k ANN set $S_k(P^1)$ in the first quadrant. A point p in the plane is called a *global minimum point* if it minimizes the aggregate distance $\text{Ad}(p)$ among all points in the plane. Below, we first find a global minimum point and prove a monotonicity property. Recall that $W = \sum_{q \in Q} w(q)$.

Let $q_x^* \in Q$ such that $\sum_{x(q) < x(q_x^*), q \in Q} w(q) < W/2$ and $w(q_x^*) + \sum_{x(q) < x(q_x^*), q \in Q} w(q) \geq W/2$. If we view $w(q)$ as the weight of $x(q)$ for each $q \in Q$, then $x(q_x^*)$ is the weighted median of the set $\{x(q) \mid q \in Q\}$ [49]. Similarly, let $q_y^* \in Q$ such that $\sum_{y(q) < y(q_y^*), q \in Q} w(q) < W/2$ and $w(q_y^*) + \sum_{y(q) < y(q_y^*), q \in Q} w(q) \geq W/2$.

We claim that $q^* = (x(q_x^*), y(q_y^*))$ is a global minimum point. To prove the claim, we first present Lemma 3.3.1, which generalizes Lemma 3.2.1. A path in the plane is *monotone* if we move from one endpoint of it to the other, the x -coordinate (resp. y -coordinate) is monotonically changing (either increasing or decreasing).

Lemma 3.3.1. *For any point p in the plane with $p \neq q^*$, if we move p towards q^* along a monotone path, the aggregate distance $\text{Ad}(p)$ is monotonically decreasing.*

Proof. According to the definition of $\text{Ad}(p)$, we have

$$\begin{aligned} \text{Ad}(p) &= \sum_{q \in Q} w(q) \cdot d(p, q) = \sum_{q \in Q} w(q) \cdot (|x(p) - x(q)| + |y(p) - y(q)|) \\ &= \sum_{q \in Q} w(q) \cdot |x(p) - x(q)| + \sum_{q \in Q} w(q) \cdot |y(p) - y(q)| \\ &= \text{Ad}_x(x(p)) + \text{Ad}_y(y(p)), \end{aligned}$$

where

$$\text{Ad}_x(x(p)) = \sum_{q \in Q} w(q) \cdot |x(p) - x(q)| \text{ and } \text{Ad}_y(y(p)) = \sum_{q \in Q} w(q) \cdot |y(p) - y(q)|.$$

If we move p towards q^* along a monotone path, on the x -projection, we are moving $x(p)$ towards $x(q^*)$. By Lemma 3.2.1, $\text{Ad}_x(x(p))$ is monotonically decreasing, so is $\text{Ad}_y(y(p))$. The lemma thus follows. \square

Lemma 3.3.1 implies that $\text{Ad}(p)$ attains a global minimum at $p = q^*$. Hence, q^* is a global minimum point (note that q^* is not necessarily in Q). In the sequel, based on the point q^* and Lemma 3.3.1, we introduce the minimal points and the skyline, and present some observations.

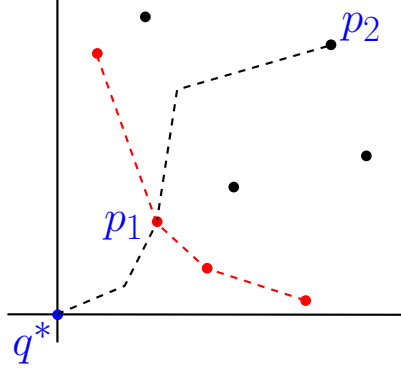


FIGURE 3.1. The four (red) points connected by the dashed line segments (the skyline) are minimal points. p_1 dominates p_2 and the dotted curve connecting q^* and p_2 is a monotone path.

3.3.1. The minimal points and the skyline

We first show how to find z_1 (i.e., the ANN of Q in P^1). For any two different points p_1 and p_2 in P^1 , we say that p_1 *dominates* p_2 if and only if $x(p_1) \leq x(p_2)$ and $y(p_1) \leq y(p_2)$. A point p in P^1 is called a *minimal point* if no point in P^1 dominates p (note that the “minimal” here is different from the “global minimum” defined earlier). If $p_1 \in P^1$ dominates $p_2 \in P^1$, then there exists a monotone path π connecting p_2 and q^* such that $p_1 \in \pi$ (see Fig. 3.1). By Lemma 3.3.1, $\text{Ad}(p_1) \leq \text{Ad}(p_2)$. Therefore, to compute z_1 , we only need to consider the set of minimal points in P^1 , denoted by M . Hence, z_1 is in M .

One tempting approach is to find M and then find z_1 . Unfortunately, since M may have $\Theta(n)$ points, we cannot afford to check every point of M . Below, we give a better approach.

For each $q \in Q$, we induce a horizontal line and a vertical line through q , respectively; let \mathcal{A} be the arrangement of the resulting $2m$ lines. Each cell of \mathcal{A} is a (possibly unbounded) rectangle. Each point in Q is a vertex of \mathcal{A} . Note that we do not explicitly compute \mathcal{A} .

We will show below that $\text{Ad}(p)$ is a linear function of $x(p)$ and $y(p)$ inside any cell C of \mathcal{A} , implying that the ANN (i.e., the top-1 ANN) of Q in $P \cap C$ is on the convex

hull of $P \cap C$, as discussed in [9]. For any cell C , suppose $C = [x_l, x_r] \times [y_b, y_t]$. Let $Q_L = \{q \in Q \mid x(q) \leq x_l\}$, $Q_R = \{q \in Q \mid x(q) \geq x_r\}$, $Q_B = \{q \in Q \mid y(q) \leq y_b\}$ and $Q_T = \{q \in Q \mid y(q) \geq y_t\}$. By the construction of \mathcal{A} , no point of Q lies strictly inside C and $Q = Q_L \cup Q_R = Q_B \cup Q_T$. We have the following lemma.

Lemma 3.3.2. *For any point p in the cell C , $\text{Ad}(p) = C_a \cdot x(p) + C_b \cdot y(p) + C_c$, where*

$$C_a = \sum_{q \in Q_L} w(q) - \sum_{q \in Q_R} w(q), \quad C_b = \sum_{q \in Q_B} w(q) - \sum_{q \in Q_T} w(q), \quad \text{and}$$

$$C_c = \sum_{q \in Q_R} w(q)x(q) - \sum_{q \in Q_L} w(q)x(q) + \sum_{q \in Q_T} w(q)y(q) - \sum_{q \in Q_B} w(q)y(q).$$

Further, with $O(m \log m)$ time preprocessing on Q , given any cell C of \mathcal{A} , we can compute C_a , C_b , and C_c in $O(\log m)$ time.

Proof. The first part (i.e., computing the values of C_a , C_b , and C_c) has been discussed in [9] and it can also be easily verified by our analysis in Lemma 3.3.1. Hence, we omit the proof for it.

For the second part, given any cell C , our goal is to compute the three values C_a , C_b , and C_c . Generally speaking, if, as preprocessing, we compute the prefix sums of the values $w(q)$ and $w(q)x(q)$ in the sorted list of the points of Q by their x -coordinates, and compute the prefix sum of $w(q)y(q)$ in the sorted list of the points of Q by their y -coordinates, then C_a , C_b , and C_c can be computed in $O(\log m)$ time. The details are given below.

To compute C_a , we need to know the value $\sum_{q \in Q_L} w(q)$ and the value $\sum_{q \in Q_R} w(q)$. Note that $\sum_{q \in Q_R} w(q) = W - \sum_{q \in Q_L} w(q)$. We can do the following preprocessing. We sort all points in Q by their x -coordinates. Suppose the sorted list is q_1, q_2, \dots, q_m from left to right. For each $1 \leq j \leq m$, we compute the value $W_1(q_j) = \sum_{i=1}^j w(q_i)$. For any given cell C , let x_l be the x -coordinate of the vertical line containing the

left side of C . By binary search on the sorted list q_1, q_2, \dots, q_m , in $O(\log m)$ time, we can find the rightmost point q' in Q such that $x(q') \leq x_l$. It is easy to see that $\sum_{q \in Q_L} w(q) = W_1(q')$. Note that the above preprocessing takes $O(m \log m)$ time, and C_a can be computed in $O(\log m)$ time.

In similar ways, we can compute C_b and C_c in $O(\log m)$ time, with $O(m \log m)$ preprocessing time. Hence, the second part of the lemma follows. \square

As discussed in [9], Lemma 3.3.2 implies that $S_1(P \cap C)$ (i.e., the ANN of Q in $P \cap C$) is on the convex hull of $P \cap C$. More specifically, $S_1(P \cap C)$ is an extreme point of $P \cap C$ along a certain direction that is determined by C_a and C_b , and thus we can do binary search on the convex hull to find it.

To compute z_1 , the algorithm in [9] checks every cell C of \mathcal{A} in the first quadrant, and it finds $S_1(P \cap C)$ by searching the convex hull of the points in $P \cap C$. The number of cells checked in [9] is $O(m^2)$. In contrast, since $z_1 \in M$, we show below that we only need to check $O(m)$ cells. Although $|M|$ can be $\Theta(n)$, we show that the number of cells of \mathcal{A} that contain these minimal points is $O(m)$, and further, we can find these cells efficiently.

If we order the points in M by their x -coordinates and connect every pair of adjacent points by a line segment, then we can obtain a path π_1 , which we call a *skyline* (see Fig. 3.1). The points of M are also considered as the *vertices* of π_1 . If we move on π_1 from its left endpoint to its right endpoint, then the x -coordinate is monotonically increasing and the y -coordinate is monotonically decreasing. Hence, π_1 is a monotone path.

Denote by \mathcal{C}_1 the set of cells of \mathcal{A} that contain the minimal points in M . Recall that z_1 is in M . We have the following lemma.

Lemma 3.3.3. $|\mathcal{C}_1| = O(m)$ and z_1 is in one of the cells of \mathcal{C}_1 .

Proof. Since z_1 is in M , based on the definition of \mathcal{C}_1 , z_1 must be in one of the cells of \mathcal{C}_1 .

Due to our general position assumption that no two points in $P \cup Q$ have the same x -coordinate or y -coordinate. Each edge of π_1 is neither horizontal nor vertical. Because π_1 is a monotone path, each line of \mathcal{A} can intersect π_1 at most once. Hence, the number of intersections between π_1 and \mathcal{A} is $O(m)$, which implies that the number of cells that intersect π_1 is $O(m)$. Since all points in M are on π_1 , the lemma follows. \square

A straightforward way to compute \mathcal{C}_1 is to first compute \mathcal{A} and then traverse \mathcal{A} by following the skyline π_1 . But this is not efficient due to: (1) computing \mathcal{A} takes $\Theta(m^2)$ time; (2) the size of π_1 may be $\Theta(n)$ due to $|M| = \Theta(n)$ in the worst case. Below in Lemma 3.3.4, we compute \mathcal{C}_1 in $O(m \log n + m \log m)$ time.

First of all, we sort all points in Q by their x -coordinates and y -coordinates, respectively; accordingly, we obtain a sorted list for the horizontal lines of \mathcal{A} and a sorted list for the vertical lines of \mathcal{A} . Then, given any point p , we can determine the cell of \mathcal{A} that contains p in $O(\log m)$ time by doing binary search on the above two sorted lists. We should point out that there might be other ways to compute \mathcal{C}_1 , but the algorithm we propose for Lemma 3.3.4 is particularly useful later when we compute other points in $S_k(P^1)$ than z_1 .

Lemma 3.3.4. *With $O(n \log n)$ time and $O(n)$ space preprocessing on P , given any Q , we can compute the set \mathcal{C}_1 in $O(m \log n + m \log m)$ time.*

Proof. One operation frequently used for computing \mathcal{C}_1 is the following *segment-dragging queries*. Given any horizontal or vertical line segment s , we move s along a given direction perpendicular to s ; the query asks for the first point of P hit by s or reports no such point exists. Chazelle [37] constructed an $O(n)$ -size data structure in $O(n \log n)$ time such that each segment-dragging query can be answered in $O(\log n)$

time. As preprocessing, we build such a data structure on P . Hence, the preprocessing takes $O(n \log n)$ time and $O(n)$ space.

For each cell C of \mathcal{C}_1 , we call the leftmost point of $M \cap C$ the *skyline-left point* of C and call the bottommost point of $M \cap C$ the *skyline-bottom point* of C . In other words, if we move along the skyline π_1 from its left endpoint to its right endpoint, then the skyline-left point of C is the first vertex of π_1 we meet in C and the skyline-bottom point of C is the last vertex of π_1 we meet in C . Note that if C has only one minimal point of M , then the only minimum point is both the skyline-left point and the skyline-bottom point of C .

We will find the skyline-left point and the skyline-bottom point for each cell $C \in \mathcal{C}_1$. Each such point p is determined by a segment-dragging query on a segment s and we call s the *generating segment* of p ; s will be associated with p for later use (for computing other points in $S_k(P^1)$ than z_1). Further, we will classify these generating segments into four types, and again, they will be useful later in Lemma 3.3.7 for computing $S_k(P^1)$.

All the vertical lines passing through points in Q partition the space into $O(m)$ regions, which we refer to as *columns* (including bounding lines). Let \mathcal{D}_M denote the set of columns of \mathcal{A} each of which contains at least one cell of \mathcal{C}_1 . We search the columns of \mathcal{D}_M from left to right. For each column $D \in \mathcal{D}_M$, we will first find the topmost cell and the bottommost cell of \mathcal{C}_1 in D ; then, from the bottommost cell to the topmost cell, we search all other cells of \mathcal{C}_1 in D in a bottom-up fashion. After the searching on D is done, we proceed to the next column of \mathcal{D}_M . The details are given below.

Note that due to the general position assumption that no two points in $P \cup Q$ have the same x - or y -coordinate, each point of P lies strictly inside a cell of \mathcal{A} .

We first determine the leftmost column of \mathcal{D}_M , denoted by D , which is the one containing the leftmost point p_0 of M (see Fig. 3.2). p_0 can be found by the following

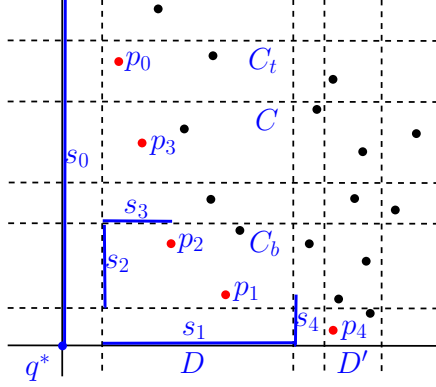


FIGURE 3.2. Illustrating the algorithm in Lemma 3.3.4: the dashed grid is \mathcal{A} .

segment-dragging query. Let $y_{max} = \max_{p \in P^1} y(p)$. Consider a vertical segment $s_0 = \overline{q^*b}$ where $b = (x(q^*), y_{max})$. If we drag s_0 rightwards (i.e., horizontally to the right), p_0 will be the first point of P^1 hit by s_0 . By using the segment-dragging query data structure on P , p_0 can be found in $O(\log n)$ time. After having p_0 , D can be determined in $O(\log m)$ time using binary search on the sorted list of the vertical lines of \mathcal{A} .

Notice that the cell of \mathcal{A} that contains p_0 is the topmost cell in $D \cap \mathcal{C}_1$, which we denote by C_t , and that p_0 is the skyline-left point of C_t (see Fig. 3.2). The segment s_0 is the generating segment of p_0 and we classify s_0 as an s_0 -type. In general, the s_0 -type generating segments are used to find the skyline-left points of the topmost cells of the columns of \mathcal{D}_M .

Next, we determine the bottommost cell of $D \cap \mathcal{C}_1$, denoted by C_b . We first determine the skyline-bottom point p_1 of C_b by a segment-dragging query as follows. Let ℓ denote the horizontal line $y = y(q^*)$. Set $s_1 = D \cap \ell$. If we drag s_1 upwards, p_1 will be the first point of P^1 hit by s_1 (see Fig. 3.2). After p_1 is found in $O(\log n)$ time, C_b can be determined in additional $O(\log m)$ time. s_1 is the generating segment of p_1 and we classify s_1 as the s_1 -type generating segment. In general, s_1 -type generating segments are used to find the skyline-bottom points of the bottommost cells of the columns of \mathcal{D}_M .

If $C_b = C_t$, then the column D contains only one cell of \mathcal{C}_1 , and our searching on D is done. Below, we assume $C_b \neq C_t$.

In the sequel, from the bottommost cell C_b , we search the cells of \mathcal{C}_1 in D in a bottom-up manner until we meet the topmost cell C_t . We first show how to determine the second lowest cell of $\mathcal{C}_1 \cap D$ (i.e., the one of $\mathcal{C}_1 \cap D$ right above C_b), denoted by C_s .

To determine C_s , we first find the skyline-left point p_2 of C_b using a segment-dragging query, as follows. Let s_2 be the left side of C_b . p_2 is the first point in P^1 hit by dragging s_2 rightwards (see Fig. 3.2). s_2 is the generating segment of p_2 and we classify s_2 as the *s₂-type* generating segment. In general, each *s₂-type* generating segment is used to find the skyline-left point of a cell whose skyline-bottom point has just been found. Next, we determine C_s by using p_2 .

We first determine the skyline-bottom point p_3 of C_s . Since $y(p_3) > y(p_2)$, $x(p_3) < x(p_2)$ (otherwise p_2 would dominate p_3). An easy observation is that p_3 is the lowest point among all points of $P^1 \cap D$ whose x -coordinates are less than $x(p_2)$ (see Fig. 3.2). We can determine p_3 by the following segment-dragging query. Let s_3 be the horizontal line segment on the top side of the cell C_b such that the left endpoint of s_3 is the upper left vertex of C_b and the right endpoint has x -coordinate $x(p_2)$ (see Fig. 3.2). Due to our general position assumption that no two points in $P \cup Q$ have the same x - or y -coordinate, p_3 is the point of P^1 hit first by dragging s_3 upwards. After p_3 is found, C_s can be determined. Therefore, we can determine C_s in $O(\log n + \log m)$ time. s_3 is the generating segment of p_3 and we classify s_3 as the *s₃-type* generating segments. In general, *s₃-type* generating segments are used to find the skyline-bottom points for non-bottommost cells of the columns of \mathcal{D}_M .

If $C_s = C_t$, we are done searching on D . Otherwise, we continue the above procedure to search other cells of $\mathcal{C}_1 \cap D$ until we meet the topmost cell C_t .

Now we proceed to the next column $D' \in \mathcal{D}_M$, in the following way. We first

determine D' by a segment-dragging query as follows. Recall that p_1 is the lowest point in $P^1 \cap D$. Let s_4 be the vertical line segment on the right bounding line of D such that the lower endpoint of s_4 has y -coordinate $y(q^*)$ and the upper endpoint has y -coordinate $y(p_1)$ (see Fig. 3.2). We drag the segment s_4 rightwards, and let p_4 be the first point of P^1 hit by s_4 (see Fig. 3.2). It is not difficult to see that p_4 is a minimal point and the column of \mathcal{A} containing p_4 is D' . Further, p_4 is the skyline-left point of the topmost cell of $\mathcal{C}_1 \cap D'$. Hence, after p_4 is found, D' and the topmost cell of $\mathcal{C}_1 \cap D'$ can be determined in $O(\log m)$ time. s_4 is the generating segment of p_4 ; note that s_4 is an s_0 -type generating segment.

Note that if the above segment-dragging query on s_4 fails to find any point (i.e., such a point p_4 does not exist), then all cells of \mathcal{C}_1 have been found, and we terminate. Otherwise, we proceed to search all cells in $\mathcal{C}_1 \cap D'$ in the same way as in the column D , and then search other columns of \mathcal{D}_M similarly.

For the running time, as shown above, the algorithm spends $O(\log n + \log m)$ time finding each cell of \mathcal{C}_1 . Due to $|\mathcal{C}_1| = O(m)$ (by Lemma 3.3.3), computing \mathcal{C}_1 takes $O(m \log n + m \log m)$ time. The lemma thus follows. \square

3.3.2. Computing the top- k ANN set $S_k(P^1)$

In this section, we compute $S_k(P^1)$ in the order of z_1, z_2, \dots, z_k .

Since z_1 is in one of the cells of \mathcal{C}_1 , once we have \mathcal{C}_1 , we compute the ANN of Q in $C \cap P$ in each cell $C \in \mathcal{C}_1$; among the $|\mathcal{C}_1|$ candidate points, z_1 is the one with the smallest aggregate distance to Q . Once z_1 is obtained, we use a similar approach to compute z_2 . Let π_2 be the skyline of $P^1 \setminus \{z_1\}$, and let \mathcal{C}_2 be the set of cells of \mathcal{A} that contain the vertices of π_2 . Again, z_2 must be in one of the cells of \mathcal{C}_2 , and we find z_2 by searching the cells of \mathcal{C}_2 . In general, let π_i be the skyline of $P^1 \setminus \{z_1, \dots, z_{i-1}\}$, and let \mathcal{C}_i be the set of cells of \mathcal{A} that contain the vertices of π_i . The point z_i must be in one of the cells of \mathcal{C}_i , and we find z_i by searching the cells of \mathcal{C}_i . We repeat this

till z_k is found.

For each $1 \leq i \leq k$, since π_i is a skyline, $|\mathcal{C}_i| = O(m)$. A straightforward implementation to compute $S_k(P^1)$ needs to search $O(km)$ cells. We will show that we only need to search $O(k + m)$ cells in total, and more importantly, we can find all these cells efficiently. More specifically, we propose an algorithm that can efficiently determine the set \mathcal{C}_i by updating the set \mathcal{C}_{i-1} , for all $2 \leq i \leq k$.

In the sequel, we first present an algorithm that can quickly compute the ANN of Q in $C \cap P$ for any cell C of \mathcal{A} . An $O(n \log^2 n)$ -size data structure was given in [9] that can be built in $O(n \log^2 n)$ time and can compute the ANN in any cell C of \mathcal{A} in $O(\log^3 n)$ time. By using compact interval trees [65], we have the following improved result in Lemma 3.3.5.

Lemma 3.3.5. *For a set P of n points in the plane, an $O(n \log n \log \log n)$ -size data structure can be built in $O(n \log n \log \log n)$ time, such that given any axis-parallel rectangle C (e.g., any cell of \mathcal{A}), the ANN of Q in $P \cap C$ can be computed in $O(\log^2 n)$ time. With trade-off between preprocessing and query time, we can build two other data structures: the first one has $O(n \log n)$ preprocessing time and space with $O(\log^2 n \log \log n)$ query time; the second one has $O(n \log n \log^* n)$ preprocessing time and space with $O(\log^2 n \log^* n)$ query time.*

Proof. Our data structure uses the compact interval tree [65], which was for solving the following *sub-path hull queries* in [65]. Let π be a simple path of n vertices in the plane and suppose the vertices are v_1, v_2, \dots, v_n ordered along π . Given two vertex indices i and j with $i < j$, the *sub-path hull query* asks for the convex hull of all vertices v_i, v_{i+1}, \dots, v_j . A compact interval tree data structure was given in [65], and for each sub-path hull query, it can report in $O(\log n)$ time a data structure that represents the convex hull such that any standard binary-search based operation on the convex hull can be implemented in $O(\log n)$ time (e.g., finding an extreme point

on the convex hull along any given direction). Assume the vertices of π are sorted by their x - or y -coordinates. The compact interval tree is of $O(n \log \log n)$ size and can be built in the same time. With trade-off between preprocessing and query time, two other compact interval trees can be built for the sub-path hull queries: the first one has $O(n)$ preprocessing time and space with $O(\log n \log \log n)$ query time; the second one has $O(n \log^* n)$ preprocessing time and space with $O(\log n \log^* n)$ query time.

Our data structure for the lemma is constructed as follows. At the high-level, it is similar to the two-dimensional orthogonal range tree [52]. A balanced binary search tree T is built based on the x -coordinates of the points in P . The leaves of T store the points of P in sorted order from left to right, and the internal nodes store splitting values to guide the search on T . For each node v of T , it also stores the subset $P(v) \subseteq P$ of points in the subtree of T rooted at v , and $P(v)$ is called the *canonical subset* of v . For each canonical subset $P(v)$, we build a compact interval tree in the following way. If we sort the points of $P(v)$ by their y -coordinates and connect each pair of adjacent points in the sorted list by a line segment, we obtain a path $\pi(v)$. The points in $P(v)$ are vertices of $\pi(v)$. Note that $\pi(v)$ is a simple path and each horizontal line intersects $\pi(v)$ at most once. We build a compact interval tree on $\pi(v)$ using the approaches in [65]. This finishes the construction of our data structure.

For each canonical subset $P(v)$, depending on which of the three compact interval trees is used, constructing the compact interval tree on $\pi(v)$ takes $O(\mu \log \log \mu)$ (or $O(\mu)$ or $O(\mu \log^* \mu)$) time and space, where $\mu = |P(v)|$. Note that the y -sorted list of $P(v)$ can be built during the construction of T in a bottom-up manner. Hence, the preprocessing time and space is $O(n \log n \log \log n)$ (or $O(n \log n)$ or $O(n \log n \log^* n)$), the same as claimed in the lemma statement.

Given any axis-parallel rectangle C , our goal is to find the ANN of Q in $C \cap P$. Essentially, we are looking for an extreme point in $C \cap P$ along a certain direction,

denoted by σ . As discussed in [9], σ is determined by the two factors C_a and C_b defined in Lemma 3.3.2, and can be computed in $O(\log m)$ time by Lemma 3.3.2. Recall that $m < n$; hence, $\log m = O(\log n)$.

Suppose $C = [x_l, x_r] \times [y_b, y_t]$. Using the range $[x_l, x_r]$, we first find the $O(\log n)$ canonical subsets whose union is the set of points in P lying between the two vertical lines $x = x_l$ and $x = x_r$. For each such canonical subset $P(v)$, we use the range $[y_b, y_t]$ to determine the sub-path of $\pi(v)$ inside C , which can be done by binary search on the y -sorted list of $P(v)$; subsequently, we use the compact interval tree data structure on $\pi(v)$ to (implicitly) report the convex hull of the sub-path in $O(\log n)$ time (or $O(\log n \log \log n)$ or $O(\log n \log^* n)$ time), after which we search the extreme point on the convex hull along the direction σ in $O(\log n)$ time. In this way, we obtain $O(\log n)$ extreme points for these $O(\log n)$ canonical subsets, and the one minimizing the aggregate distance to Q is the ANN of Q in $C \cap P$. Assuming that we have computed the three factors C_a , C_b , and C_c as defined in Lemma 3.3.2, for each extreme point found above, its aggregate distance to Q can be computed in constant time.

Therefore, the ANN of Q in $C \cap P$ can be found in $O(\log^2 n)$ time (or $O(\log^2 n \log \log n)$ or $O(\log^2 n \log^* n)$ time). The lemma thus follows. \square

In the following, to avoid the tedious discussion, unless otherwise stated, when we refer to Lemma 3.3.5 we will always use the data structure with $O(\log^2 n)$ query time, with the understanding that using different data structures will give different performances (i.e., preprocessing and query time) accordingly.

Let $m_i = |\mathcal{C}_i|$ for each $1 \leq i \leq k$. By Lemma 3.3.5, we can determine z_1 in $O(m_1 \log^2 n)$ time. Next we compute z_2 . To this end, we need to find the set \mathcal{C}_2 first. Instead of computing \mathcal{C}_2 from scratch as we did for \mathcal{C}_1 , we obtain \mathcal{C}_2 by updating \mathcal{C}_1 . Specifically, if some cells are both in \mathcal{C}_1 and \mathcal{C}_2 , we do not need to compute them

again. In other words, we only need to compute the cells in $\mathcal{C}_2 \setminus \mathcal{C}_1$. Let $C(z_1)$ denote the cell containing z_1 . We will show that all the cells of \mathcal{C}_1 except $C(z_1)$ must be in \mathcal{C}_2 . The cell $C(z_1)$ may or may not be in \mathcal{C}_2 . If $C(z_1) \in \mathcal{C}_2$, then special care needs to be taken when searching $C(z_1)$ because we are looking for z_2 and the point z_1 should not be considered any more. The details are given below.

For each i with $2 \leq i \leq k$, let $\mathcal{C}'_i = \mathcal{C}_i \setminus \mathcal{C}_{i-1}$ and $m'_i = |\mathcal{C}'_i|$. We first show that \mathcal{C}_2 can be obtained in $O(m'_2(\log^2 n + \log m))$ time, and specifically, we compute the cells of \mathcal{C}'_2 and determine whether $C(z_1) \in \mathcal{C}_2$, which is done in Lemma 3.3.7.

The algorithm in Lemma 3.3.7 needs a dynamic version of the segment-dragging query data structure that can support point deletions for P . Later after we finish the query, we also need to insert those points that have been deleted back to P , and we call them *special insertions*, i.e., whenever we insert a point p to P for the segment-dragging query data structure, p has already been deleted from P before. In the following Lemma 3.3.6, we present such a dynamic data structure for the segment-dragging queries by using the range trees [52]. Note that the performance of the data structure in Lemma 3.3.6 may not be theoretically the best: since other parts of our algorithm for computing $S_k(P^1)$ dominate the overall running time, we choose to present a data structure that is simple and does not affect the overall performance.

Lemma 3.3.6. *With $O(n \log n)$ time and $O(n \log n)$ space preprocessing on P , we can answer each segment-dragging query in $O(\log^2 n)$ time and support each point deletion and special insertion for P in $O(\log^2 n)$ time.*

Proof. Our data structure consists of two range trees, one for horizontal segment-dragging queries and the other for vertical segment-dragging queries. Below, we only present the one for horizontal segment-dragging queries and the other one can be obtained similarly.

We first sort the points in P by their x -coordinates and y -coordinates, respectively.

We build a balanced binary search tree T based on the x -coordinates of the points in P . The leaves of T store the points of P in sorted order from left to right. Each node v of T also stores the subset $P(v)$ of points stored in the leaves of the subtree rooted at v ; $P(v)$ is called the *canonical subset* of v . For each node v , we use another balanced binary search tree $T(v)$ to store the points in $P(v)$ based on the y -coordinates of the points. It is commonly known that T can be constructed in $O(n \log n)$ time using $O(n \log n)$ space [52].

Consider any segment-dragging query. Without loss of generality, assume we drag upwards a horizontal segment $s = [x_1(s), x_2(s)] \times \{y(s)\}$ (i.e., its y -coordinate is $y(s)$ and its x -coordinate spans the interval $[x_1(s), x_2(s)]$). We first determine the $O(\log n)$ canonical subsets of T whose union is the subset of points of P with x -coordinates lying in $[x_1(s), x_2(s)]$. For each canonical subset $P(v)$, we use the tree $T(v)$ to determine in $O(\log n)$ time the lowest point of $P(v)$ whose y -coordinate is no less than $y(s)$ and that point will be the first point hit by dragging s upwards. After we find such a point in each canonical subset, we report the point with smallest y -coordinate as the answer to the segment-dragging query for s . The total query time is $O(\log^2 n)$ time.

Now consider deleting a point p from P . We first find the leaf v_p of T storing p . Then, for each node v in the path of T from the root to v_p , we delete p from the tree $T(v)$, which can be done in $O(\log n)$ time. Hence, it takes $O(\log^2 n)$ time for each point deletion. Consider a special insertion that inserts a point p to P . Since it is a special insertion, p was in P before but has been deleted. We first find the leaf v_p of T that stored p before. Then, for each node v in the path of T from the root to v_p , we insert p to the tree $T(v)$, which can be done in $O(\log n)$ time. Hence, it takes $O(\log^2 n)$ time for each special insertion.

The lemma thus follows. □

Next in Lemma 3.3.7, we compute \mathcal{C}_2 based on \mathcal{C}_1 . The algorithm for Lemma 3.3.7 essentially follows the behavior of the algorithm for Lemma 3.3.4, but only focuses on searching the cells of \mathcal{C}'_2 . The efficiency of the algorithm for Lemma 3.3.7 also hinges on the observation that the cells of \mathcal{C}'_2 form at most two subsets (separated by $C(z_1)$ if $C(z_1) \in \mathcal{C}_2$) of consecutive cells of \mathcal{C}_2 if we order the cells of \mathcal{C}_2 from “northwest” to “southeast”.

Lemma 3.3.7. *We can determine the set \mathcal{C}_2 in $O((1 + m'_2)(\log^2 n + \log m))$ time, where $m'_2 = |\mathcal{C}'_2|$, and more specifically, our algorithm will compute the cells of \mathcal{C}'_2 and determine whether $C(z_1) \in \mathcal{C}_2$.*

Proof. We call the order of the cells of \mathcal{C}_1 by which the skyline π_1 crosses them from left to right the *canonical order* of \mathcal{C}_1 . In other words, the canonical order of \mathcal{C}_1 follows the northwest-to-southeast order. We define the canonical order of \mathcal{C}_2 similarly.

Suppose the canonical order of the cells of \mathcal{C}_1 is: C_1, C_2, \dots, C_{m_1} . Note that we can obtain this ordered list during computing \mathcal{C}_1 in Lemma 3.3.4 within the same time. Recall that when computing \mathcal{C}_1 we also computed a skyline-left point and a skyline-bottom point for each cell of \mathcal{C}_1 as well as their generating segments. Let $C_i = C(z_1)$, i.e., the cell that contains z_1 . We assume $i \neq 1$ and $i \neq m_1$ (otherwise the algorithm is similar and much simpler).

In order to better understand the algorithm we will present below, we first discuss a question: which cells are possibly in \mathcal{C}'_2 ? Imagine that we partition the plane into four quadrants with respect to z_1 by the vertical line through z_1 and the horizontal line through z_1 ; an easy observation is that only the cells intersecting the first quadrant can possibly be in \mathcal{C}'_2 because only points in the first quadrant are dominated by z_1 . Further, for each cell C_j with $j \neq i$, none of the vertices of the skyline π_1 in C_j is dominated by z_1 , and thus C_j is still in \mathcal{C}_2 . In other words, all cells of $\mathcal{C}_1 \setminus \{C_i\}$ are still in \mathcal{C}_2 . The cell C_i may or may not be in \mathcal{C}_2 . Also note that if we remove z_1

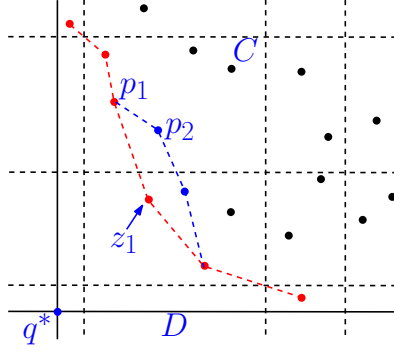


FIGURE 3.3. The red points are in π_1 and the blue points are in $\pi_2 \setminus \pi_1$. The skyline-bottom point of the cell C is p_1 in π_1 but p_2 in π_2 .

from P , then the skyline-bottom point of C_{i-1} may be changed (see Fig. 3.3), but the skyline-left point of C_{i-1} does not change; for each cell C_j with $1 \leq j \leq i-2$, neither its skyline-left point nor its skyline-bottom point changes. Similarly, due to the removal of z_1 , the skyline-left point of C_{i+1} may be changed, but its skyline-bottom point does not change; for each cell C_j with $i+2 \leq j \leq m_1$, neither its skyline-left point nor its skyline-bottom point changes.

The above implies that to determine \mathcal{C}_2 , we need to do the following. (1) Find all cells in \mathcal{C}'_2 , and as in Lemma 3.3.4, for each cell of \mathcal{C}'_2 , compute its skyline-left point and skyline-bottom point as well as their generating segments. (2) Determine whether C_i is still in \mathcal{C}_2 , and if yes, compute its new skyline-left point and skyline-bottom point as well as their generating segments, if any of them changes. (3) Compute the new skyline-bottom point (and its generating segment) for C_{i-1} if it changes. (4) Compute the new skyline-left point (and its generating segment) for C_{i+1} if it changes.

Let \mathcal{D}_P be the dynamic segment-dragging data structure in Lemma 3.3.6 we built on P . Below, we give an algorithm that can determine \mathcal{C}_2 in $O((1+m'_2)(\log^2 n + \log m))$ time, and in particular, we need to find the cells of \mathcal{C}'_2 . Intuitively, if $C_i \notin \mathcal{C}_2$, then \mathcal{C}'_2 consists of all cells of \mathcal{C}_2 between C_{i-1} and C_{i+1} in the canonical order; otherwise, \mathcal{C}'_2 consists of all cells of \mathcal{C}_2 between C_{i-1} and C_i and all cells between C_i and C_{i+1} .

Our algorithm essentially follows the behavior of the algorithm in Lemma 3.3.4, but only focuses on the cells in $\mathcal{C}'_2 \cup \{C_{i-1}, C_i, C_{i+1}\}$. Recall that C_i is the cell of \mathcal{C}_1 that contains z_1 .

First of all, we delete the point z_1 from the data structure \mathcal{D}_P . The point z_1 can be the skyline-left point of C_i , or the skyline-bottom point of C_i , or both of them, or neither of them. Our algorithm works differently for these cases, as follows. Recall that according to our algorithm in Lemma 3.3.4, if z_1 is either the skyline-left point or the skyline-bottom point of C_i , then z_1 has a generating segment, denoted by $s(z_1)$. In other words, z_1 is identified by a segment-dragging query on $s(z_1)$ in our algorithm in Lemma 3.3.4.

1. If z_1 is neither the skyline-left point nor the skyline-bottom point of C_i , then C_i is still in \mathcal{C}_2 and $\mathcal{C}'_2 = \emptyset$. In fact, $\mathcal{C}_2 = \mathcal{C}_1$. Further, the skyline-left and skyline-bottom points of any cell of \mathcal{C}_1 do not change. Hence, we are done for this case.
2. If z_1 is the skyline-left point but not the skyline-bottom point, then according to our algorithm in Lemma 3.3.4, the generating segment $s(z_1)$ is either an s_2 -type or an s_0 -type. Note that since z_1 is not the skyline-bottom point of C_i , the skyline-bottom point of C_i is still in the skyline π_2 , which implies that C_i is still in \mathcal{C}_2 and no cell of \mathcal{C}'_2 is between C_i and C_{i+1} in the canonical order of \mathcal{C}_2 . In other words, all cells of \mathcal{C}'_2 are between C_{i-1} and C_i in the canonical order of \mathcal{C}_2 . Denote by D the column of \mathcal{A} that contains C_i .
 - (a) If $s(z_1)$ is an s_2 -type, then C_i is not the topmost cell of \mathcal{C}_1 in the column D , which implies that C_{i-1} is in D . According to the algorithm in Lemma 3.3.4, $s(z_1)$ is the left side of C_i (i.e., z_1 is the first point of P hit by dragging $s(z_1)$ rightwards). By using the data structure \mathcal{D}_P (after deleting

z_1), we do a segment-dragging query by dragging $s(z_1)$ rightwards to find the first point of $P \setminus \{z_1\}$ hit by $s(z_1)$, and we denote the point by p . Then, p is the new skyline-left point of C_i (without considering z_1). Note that $s(z_1)$ is still an s_2 -type generating segment for p .

Next, from C_i , we continue to find the cells of \mathcal{C}'_2 in a bottom-up manner in the same way as the algorithm in Lemma 3.3.4 until we meet the cell C_{i-1} . Note that it is possible that $\mathcal{C}'_2 = \emptyset$. Again, it takes two segment-dragging queries (using \mathcal{D}_P) on each cell of \mathcal{C}'_2 to find its skyline-left and skyline-bottom point as well as their generating segments. Also, the algorithm will find the new skyline-bottom point of C_{i-1} if it changes in π_2 . Recall that given any point p , we can determine the cell of \mathcal{A} that contains p in $O(\log m)$ time (by binary search on the sorted vertical lines of \mathcal{A} and on the sorted horizontal lines of \mathcal{A}). Therefore, in this case, the total running time to determine \mathcal{C}_2 is $O((1 + m'_2)(\log^2 n + \log m))$ time.

- (b) If $s(z_1)$ is an s_0 -type, then C_i is the topmost cell of \mathcal{C}_1 in the column D , which implies that C_{i-1} is in a column to the left of D . Denote by D' the column of \mathcal{A} containing C_{i-1} and let p be the skyline-bottom point of C_{i-1} . According to the algorithm in Lemma 3.3.4, $s(z_1)$ is the vertical line segment on the right side of D' where the lower endpoint of $s(z_1)$ is on the horizontal line $y = y(q^*)$ and the upper endpoint has the same y -coordinate as p , and z_1 is the first point of P hit by dragging $s(z_1)$ rightwards.

By using the data structure \mathcal{D}_P (after deleting z_1), we do a segment-dragging query by dragging $s(z_1)$ rightwards; let p' be the point returned by the query (i.e., p' is the first point of $P \setminus \{z_1\}$ hit by dragging $s(z_1)$ rightwards). Note that $s(z_1)$ is still an s_0 -type generating segment for p' .

- i. If p' is in C_i , then p' is the new skyline-left point of C_i , and C_i is still

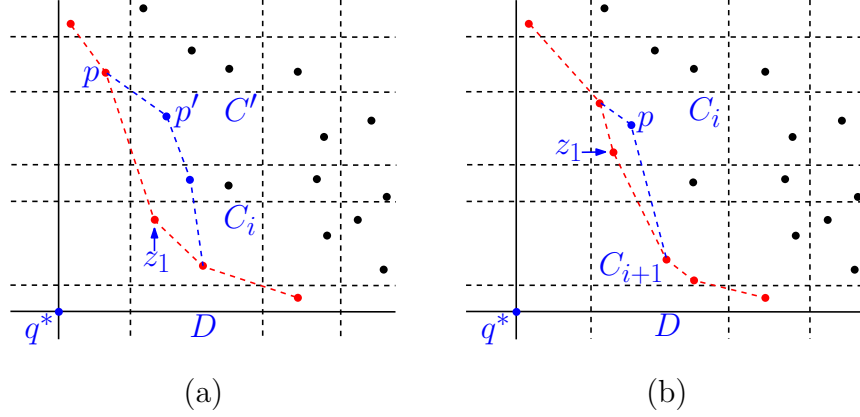


FIGURE 3.4. (a). Illustrating Case 2(b)ii: If $C' \neq C_i$, then C' is in D and higher than C_i . (b). Illustrating Case 3b: p is the first point hit by dragging $s(z_1)$ upwards without considering z_1 .

the topmost cell of \mathcal{C}_2 in D , which implies $\mathcal{C}'_2 = \emptyset$.

- ii. If p' is not in C_i , then let C' be the cell containing p' and p' is the skyline-left point of C' . Since z_1 is not the skyline-bottom point of C_i , the cell C' is still in the column D and is higher than C_i (see Fig. 3.4(a)). Then, from the cell C_i to C' , we use the bottom-up procedure as in the algorithm in Lemma 3.3.4 to find the cells of \mathcal{C}_2 between C_i and C' in the column D and these cells (except C_i) constitute the set \mathcal{C}'_2 . Again, it takes two segment-dragging queries (using \mathcal{D}_P) for each cell of \mathcal{C}'_2 to find its skyline-left and skyline-bottom point as well as their generating segments.

The total running time is $O((1 + m'_2)(\log^2 n + \log m))$ time.

3. If z_1 is the skyline-bottom point but not the skyline-left point, then according to our algorithm in Lemma 3.3.4, $s(z_1)$ is either an s_1 -type or an s_3 -type. Note that since z_1 is not the skyline-left point of C_i , the skyline-left point of C_i is still in the skyline π_2 , which implies that C_i is still in \mathcal{C}_2 and no cell of \mathcal{C}'_2 is between C_{i-1} and C_i in the canonical order of \mathcal{C}_2 . In other words, all cells of \mathcal{C}'_2 are between C_i and C_{i+1} in the canonical order of \mathcal{C}_2 . Denote by D the column of \mathcal{A} that contains C_i .

- (a) If $s(z_1)$ is an s_1 -type, then C_i is the bottommost cell of \mathcal{C}_1 in D . According to the algorithm in Lemma 3.3.4, $s(z_1)$ is the intersection of D and the horizontal line $y = y(q^*)$. By using the data structure \mathcal{D}_P (after deleting z_1), we do a segment-dragging query by dragging $s(z_1)$ upwards and let p be the point returned by the query. Then p is the new skyline-bottom point of C_i . Next, we find the cells in \mathcal{C}'_2 .

Let s' be the vertical segment on the right side of D where the lower endpoint of s' is on the horizontal line $y = y(q^*)$ and the upper endpoint of s' has the same y -coordinate as p . We do a segment-dragging query by dragging s' rightwards and let p' be the point given by the query. The segment s' is the generating segment of p' , and in fact, s' is an s_0 -type generating segment by our definition in the proof of Lemma 3.3.4. Denote by $C(p')$ the cell of \mathcal{A} that contains p' . Let D' be the column that contains C_{i+1} .

- i. If $C(p')$ is in D' , then there are further two cases. If $C(p')$ is C_{i+1} , then p' is the new skyline-left point of C_{i+1} , and $\mathcal{C}'_2 = \emptyset$. Otherwise, from C_{i+1} to $C(p')$, we use the same bottom-up procedure as in the algorithm in Lemma 3.3.4 to find all cells of \mathcal{C}_2 between C_{i+1} and $C(p')$, and these cells (except C_{i+1}) constitute the set \mathcal{C}'_2 .
 - ii. If $C(p')$ is not in D' , then it must be in a column to the left of D' . From the cell $C(p')$, we proceed in the same way as in the algorithm in Lemma 3.3.4 until the first time we find a cell in the column D' . Then, we use the same algorithm as the above case where $C(p')$ is in D' .
- (b) If $s(z_1)$ is an s_3 -type, then C_i is the not bottommost cell of \mathcal{C}_1 in D , which implies that C_{i+1} is in D . We show below that $\mathcal{C}'_2 = \emptyset$; further, we will

find a new skyline-bottom point in C_i (without considering z_1).

Based on our algorithm in Lemma 3.3.4, the generating segment $s(z_1)$ of z_1 is the horizontal line segment on the top side of C_{i+1} whose left endpoint is the upper left vertex of C_{i+1} and right endpoint has the same x -coordinate as the skyline-left point of C_{i+1} . By using the data structure \mathcal{D}_P (after deleting z_1), we do a segment-dragging query by dragging $s(z_1)$ upwards, and let p be the point returned by the query.

Note that z_1 is the lowest point of P that will be hit by dragging $s(z_1)$ upwards and z_1 is in C_i . The point p is the lowest point of $P \setminus \{z_1\}$ that will be hit by dragging $s(z_1)$ upwards (see Fig. 3.4(b)). Clearly, p cannot be any cell of D lower than C_i . On the other hand, since z_1 is not the skyline-left point of C_i , the skyline-left point of C_i is still in C_i . Note that when we drag $s(z_1)$ upwards, the skyline-left point of C_i will be hit by $s(z_1)$ (but not necessarily the first point hit by $s(z_1)$), and this implies that the point p must be in C_i . In other words, p is the skyline-bottom point of C_i in the new skyline π_2 , and further $\mathcal{C}'_2 = \emptyset$.

In any case above, the total running time is $O((1 + m'_2)(\log^2 n + \log m))$ time.

4. It remains to discuss the case where z_1 is both the skyline-left point and the skyline-bottom point of C_i .

In this case, z_1 is the first time identified as either the skyline-left point or the skyline-bottom point. In general, unlike the second and the third cases where the cells of \mathcal{C}'_2 are either between C_{i-1} and C_i or between C_i and C_{i+1} in the canonical order of \mathcal{C}_2 , in this case the cells of \mathcal{C}'_2 may lie both between C_{i-1} and C_i and between C_i and C_{i+1} . Hence, our algorithm may need to search on both “directions”. In addition, in the previous three cases, the cell C_i must be in \mathcal{C}_2 ; in this case, however, it is possible that C_i is not in \mathcal{C}_2 .

(a) If z_1 is the first time identified as the skyline-left point of C_i , then C_i must be the topmost cell of \mathcal{C}_1 in D where D is the column of \mathcal{A} that contains C_i , which implies that its generating segment $s(z_1)$ must be an s_0 -type. Let p be the skyline-bottom point of the cell C_{i-1} . Let D' be the column of \mathcal{A} that contains C_{i-1} . According to the algorithm in Lemma 3.3.4, $s(z_1)$ is the vertical line segment on the right side of D' where the lower endpoint of $s(z_1)$ is on the horizontal line $y = y(q^*)$ and the upper endpoint has the same y -coordinate as p . By using the data structure \mathcal{D}_P (after deleting z_1), we do a segment-dragging query by dragging $s(z_1)$ rightwards, and let p' be the point given by the query. Let $C(p')$ be the cell that contains p' . Note that p' is the skyline-left point of $C(p')$. Let D'' be the column that contains the cell C_{i+1} . Note that it is possible that $D'' = D$.

i. If $C(p')$ is also in D'' , then there are further two cases.

If $C(p') = C_{i+1}$, then $\mathcal{C}'_2 = \emptyset$ and $C_i \notin \mathcal{C}_2$.

Otherwise, $C(p')$ must be higher than C_{i+1} in D'' . Then, from the cell C_{i+1} to $C(p')$, we use the same bottom-up procedure as in the algorithm in Lemma 3.3.4 to find all cells of \mathcal{C}_2 between C_{i+1} and $C(p')$, and these cells (except C_{i+1} and possibly C_i) constitute the set \mathcal{C}'_2 . Note that the cell C_i may or may not be identified as in \mathcal{C}_2 in the above procedure.

ii. If $C(p')$ is not in D'' , then $C(p')$ must be in a column to the left of D'' . We proceed from $C(p')$ in the same way as in the algorithm in Lemma 3.3.4 until the first time we find a cell in D'' . Then, we use the same algorithm as in the above case (i.) to determine \mathcal{C}_2 .

In any case, the total running time is $O((1 + m'_2)(\log^2 n + \log m))$ time.

(b) If z_1 is the first time identified as the skyline-bottom point, then its

generating segment $s(z_1)$ can be either an s_1 -type or an s_3 -type segment. Let D be the column that contains C_i . In this case, C_i is not the topmost cell of \mathcal{C}_2 in D since otherwise z_1 would be the first time identified as the skyline-left point of C_i . This means that C_{i-1} is also in D .

- i. If $s(z_1)$ is an s_1 -type segment, then C_i must be the bottommost cell of \mathcal{C}_1 in the column D . According to the algorithm in Lemma 3.3.4, $s(z_1)$ is the intersection of D and the horizontal line $y = y(q^*)$. By using the data structure \mathcal{D}_P (after deleting z_1), we do a segment-dragging query by dragging $s(z_1)$ upwards and let p be the point returned by the query. Let $C(p)$ be the cell that contains p . Clearly, $C(p)$ is in \mathcal{C}_2 . Let \mathcal{C}'_{21} be the subset of cells in \mathcal{C}'_2 that are between C_{i-1} and $C(p)$ in the canonical order of \mathcal{C}_2 , and let $\mathcal{C}'_{22} = \mathcal{C}'_2 \setminus \mathcal{C}'_{21}$; in other words, \mathcal{C}'_{22} is the subset of cells in \mathcal{C}'_2 that are between $C(p)$ and C_{i+1} in the canonical order of \mathcal{C}_2 . Below, we will find \mathcal{C}'_{21} and \mathcal{C}'_{22} separately, by searching from $C(p)$ towards two “directions”: one towards C_{i-1} and the other towards C_{i+1} .

Since C_{i-1} is also in D , from $C(p)$ to C_{i-1} , we use the bottom-up procedure to find the cells of \mathcal{C}_2 between $C(p)$ and C_{i-1} , and these cells (except C_{i-1} and possibly C_i) constitute the set \mathcal{C}'_{21} . Note that the cell C_i may also be identified in \mathcal{C}_2 .

Next, we find the set \mathcal{C}'_{22} , which can be done by the same algorithm as in Case 3a. We omit the details.

- ii. If $s(z_1)$ is an s_3 -type, then C_i is the not bottommost cell of \mathcal{C}_1 in D , which implies that C_{i+1} is in D .

Based on our algorithm in Lemma 3.3.4, the generating segment $s(z_1)$ of z_1 is the horizontal line segment on the top side of C_{i+1} whose left

endpoint is the upper left vertex of C_{i+1} and right endpoint has the same x -coordinate as the skyline-left point of C_{i+1} . By using the data structure \mathcal{D}_P (after deleting z_1), we do a segment-dragging query by dragging $s(z_1)$ upwards, and let p be the point returned by the query. Let $C(p)$ be the cell that contains p .

Note that z_1 is the lowest point of P that will be hit by dragging $s(z_1)$ upwards and z_1 is in C_i . The point p is the lowest point of $P \setminus \{z_1\}$ that will be hit by dragging $s(z_1)$ upwards. Since C_{i-1} is also in D , C_{i-1} is higher than C_i . Hence, the cell $C(p)$ is one of the cells of D between (and including) C_{i-1} and C_i (this is because the vertices of π_1 in C_{i-1} are all to the left of the right endpoint of $s(z_1)$ and to the right of the left endpoint of $s(z_1)$). Hence, from $C(p)$ to C_{i-1} , we can use the bottom-up procedure as before to find the cells of \mathcal{C}_2 , these cells (except C_{i-1} and possibly C_i) constitute the set \mathcal{C}'_2 .

In any case, the total running time is $O((1 + m'_2)(\log^2 n + \log m))$ time.

In summary, we can determine the set \mathcal{C}_2 in $O((1 + m'_2)(\log^2 n + \log m))$ time. More specifically, for each cell in \mathcal{C}'_2 , we have computed its skyline-left point and its skyline-bottom point as well as their generating segments. We have also determined whether C_i is in \mathcal{C}_2 , and if yes, its new skyline-bottom point and skyline-left point are computed if any of them changes. The new skyline-bottom point of C_{i-1} has been found if it changes, and the new skyline-left point of C_{i+1} has been found if it changes. In addition, in the above algorithm, we can also order all cells of \mathcal{C}'_2 (with C_i if $C_i \in \mathcal{C}_2$) from northwest to southeast with the same running time, and therefore, along with the ordered cells from C_1 to C_{i-1} and the ordered cells from C_{i+1} to C_{m_1} , we have obtained a canonical order for \mathcal{C}_2 . \square

By Lemma 3.3.7, we can determine the set \mathcal{C}_2 , and in particular, we have the set

\mathcal{C}'_2 explicitly, and we know whether the cell $C(z_1) \in \mathcal{C}_2$. Similarly to Lemma 3.3.3, the second ANN z_2 is in one of the cells of \mathcal{C}_2 . Denote by $P_1 = P \setminus \{z_1\}$.

To find z_2 , as in the case for finding z_1 , a straightforward approach is to compute the ANN of Q in $P_1 \cap C$ for each $C \in \mathcal{C}_2$, and then among the $|\mathcal{C}_2|$ candidate points, report the one with the smallest aggregate distance to Q as z_2 . This approach will lead to an $O(km)$ time query algorithm for finding $S_k(P^1)$. Below, we present a better method.

Note that when computing z_1 , we have computed the ANN $S_1(P \cap C)$ for each $C \in \mathcal{C}_1$. Also, for each cell $C \in \mathcal{C}_1$, if $C \neq C(z_1)$, then $C \in \mathcal{C}_2$ and $P \cap C = P_1 \cap C$. Therefore, if we maintain the ANNs for all cells of $\mathcal{C}_1 \setminus C(z_1)$, we do not have to compute them again. In other words, when computing z_2 , we only need to compute the ANNs in the cells of \mathcal{C}'_2 . In addition, if $C(z_1) \in \mathcal{C}_2$, we will use a special approach to compute $S_1(P_1 \cap C(z_1))$. To maintain the ANNs in the involved cells mentioned above, we use a min-heap H , as follows.

When searching z_1 , for each $C \in \mathcal{C}_1$, after the ANN $S_1(P \cap C)$ is computed, we insert it into H with its aggregate distance to Q as the “key”. After the ANNs for all cells of \mathcal{C}_1 are computed and inserted into H , the point in H with the smallest key is z_1 . Note that H has $m_1 = |\mathcal{C}_1|$ points. To compute first determine \mathcal{C}'_2 by Lemma 3.3.7. By the “Extract-Min” operation of min-heaps [49], we remove z_1 from H . We compute the ANNs of the cells in \mathcal{C}'_2 and insert them into H . If $C(z_1) \notin \mathcal{C}_2$, then the point of H with the smallest key is z_2 . Otherwise, we use the following special approach to determine $S_1(P_1 \cap C(z_1))$.

One tempting approach is to have a dynamic version of the data structure in Lemma 3.3.5 to support point deletions from P . Unfortunately, due to the “static” nature of compact interval trees, it is not clear to us how to design such a dynamic data structure without deteriorating the performance. Instead, we present another method to “mimic” point deletions, as follows.

We divide the cell $C(z_1)$ into two sub-cells $C_1(z_1)$ and $C_2(z_1)$ using the horizontal line through z_1 . Hence, z_1 is on the common edge of the two sub-cells. Due to our general position assumption, no point of P is on the boundary of $C(z_1)$. Hence, no point of $P_1 = P \setminus \{z_1\}$ is on the boundary of $C_1(z_1)$ (or $C_2(z_1)$). Below, we use $C_1(z_1)$ (resp., $C_2(z_1)$) to refer to only its interior. Instead of computing the ANN $S_1(P_1 \cap C(z_1))$ and insert it into H , we compute the ANNs $S_1(P \cap C_1(z_1))$ and $S_1(P \cap C_2(z_1))$ and insert them into H ; note that one of them is $S_1(P_1 \cap C(z_1))$. The reason we divide $C(z_1)$ into two sub-cells as above is that we can now simply use the data structure in Lemma 3.3.5 to compute $S_1(P \cap C_1(z_1))$ and $S_1(P \cap C_2(z_1))$; in other words, z_1 appears to be “deleted” from the data structure of Lemma 3.3.5. Clearly, now, the point of H with smallest key is z_2 .

To analyze the running time for computing z_2 , \mathcal{C}_2 can be determined in $O((1 + m'_2)(\log^2 n + \log m))$ time, after which, we compute the ANNs for the cells of \mathcal{C}'_2 and possibly for the two sub-cells of $C(z_1)$ in $O((2 + m'_2) \log^2 n)$ time by Lemma 3.3.5. Then, one “Extract-Min” operation and at most $m'_2 + 2$ insertions on H together take $O((m'_2 + 3) \log(|H|))$ time; note that $|H| \leq m_1 + m'_2 + 2$ (here “2” corresponds to the number of possible sub-cells).

It should be noted that we need to explicitly maintain the two sub-cells $C_1(z_1)$ and $C_2(z_1)$ because later they may be further divided into smaller sub-cells (e.g., if $z_2 \in C_1(z_1)$ and $C(z_1) \in \mathcal{C}_3$, then $C_1(z_1)$ will be divided for computing z_3). Also note that these sub-cells are only maintained for computing ANNs and they will not be considered when we determine the sets \mathcal{C}_i 's (in Lemma 3.3.7). After z_2 is found, we proceed to search the third ANN z_3 similarly.

In general, suppose we have computed \mathcal{C}_i and z_i , and we are about to find z_{i+1} . We first determine \mathcal{C}_{i+1} by computing \mathcal{C}'_{i+1} and determining whether $C(z_i) \in \mathcal{C}_{i+1}$, where $C(z_i)$ is the cell of \mathcal{C}_i that contains z_i ; this can be done in $O((1 + m'_{i+1})(\log^2 n + \log m))$ time similarly as in Lemma 3.3.7.

Note that for any cell $C \in \mathcal{C}'_{i+1}$, it never appears in \mathcal{C}_j for any $1 \leq j \leq i$. Next, we determine the ANNs in the cells of \mathcal{C}'_{i+1} by Lemma 3.3.5 and insert them into the heap H . We also need to remove z_i from H . If $C(z_i) \notin \mathcal{C}_{i+1}$, then the point of H with smallest key is z_{i+1} . Otherwise, as before, we divide $C(z_i)$ into two sub-cells and compute their ANNs and insert them into H . Note that $C(z_i)$ may have already been divided into many sub-cells before. If so, they are explicitly maintained, and we can find the sub-cell that contains z_i in $O(\log k)$ time by binary search since $C(z_i)$ has at most $k - 1$ sub-cells ordered by y values. Then, we divide the sub-cell into two smaller sub-cells by the horizontal line through z_i and compute the ANNs in the two smaller sub-cells by Lemma 3.3.5 and insert them into H . Now, the point of H with smallest key is z_{i+1} .

To analyze the running time for computing z_{i+1} , \mathcal{C}_{i+1} can be determined in $O((1 + m'_{i+1})(\log^2 n + \log m))$ time. The time for computing the ANNs for the cells in \mathcal{C}'_{i+1} and possibly two sub-cells is bounded by $O((2 + m'_{i+1})\log^2 n)$. There are $O(2 + m'_{i+1})$ insertions and one “Extract-Min” operation on H , which together take $O((m'_{i+1} + 3)\log(|H|))$ time. Note that $|H| \leq m_i + m'_{i+1} + 2$.

We repeat the above procedure until z_k is found. We have the following lemma (a crucial observation is that $m_1 + \sum_{i=2}^k m'_i = O(m + k)$).

Lemma 3.3.8. *The overall running time of our query algorithm for finding $S_k(P^1) = \{z_1, z_2, \dots, z_k\}$ is $O(m \log m + (k + m) \log^2 n)$.*

Proof. Let $\lambda = m_1 + \sum_{i=2}^k m'_i$ denote the total number of cells in $\mathcal{C}_1 \cup \bigcup_{i=2}^k \mathcal{C}'_i$.

By Lemma 3.3.4, we compute \mathcal{C}_1 in $O(m \log n + m \log m)$ time. By Lemma 3.3.7, the total time for finding all cells of $\bigcup_{i=2}^k \mathcal{C}'_i$ is $O((k + \lambda)(\log^2 n + \log m))$.

In the entire algorithm, the total number of operations for finding the ANNs in the cells of \mathcal{A} (not including the sub-cells) is $O(\lambda)$ because the above cells are those in $\mathcal{C}_1 \cup \bigcup_{i=2}^k \mathcal{C}'_i$. After finding z_i for each $1 \leq i \leq k$, we have at most two more sub-cells,

and thus the total number of operations for finding the ANNs in the sub-cells is $O(k)$. Hence, by Lemma 3.3.5, the total time for finding the ANNs in the cells and sub-cells is $O((\lambda + k) \log^2 n)$. Also, we only need to explicitly maintain at most $O(k)$ sub-cells in the entire algorithm.

Similarly, the total number of operations on the heap H is $O(\lambda + k)$, and the size of H in the entire algorithm is always bounded by $O(\lambda + k)$. Hence, the total operations on H take $O((\lambda + k) \log(\lambda + k))$ time.

In summary, the overall running time is $O((k + \lambda)(\log^2 n + \log m + \log(\lambda + k)))$. To prove the lemma, we prove an important *claim*: $\lambda = O(m + k)$.

The proof for the claim is based on the fact that $|\mathcal{C}_i| = O(m)$ for each $1 \leq i \leq k$, since each skyline π_i intersects $O(m)$ cells. In particular, $|\mathcal{C}_k| = O(m)$. For each $1 \leq i \leq k - 1$, all the cells of \mathcal{C}_i except $C(z_i)$ are in \mathcal{C}_{i+1} and the cell $C(z_i)$ may or may not be in \mathcal{C}_{i+1} . Hence, $|\mathcal{C}_{i+1}| \geq |\mathcal{C}_i| - 1 + |\mathcal{C}'_{i+1}|$, i.e., $m_{i+1} \geq m_i - 1 + m'_{i+1}$. Therefore, $m_k \geq m_1 + \sum_{i=2}^k m'_i - (k - 1)$. Due to $m_k = O(m)$, we have $\lambda = m_1 + \sum_{i=2}^k m'_i \leq m_k + k - 1 = O(m + k)$. The above claim thus follows.

Due to the above claim, the overall running time for finding $S_k(P^1)$ is $O((k + m)(\log^2 n + \log(k + m)))$, which is $O(m \log m + (k + m) \log^2 n)$ (to see this, note that if $k > m$, then since $n \geq k$, $(k + m) \log(k + m) = O((k + m) \log^2 n)$ holds). \square

After obtaining $S_k(P^1)$, we also need to insert the points of $S_k(P^1)$ back to the data structure in Lemma 3.3.6 for answering other top- k ANN queries in future.

Putting everything together, we obtain the following.

Theorem 3.3.9. *Given a set P of n points in the plane, a data structure of size $O(n \log n \log \log n)$ can be built in $O(n \log n \log \log n)$ time, such that for any weighted point set Q and integer k , the top- k ANNs can be found in $O(m \log m + (k + m) \log^2 n)$ time. With trade-off between preprocessing and query time, we also build two other data structures: the first one has $O(n \log n)$ preprocessing time and space with $O(m \log m +$*

$(k + m) \log^2 n \log \log n$) query time; the second one has $O(n \log n \log^* n)$ preprocessing time and space with $O(m \log m + (k + m) \log^2 n \log^* n)$ query time.

Proof. Our preprocessing on P includes the following steps. (1) Sort all points in P by their x -coordinates and y -coordinates, respectively. (2) Build the dynamic segment-dragging query data structure in Lemma 3.3.6 on P . (3) Construct the data structure in Lemma 3.3.5. The total time and space are dominated by Step (3), regardless of which data structure of Lemma 3.3.5 is used.

Given any query set Q and any k , we compute $S_k(P)$ in the following steps. (1) Sort all points in Q by their x -coordinates and y -coordinates, respectively. (2) Process Q as in Lemma 3.3.2. (3) Compute a global minimum point q^* . (4) Divide the plane into four quadrants with respect to q^* . In each quadrant R , we find the top- k ANNs of Q in $P \cap R$ as follows. Suppose R is the first quadrant. (4.1) Find the set \mathcal{C}_1 by Lemma 3.3.4, and for each cell $C \in \mathcal{C}_1$, find the ANN of Q in $P \cap C$ by Lemma 3.3.5 and insert the point into a min-heap H ; the point of H with smallest aggregate distance to Q is the ANN of Q in $P \cap R$. (4.2) Based on \mathcal{C}_1 and z_1 , determine \mathcal{C}_2 and find z_2 . (4.3) The above procedure continues until we find z_k . (5) Among the found $4k$ points from all four quadrants of q^* (their aggregate distances to Q have also been computed), we report the k points with smallest aggregate distances to Q as $S_k(P)$. (6) Insert the above $4k$ points back to the data structure in Lemma 3.3.6 (for answering other top- k ANN queries in future).

For the running time of the query algorithm, the first three steps can be done in $O(m \log m)$ time; Step (4) can be done in $O(m \log m + (k + m) \log^2 n)$ time. Step (5) takes $O(k)$ time. Step (6) needs $O(k \log^2 n)$ time. Hence, the total query time is bounded by $O(m \log m + (k + m) \log^2 n)$. If we use the other two data structures in Lemma 3.3.5, then we have the query times of $O(m \log m + (k + m) \log^2 n \log \log n)$ and $O(m \log m + (k + m) \log^2 n \log^* n)$, respectively. \square

3.4. Top- k Aggregate Farthest Neighbor Searching

Our techniques can be extended to solve the top- k AFN searching, with the same bounds as in Theorems 3.2.2 and 3.3.9.

For the 1-D case, given any Q and k , recall that for computing the top- k ANNs, we first compute the global minimum point q^* and then search in P from q^* towards left and right. To compute the top- k AFNs, due to Lemma 3.2.1, we search in P from the leftmost and rightmost points of P towards the middle (e.g., either the leftmost or the rightmost point of P is the top-1 AFN of Q by Lemma 3.2.1). The rest of the algorithm is similar as that for Theorem 3.2.2 and the performance is the same as Theorem 3.2.2. We omit the details.

Theorem 3.4.1. *Given a set P of n points on the real line L , with $O(n \log n)$ preprocessing time and $O(n)$ space, the top- k AFNs can be found in $O(\min\{k, \log m\} \cdot m + k + \log n)$ time for any query set Q and any k ; if the points of Q are given sorted on L , then the query time is $O(k + m + \log n)$.*

For the 2-D case, consider any weighted point set Q and integer k . As in the ANN case, we first compute a global minimum point q^* and then compute the top- k AFNs in each quadrant of q^* . Suppose R is the first quadrant with respect to q^* . We find the top- k AFNs in R as follows. Recall that in the ANN case we search the top- k ANNs in a direction from q^* towards northeast. In the AFN case, due to Lemma 3.3.1, we search the top- k AFNs along the opposite direction, i.e., from northeast towards q^* . Specifically, here we re-define the “dominate” relationship in the opposite way as before: a point p_1 *dominates* p_2 if and only if $x(p_1) \geq x(p_2)$ and $y(p_1) \geq y(p_2)$. A point p in $P \cap R$ is called a *maximal point* if no other point in $P \cap R$ dominates p . Similarly, we re-define the *skyline* as the path connecting all maximal points of $P \cap R$. According to Lemma 3.3.1, the AFN of Q in $P \cap R$ must be in the skyline. Then, we can use a similar algorithm as in the ANN case to compute all top- k AFNs. More

specifically, we first compute the AFN p on the skyline and then search the second AFN on the next skyline (without considering p); we continue this procedure until we find the k -th AFN. The algorithm is similar (or symmetric) as the ANN case and we omit the details. The performance is the same as that in Theorem 3.3.9.

Theorem 3.4.2. *Given a set P of n points in the plane, a data structure of $O(n \log n \log \log n)$ size can be built in $O(n \log n \log \log n)$ time, such that for any query set Q and any k , the top- k AFNs can be found in $O(m \log m + (k + m) \log^2 n)$ time. With trade-off between preprocessing and query time, we also build two other data structures: the first one has $O(n \log n)$ preprocessing time and space with $O(m \log m + (k + m) \log^2 n \log \log n)$ query time; the second one has $O(n \log n \log^* n)$ preprocessing time and space with $O(m \log m + (k + m) \log^2 n \log^* n)$ query time.*

3.5. Conclusion

We presented efficient methods for the top- k aggregate nearest and farthest neighbor searching in the plane under the L_1 metric. Our results are the first-known solutions for the general top- k queries on the weighted query points. Even for the special case where $k = 1$ or the unweighted query points, our results are still generally better than the previous work. An interesting open problem is whether and how the techniques proposed in this chapter can be extended to higher dimensional spaces.

Probabilistic Nearest Neighbor

4.1. Introduction

In this chapter, we consider the probabilistic nearest neighbor (PNN) problem, a counterpart of the ENN problem in Chapter 2. Despite many efforts devoted to the PNN problem, it still lacks a theoretical foundation. Specifically, not only are we yet to understand its complexity (is the problem inherently more difficult than on precise data?), but we also lack efficient algorithms to solve it. Furthermore, existing solutions all use heuristics without nontrivial performance guarantees. This chapter addresses some of these issues.

4.1.1. Problem definition

We assume that each uncertain point P follows from the locational uncertainty model defined in Section 1.2. When P is represented by a continuous probability distribution, the *uncertainty region* of P (or the *support* of f_P) is the set of points for which f_P is positive, i.e., $\text{Sup } f_P = \{x \in \mathbb{R}^d \mid f_P(x) > 0\}$. We assume P has a bounded uncertainty region: if f_P is Gaussian, we work on truncated Gaussian, as in [28, 44]. Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n uncertain points in \mathbb{R}^2 , and let $d(\cdot, \cdot)$

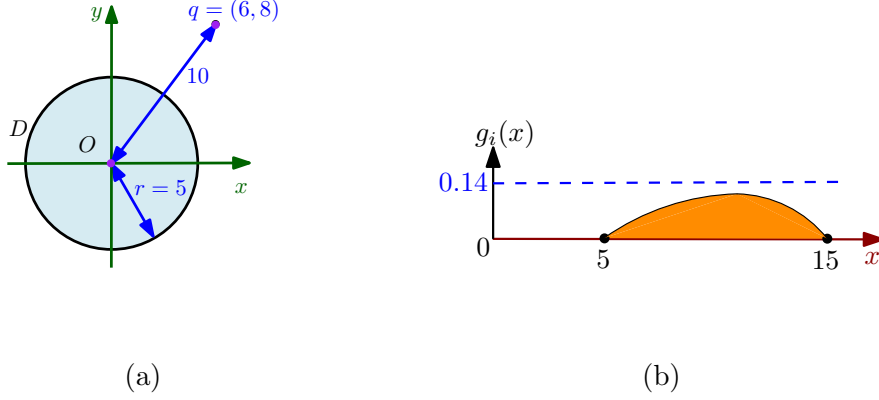


FIGURE 4.1. (a) P_i is represented by a uniform distribution defined on a disk D of radius $r = 5$ and centered at origin O , $q = (6, 8)$, and $d(\cdot, \cdot)$ is L_2 metric; (b) $g_{q,i}(x)$.

be the Euclidean distance. For a point $q \in \mathbb{R}^2$, let $\pi_i(q) = \pi(P_i, q)$ be the probability of $P_i \in \mathcal{P}$ being the nearest neighbor of q , referred to as its *qualification probability*, defined as follows:

For a point q , and $i = 1, \dots, n$, let $g_{q,i}$ be the pdf of the distance between q and P_i . That is, $g_{q,i}(x) = \Pr[x \leq d(q, P_i) \leq x + dx]$. See Fig. 4.1 for an example of $g_{q,i}$. Let $G_{q,i}(x) = \int_0^x g_{q,i}(y)dy$ denote the cumulative distribution function (cdf) of the distance between q and P_i . Then $\pi_i(q)$, the probability that P_i is the NN of q , is

$$\pi_i(q) = \int_0^\infty g_{q,i}(r) \prod_{j \neq i} (1 - G_{q,j}(r)) dr. \quad (4.1)$$

If P_i 's are represented by discrete distributions, then $\pi_i(q)$ can be rewritten as follows. Let $G_{q,j}(r) = \sum_{d(p_{jt}, q) < r} w_{jt}$, then

$$\pi_i(q) = \sum_{p_{is} \in P_i} w_{is} \prod_{j \neq i} (1 - G_{q,j}(d(p_{is}, q))). \quad (4.2)$$

Given a set \mathcal{P} of n uncertain points, the *probabilistic nearest neighbor* (PNN) problem is to preprocess \mathcal{P} into a data structure so that, for any given query point q , we can efficiently return all pairs $(P_i, \pi_i(q))$ with $\pi_i(q) > 0$.

In addition, one can consider the *most likely* NN of q , denoted NN_L , which is the P_i with the maximum $\pi_i(q)$; or the *threshold* NN, i.e., the set of all the P_i 's with $\pi_i(q)$

exceeding a given threshold τ .

Usually, the PNN problem is divided into the following two subproblems, which are often considered separately.

Nonzero NNs. The first subproblem is to find all the P_i 's with $\pi_i(q) > 0$ without computing the actual qualification probabilities, i.e., to find

$$\text{NN}_{\neq 0}(q, \mathcal{P}) = \{P_i \mid \pi_i(q) > 0\}.$$

If the point set \mathcal{P} is obvious from the context, we drop the argument \mathcal{P} from $\text{NN}_{\neq 0}(q, \mathcal{P})$, and write it as $\text{NN}_{\neq 0}(q)$. Note that $\text{NN}_{\neq 0}(q)$ depends (besides q) only on the uncertainty regions of the uncertain points, but not on the actual **pdf**'s.

A possible approach to compute nearest neighbors is to use Voronoi diagrams. For example, the standard Voronoi diagram of a set of points in \mathbb{R}^2 (without uncertainty) is the planar subdivision so that all points in the same face have the same nearest neighbor. In our case, we define the *nonzero Voronoi diagram*, denoted by $\mathcal{V}_{\neq 0}(\mathcal{P})$, to be the subdivision of \mathbb{R}^2 into maximal connected regions such that $\text{NN}_{\neq 0}(q)$ is the same for all points q within each region. That is, for a subset $\mathcal{J} \subseteq \mathcal{P}$, let

$$\text{cell}_{\neq 0}(\mathcal{J}) = \{q \in \mathbb{R}^2 \mid \text{NN}_{\neq 0}(q) = \mathcal{J}\}. \quad (4.3)$$

Although there are 2^n subsets of \mathcal{P} , we will see below that only a small number of them have nonempty Voronoi cells. The planar subdivision $\mathcal{V}_{\neq 0}(\mathcal{P})$ is induced by all the nonempty $\text{cell}_{\neq 0}(\mathcal{J})$'s for $\mathcal{J} \subseteq \mathcal{P}$. The (combinatorial) complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ is the total number of vertices, edges, and faces in $\mathcal{V}_{\neq 0}(\mathcal{P})$. In this chapter, we study the worst-case complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ and how it can be efficiently constructed. The complexity of the Voronoi diagram is often regarded as a measure of the complexity of the corresponding nearest-neighbor problem. In addition, once we have $\mathcal{V}_{\neq 0}(\mathcal{P})$, a point-location structure can be built on top of it to support $\text{NN}_{\neq 0}$ queries in logarithmic time.

Similarly, one can consider the *most likely Voronoi diagram* (partitioning the plane into regions having the same most likely NN) and the *threshold Voronoi diagram* (partitioning the plane into regions having the same set of points with qualification probabilities exceeding τ). However, these Voronoi diagrams tend to be more complex as they depend on the actual distributions of the uncertain points.

Computing qualification probabilities The second subproblem is to compute the qualification probability $\pi_i(q)$ for a given q and P_i . Since exact values of these probabilities are often unstable — a far away point can affect these probabilities — and computing them requires complex n -dimensional integration, which is often expensive. As such, we resort to computing $\pi_i(q)$ approximately within a given additive error tolerance $\varepsilon \in (0, 1)$. More precisely, we aim at returning a value $\hat{\pi}_i(q)$ such that $|\pi_i(q) - \hat{\pi}_i(q)| \leq \varepsilon$.

Note that, having solved these two subproblems, we obtain immediate approximate solutions to the most likely NN and the threshold NN problems.

4.1.2. Previous work

Besides the related work on ENN stated in Section 2.1, here we survey some more that are related to PNN.

Nonzero NN s. Sember and Evans [111] showed that the complexity of the nonzero Voronoi diagram (though they did not use this term explicitly) when the uncertainty regions of the points are disks is $O(n^4)$; they did not offer any lower bound. If one only considers those cells of $\mathcal{V}_{\neq 0}(P)$ in which $\text{NN}_{\neq 0}(q)$ contains only one uncertain point P_i , i.e., only P_i has a non-zero probability of being the NN of q , they showed that the complexity of these cells is $O(n)$. Note that for such a cell, we always have $\pi_i(q) = 1$ for any q in the cell, so they are called the *guaranteed Voronoi diagram*. Probably unaware of the work by Sember and Evans [111], Cheng *et al.* [47] proved an

exponential upper bound for the complexity of the nonzero Voronoi diagram, which they referred to as UV-diagram.

The nonzero Voronoi diagram is not the only way to find the nonzero NNs. Cheng *et al.* [46] designed a branch-and-prune solution based on the R -tree. Recently, Zhang *et al.* [128] proposed to combine the nonzero Voronoi diagram with R -tree-like bounding rectangles. These methods do not have any performance guarantees.

Computing qualification probabilities. Computing the qualification probabilities has attracted a lot of attention in the database community. Cheng *et al.* [46] used numerical integration, which is quite expensive. Cheng *et al.* [44] and Bernecker *et al.* [27] proposed some filter-refinement methods to give upper and lower bounds on the qualification probabilities. Kriegel *et al.* [85] took a random sample from the continuous distribution of each uncertain point to convert it to a discrete one, so that the integration becomes a sum, and they clustered each sample to further reduce the complexity of the query computation. These methods are best-effort based: they do not always give the ε -error that we aim at — how tight the bounds are depends on the data.

Other variants of the problem. The PNN problem we focus on in this chapter is the most commonly studied version of the problem, but many variants and extensions have been considered.

The probabilistic model we use is often called the *locational model*, where the location of an uncertain point follows the given distribution. This is to be compared with the *existential model*, where each point has a precise location but it appears with a given probability.

Besides using the qualification probability, one can also consider the expected distance from a query point q to an uncertain point, and return the one minimizing the expected distance as the nearest neighbor; this was studied by Agarwal *et al.*

[9](Chapter 2). This NN definition is much easier since the expected distance to each uncertain point can be computed separately, whereas the qualification probability involves the interaction among all uncertain points. However, the expected nearest neighbor is not a good indicator under large uncertainty (see [127] for details).

Finally, instead of returning only the nearest neighbor, one can ask to return the k nearest neighbors in a ranked order (the k NN problem). If we use expected distance, the ranking of points is straightforward, namely, rank them in a non-decreasing order of the expected distance from the query point. However, when qualification probabilities are considered, many different criteria for ranking the results are possible, leading to different problem variants [75].

Various combinations of these extensions have been studied in the literature; see, e.g., [28, 45, 80, 91, 127].

4.1.3. Our results

The main result of this chapter is a tight bound on the combinatorial complexity of nonzero Voronoi diagrams. We also present efficient algorithms for answering $\text{NN}_{\neq 0}$ queries as well as for computing the qualification probabilities.

Nonzero NNs. We first study (in Section 4.2) the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$. Suppose the uncertainty region of each $P_i \in \mathcal{P}$ is a disk and $d(\cdot, \cdot)$ is the L_2 metric. We show that $\mathcal{V}_{\neq 0}(\mathcal{P})$ has $O(n^3)$ complexity, and that this bound is tight in the worst case. This significantly improves the bound in [111] and closes the problem. If the disks are pairwise disjoint and the ratio of their radii is at most λ , then the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ is $O(\lambda n^2)$, and we prove a lower bound of $\Omega(n^2)$. If each point in \mathcal{P} has a discrete distribution of description complexity k , then we show that $\mathcal{V}_{\neq 0}(\mathcal{P})$ has $O(kn^3)$ complexity.

We present a randomized algorithm for computing $\mathcal{V}_{\neq 0}(\mathcal{P})$ in $O(n^2 \log n + \mu)$ expected time, where μ is the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$. A point-location structure can

be built on top of $\mathcal{V}_{\neq 0}(\mathcal{P})$ whose size is proportional to the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ and that answers an $\text{NN}_{\neq 0}$ query in $O(\log n + t)$ time, where t is the output size.

Next, we consider (in Section 4.3) how quickly $\text{NN}_{\neq 0}$ queries can be answered using less space. If the uncertainty region of each point is a disk, then an $\text{NN}_{\neq 0}$ query can be answered in $O(\log n + t)$ time using $O(n^{1+\varepsilon})$ space, for any constant $\varepsilon > 0$, where t is the output size. If each uncertain point has at most k possible locations, then an $\text{NN}_{\neq 0}$ query can be answered in $O(\log(nk) + t)$ time using $O((nk)^{2+\varepsilon})$ (for any $\varepsilon > 0$) space, or in $O((nk)^{1/2+\varepsilon} + t)$ time using $O(nk)$ space, where t is the output size.

Computing qualification probabilities. We present two algorithms for approximating the qualification probabilities efficiently. The first (see Section 4.4.1) is a Monte-Carlo algorithm for estimating $\pi_i(q)$ for any P_i and q within additive error ε with probability at least $1 - \delta$, for parameters $\varepsilon, \delta \in (0, 1)$. First we argue that if each uncertain point has a discrete distribution of description complexity k , then we can estimate $\pi_i(q)$ within additive error ε with probability at least $1 - \delta$ by using $s_{\varepsilon, \delta} = O((1/\varepsilon^2) \log(nk/\delta))$ random instantiations of \mathcal{P} . (Note that there are at most $1/\varepsilon$ P_i 's for which $\pi_i(q) > \varepsilon$.) Consequently, we can preprocess \mathcal{P} into a data structure of size $O((n/\varepsilon^2) \log(nk/\delta))$ so that for any query point $q \in \mathbb{R}^2$, $\pi_i(q)$ for all P_i 's can be estimated within additive error ε in $O((1/\varepsilon^2) \log(nk/\delta) \log n)$ time, with probability at least $1 - \delta$. The algorithm explicitly computes the estimates of $\pi_i(q)$'s for at most $s_{\varepsilon, \delta}$ points and sets the estimate to 0 for the rest of the points. This data structure can be used to find the (approximate) most likely NN and the threshold NN within the same time bound. We also show that this approach works even if the distribution of each P_i is continuous, by approximating a continuous distribution with a discrete one. A key observation is that it suffices to sample polynomial number of points from the distribution of each P_i to ensure that the error in the quantification probability

is at most ε .

Next, we describe (in Section 4.4.2) a deterministic algorithm for computing $\pi_i(q)$ approximately provided that the distribution of each P_i is discrete. Let $P_i = \{p_{i1}, \dots, p_{ik}\}$ and $w_{ij} = \Pr[P_i \text{ is } p_{ij}]$. We set $\rho = \frac{\max_{i,j} w_{ij}}{\min_{i,j} w_{ij}}$, where maximum and minimum are taken over all the location probabilities of points in $S = \bigcup_{i=1}^n P_i$. We show that \mathcal{P} can be preprocessed into a data structure of $O(n)$ size so that for any $q \in \mathbb{R}^2$ and for any $\varepsilon \in (0, 1)$, $\pi_i(q)$, for all $i \in \{1, \dots, n\}$, can be computed with additive error at most ε in $O(\rho k \log(\rho/\varepsilon) + \log n)$ time. Our result shows that there are at most $m(\rho, \varepsilon) = \rho k \ln(\rho/\varepsilon) + k - 1$ points of \mathcal{P} for which $\pi_i(q) > \varepsilon$. The algorithm explicitly estimates $\pi_i(q)$ for at most $m(\rho, \varepsilon)$ points and sets the estimate to 0 for the rest of the points. As earlier, this data structure can be used to solve the most likely NN and the threshold NN problem approximately within the same time bound.

Finally, we present experimental results, in Section 4.5, to demonstrate the efficacy of our approach for estimating quantification probabilities.

4.2. Nonzero Probabilistic Voronoi Diagram

Let \mathcal{P} be a set of n uncertain points as described earlier. We analyze the combinatorial structure of $\mathcal{V}_{\neq 0}(\mathcal{P})$ and describe algorithms for constructing it. We first consider the case when the distribution of each point is continuous and then consider the discrete case.

4.2.1. Continuous case

For simplicity, we assume that the uncertainty region of each P_i is a circular disk D_i of radius r_i centered at c_i .

We first observe that the actual pdf of P_i is not important for computing $\mathcal{V}_{\neq 0}(\mathcal{P})$. What really matters is the uncertainty region D_i . More precisely, for each $1 \leq i \leq n$

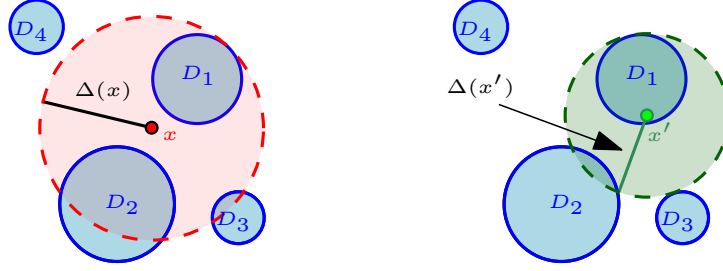


FIGURE 4.2. $\mathcal{P} = \{P_1, \dots, P_5\}$. $\Delta(x) = \Delta_1(x)$, $\text{NN}_{\neq 0}(x, \mathcal{P}) = \{P_1, P_2, P_3\}$, $\Delta(x') = \Delta_1(x')$, $\text{NN}_{\neq 0}(x', \mathcal{P}) = \{P_1, P_2\}$, and x' lies on an edge of $\mathcal{V}_{\neq 0}(\mathcal{P})$.

and for $q \in \mathbb{R}^2$, let

$$\Delta_i(q) = \max_{p \in D_i} d(q, p) = d(q, c_i) + r_i,$$

$$\delta_i(q) = \min_{p \in D_i} d(q, p) = \max\{d(q, c_i) - r_i, 0\}$$

be the maximum and minimum possible distance, respectively, from q to P_i .

The proof of the following lemma is straightforward.

Lemma 4.2.1. *For a point $x \in \mathbb{R}^2$, a point $P_i \in \mathcal{P}$ belongs to $\text{NN}_{\neq 0}(x, \mathcal{P})$ if and only if*

$$\delta_i(x) < \Delta_j(x) \text{ for all } 1 \leq j \neq i \leq n.$$

Let $\Delta: \mathbb{R}^2 \rightarrow \mathbb{R}$ denote the *lower envelope*¹ of $\Delta_1, \dots, \Delta_n$; that is, for any $q \in \mathbb{R}^2$,

$$\Delta(q) = \min_{1 \leq i \leq n} \Delta_i(q).$$

The projection of the graph of $\Delta(x)$ onto the xy -plane is the additive-weighted Voronoi diagram of the points c_1, \dots, c_n , where the weight of c_i is r_i , and the weighted distance from q to c_i is $d(q, c_i) + r_i$, for $i = 1, \dots, n$. Let \mathbb{M} denote this planar subdivision. It has linear complexity and each of its edges is a hyperbolic arc; see [21].

Lemma 4.2.1 implies that, for any $q \in \mathbb{R}^2$,

$$\text{NN}_{\neq 0}(q, \mathcal{P}) = \{P_i \mid \delta_i(q) < \Delta(q)\}. \quad (4.4)$$

¹The *lower envelope*, L_F , of a set F of functions is their pointwise minimum, i.e., $L_F(x) = \min_{f \in F} f(x)$. The *upper envelope*, U_F , of F is the pointwise maximum, i.e., $U_F(x) = \max_{f \in F} f(x)$.

See Fig. 4.2. It also implies that, as we move x continuously in \mathbb{R}^2 , $\text{NN}_{\neq 0}(x, \mathcal{P})$ remains the same until $\delta_i(x)$, for some $1 \leq i \leq n$, becomes equal to $\Delta(x)$ (e.g., x' in Fig. 4.2). The above was also observed previously; see, e.g. [44, 46]. Using this observation we can now characterize $\mathcal{V}_{\neq 0}(\mathcal{P})$.

For $i = 1, \dots, n$, let $\gamma_i = \{x \in \mathbb{R}^2 \mid \delta_i(x) = \Delta(x)\}$ be the zero set of the function $\Delta(x) - \delta_i(x)$. Set $\Gamma = \{\gamma_1, \dots, \gamma_n\}$.

The curve γ_i partitions the plane into two open regions: $\Delta(x) < \delta_i(x)$ and $\Delta(x) > \delta_i(x)$. By Eq. (4.4), $P_i \in \text{NN}_{\neq 0}(x, \mathcal{P})$ for all points x inside the latter region and for none of the points x inside the former region. It is well known that for any fixed $j \neq i$, $\gamma_{ij} = \{x \in \mathbb{R}^2 \mid \delta_i(x) = \Delta_j(x)\}$ is a hyperbolic curve [21]. The curve γ_i is composed of pieces of γ_{ij} , for $j \neq i$. We refer to the endpoints of these pieces as *breakpoints* of γ_i . They are the intersection points of γ_i with an edge of \mathbb{M} and correspond to points q such that the disk of radius $\Delta(q)$ centered at q touches (at least) two disks of \mathcal{D} from inside, touches D_i from outside, and does not contain any disk of \mathcal{D} in its interior. See Fig. 4.3. Formally, we say that a disk D_1 touches a disk D_2 from the *outside* (resp. *inside*) if $\partial D_1 \cap \partial D_2 \neq \emptyset$ and $\text{int } D_1 \cap \text{int } D_2 = \emptyset$ (resp. $\text{int } D_2 \subseteq \text{int } D_1$).

Lemma 4.2.2. *The curve γ_i , $1 \leq i \leq n$, has at most $2n$ breakpoints, and it can be computed in $O(n \log n)$ time.*

Proof. Let $\Gamma_i = \{\gamma_{ij} \mid j \neq i, 1 \leq j \leq n\}$. It can be verified that a ray emanating from c_i intersects the hyperbolic curve γ_{ij} , for any $j \neq i$, in at most one point, so γ_{ij} can be viewed as the graph of a function in polar coordinates with c_i as the origin. That is, let $\gamma_{ij}: [0, 2\pi) \rightarrow \mathbb{R}_{\geq 0}$, where $\gamma_{ij}(\theta)$ is the distance from c_i to γ_{ij} in direction θ . Then γ_i is the lower envelope of Γ_i . Since each pair of curves in Γ_i intersects at most twice, a well-known result on lower envelopes implies that γ_i has at most $2n$ breakpoints, and that it can be computed in $O(n \log n)$ time [113]. See Fig. 4.4 for

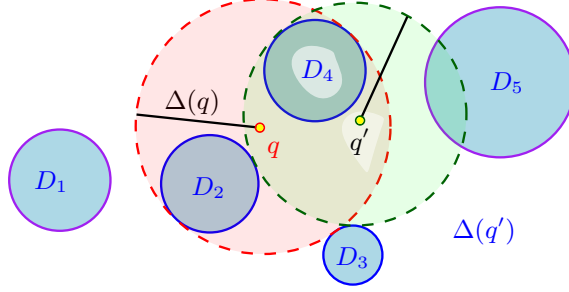


FIGURE 4.3. The point q is a breakpoint of γ_3 and q' is an intersection point of γ_2 and γ_3 .

an example. □

Let $\mathcal{A}(\Gamma)$ denote the planar subdivision induced by Γ : its vertices are the breakpoints of γ_i 's and the intersection points of two curves in Γ , its edges are the portions of γ_i 's between two consecutive vertices, and its cells are the maximal connected regions of Γ that do not intersect any curve of Γ . We refer to vertices, edges, and cells of $\mathcal{A}(\Gamma)$ as its 0-, 1-, and 2-dimensional *faces*.

For a face ϕ (of any dimension), and for any two points $x, y \in \phi$, the sets $\{P_i \mid \delta_i(x) < \Delta(x)\}$ and $\{P_j \mid \delta_j(y) < \Delta(y)\}$ are the same; we denote this set by \mathcal{P}_ϕ . Furthermore, if x, y lie in two neighboring faces ϕ and ϕ' , respectively, then $\mathcal{P}_\phi \neq \mathcal{P}_{\phi'}$. The following lemma is an immediate consequence of Eq. (4.4).

Lemma 4.2.3. *Let $x \in \mathbb{R}^2$ be a point lying in a face ϕ of $\mathcal{A}(\Gamma)$. Then $\text{NN}_{\neq 0}(x, \mathcal{P}) = \mathcal{P}_\phi$.*

For a subset $\mathcal{T} \subseteq \mathcal{P}$, let $\text{cell}_{\neq 0}(\mathcal{T})$ be as defined in Eq. (4.3). An immediate corollary of the above lemma is:

Corollary 4.2.4. *(i) For any $\mathcal{T} \subseteq \mathcal{P}$, $\text{cell}_{\neq 0}(\mathcal{T}) \neq \emptyset$ if and only if there is a face ϕ of $\mathcal{A}(\Gamma)$ with $\mathcal{T} = \mathcal{P}_\phi$.*

(ii) The planar subdivision $\mathcal{A}(\Gamma)$ coincides with $\mathcal{V}_{\neq 0}(\mathcal{P})$.

We now bound the complexity of $\mathcal{A}(\Gamma)$ and thus of $\mathcal{V}_{\neq 0}(\mathcal{P})$.

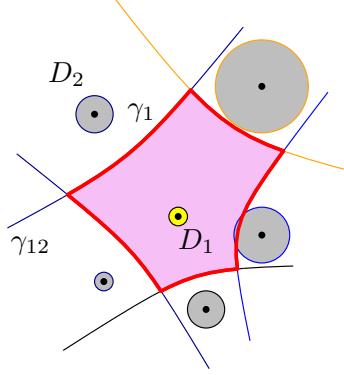


FIGURE 4.4. An example of γ_1 .

Theorem 4.2.5. *Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n uncertain points in \mathbb{R}^2 whose uncertainty regions are disks. Then $\mathcal{V}_{\neq 0}(\mathcal{P})$ has $O(n^3)$ complexity. Moreover, it can be computed in $O(n^2 \log n + \mu)$ expected time, where μ is the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$.*

Proof. Using a standard perturbation argument (see e.g. [113]), it suffices to bound the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ when the points of \mathcal{P} are in general position, so we can assume that the degree of every vertex in $\mathcal{V}_{\neq 0}(\mathcal{P})$ is constant. Since $\mathcal{V}_{\neq 0}(\mathcal{P})$ is a planar subdivision and the degree of every vertex is constant, the number of edges and cells in $\mathcal{V}_{\neq 0}(\mathcal{P})$ is proportional to the number of its vertices. Hence, it suffices to bound the number of vertices. Let $\Gamma = \{\gamma_1, \dots, \gamma_n\}$ be the set of curves as defined above. By Lemma 4.2.2, each γ_i has $O(n)$ breakpoints, so there are a total of $O(n^2)$ breakpoints. We claim that each pair of curves γ_i and γ_j intersect $O(n)$ times — each such intersection point corresponds to a point $v \in \mathbb{R}^2$ such that the disk of radius $\Delta(v)$ centered at v touches D_i and D_j from the outside and another disk D_k of \mathcal{D} , the one realizing the value of $\Delta(v)$, from the inside (e.g., q' in Fig. 4.3). For a fixed k , we show that there are at most two points v such that $\delta_i(v) = \delta_j(v) = \Delta_k(v)$. Note that $\delta_i(v) = \Delta_k(v)$ represents either an empty set or one hyperbolic branch. Similarly for $\delta_j(v) = \Delta_k(v)$. Two such hyperbolic branches intersect at most twice, implying that $\delta_i(v) = \delta_j(v) = \Delta_k(v)$ contributes at most two vertices. Hence, the number of vertices in $\mathcal{V}_{\neq 0}(\mathcal{P})$ is $O(n^3)$, as claimed.

By Lemma 4.2.2, Γ can be computed in $O(n^2 \log n)$ time. The planar subdivision $\mathcal{A}(\Gamma)$ can be computed in $O(n \log n + \mu)$ expected time using randomized incremental method [15], where μ is the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$. Hence $\mathcal{V}_{\neq 0}(\mathcal{P})$ can be computed in $O(n^2 \log n + \mu)$ expected time. \square

Remarks. A near-cubic bound on the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ holds if the uncertainty region of each point is a *semi-algebraic set of constant description complexity*, i.e., each region is defined by Boolean operations (union, intersection, and complementation) of a constant number of bivariate polynomial inequalities of constant maximum degree each.

Next we show that the above upper bound is tight in the worst case.

Theorem 4.2.6. *There exists a set \mathcal{P} of n uncertain points whose uncertainty regions are disks such that $\mathcal{V}_{\neq 0}(\mathcal{P})$ has $\Omega(n^3)$ vertices.*

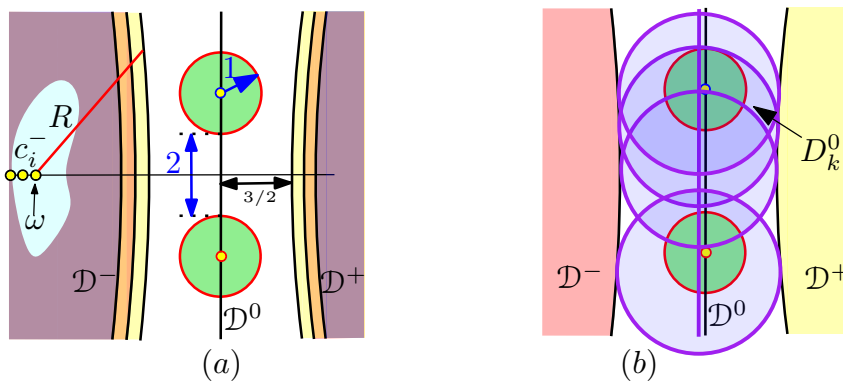


FIGURE 4.5. (a) $\Omega(n^3)$ lower bound construction with $m = 3$; only some disks are drawn. (b) An illustration of the proof.

Proof. Assume that $n = 4m$ for some $m \in \mathbb{N}^+$. We choose two parameters $R = 8n^2$ and $\omega = 1/n^2$. We construct three families of disks: $\mathcal{D}^- = \{D_1^-, \dots, D_m^-\}$, $\mathcal{D}^+ = \{D_1^+, \dots, D_m^+\}$, and $\mathcal{D}^0 = \{D_1^0, \dots, D_{2m}^0\}$. The radius of all disks in $\mathcal{D}^- \cup \mathcal{D}^+$ is R and their centers lie on the x -axis; the radius of all disks in \mathcal{D}^0 is 1 and their

centers lie on the y -axis. More precisely, for $1 \leq i, j \leq m$, the center of D_i^- is $c_i^- = (-R - 3/2 - (i - 1)\omega, 0)$ and the center of D_j^+ is $c_j^+ = (R + 3/2 + (j - 1)\omega, 0)$, and for $1 \leq k \leq 2m$, the center of D_k^0 is $(0, 4(k - m) - 2)$. See Fig. 4.5(a).

We claim that for every triple i, j, k with $1 \leq i, j \leq m$ and $1 \leq k \leq 2m$, there are two disks each of which touches D_i^- and D_j^+ from the outside and D_k^0 from the inside and does not contain any disk of $\mathcal{D}^- \cup \mathcal{D}^+ \cup \mathcal{D}^0$ in its interior. See Fig. 4.5(b).

Fix such a triple i, j, k . Since the radius of D_i^- and D_j^+ is the same, the locus b_{ij} of the centers of disks that simultaneously touch D_i^- and D_j^+ from the outside is the bisector of their centers, i.e., b_{ij} is the vertical line $x = (x(c_i^-) + x(c_j^+))/2 = (j - i)\omega/2$. Let σ_{ij} denote the intersection point of b_{ij} and the x -axis; $\sigma_{ij} = (\frac{1}{2}(j - i)\omega, 0)$. A point on b_{ij} can be represented by its y -coordinate; we will not distinguish between the two. For y -value a , let ξ_a be the disk centered at a and simultaneously touching D_i^- and D_j^+ . The radius of ξ_a is

$$\begin{aligned} \|a - c_i^-\| - R &= \sqrt{a^2 + \|c_i^- - \sigma_{ij}\|^2} - R \\ &= \sqrt{a^2 + \left(R + 3/2 + \left(\frac{i + j}{2} - 1\right)\omega\right)^2} - R. \end{aligned}$$

The radius of ξ_a is thus at least $3/2$, and for $a \in [-4m, 4m]$, it is at most 2 (using the fact that $R \geq 8n^2$ and $\omega = 1/n^2$). Hence for $a \in [-4m, 4m]$, ξ_a contains at most one disk of \mathcal{D}^0 in its interior, and obviously ξ_a does not contain any disk of $\mathcal{D}^- \cup \mathcal{D}^+$ in its interior.

Let $a_k = 4(k - m) - 2$. Then the disk ξ_{a_k} contains D_k^0 in its interior because the distance between the centers of D_k^0 and ξ_{a_k} is at most $m\omega \leq 1/(4n)$, the radius of D_k^0 is 1 , and the radius of ξ_{a_k} is at least $3/2$. On the other hand, the disk ξ_a for $a = a_k \pm 2$ does not contain D_k^0 in its interior because the radius of ξ_a is at most 2 and the distance between the center of D_k^0 and ξ_a is at least 2 . Therefore, by a continuity argument, there is a value $a^+ \in [a_k, a_k + 2]$ at which ξ_{a^+} touches D_k^0 from the inside.

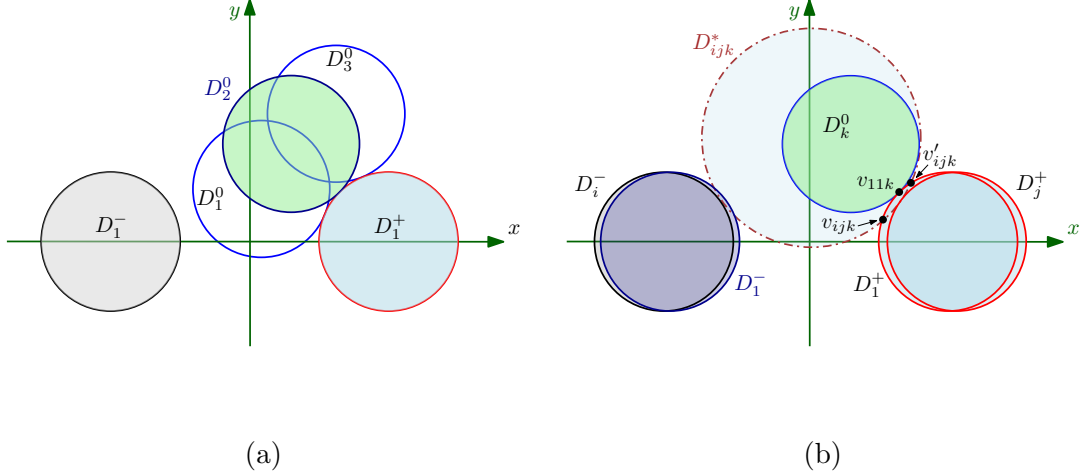


FIGURE 4.6. (a) $\Omega(n^3)$ lower bound construction using disks of same radius with $m = 3$; only some disks are drawn. (b) Illustration of the proof.

Similarly, there is a value $a^- \in [a_k - 2, a_k]$ at which ξ_{a^-} touches D_k^0 from the inside.

This proves the claim that there are two disks touching D_i^- and D_j^+ from the outside and D_k^0 from the inside and not containing any disk of $\mathcal{D}^- \cup \mathcal{D}^+ \cup \mathcal{D}^0$ in its interior. In other words, each triple i, j, k contributes two vertices to $\mathcal{V}_{\neq 0}(\mathcal{P})$. Hence $\mathcal{V}_{\neq 0}(\mathcal{P})$ has $\Omega(n^3)$ vertices. \square

Next, we show that a more careful construction gives an $\Omega(n^3)$ lower bound on the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ even if all disks in \mathcal{D} have the same radius.

Theorem 4.2.7. *There exists a set \mathcal{P} of n uncertain points, whose uncertainty regions are disks of the same radius, for which $\mathcal{V}_{\neq 0}(\mathcal{P})$ has $\Omega(n^3)$ vertices.*

Proof. Assume that $n = 3m$ for some $m \in \mathbb{N}^+$. We choose two parameters $\theta = \frac{\pi}{2} \cdot \frac{1}{(m+1)}$, and a sufficiently small positive number ω . We construct three families of disks: $\mathcal{D}^- = \{D_1^-, \dots, D_m^-\}$, $\mathcal{D}^+ = \{D_1^+, \dots, D_m^+\}$, and $\mathcal{D}^0 = \{D_1^0, \dots, D_m^0\}$. Without loss of generality, we assume that the radius of all disks is 1. The centers of disks in $\mathcal{D}^- \cup \mathcal{D}^+$ lie on the x -axis, and the centers of disks in \mathcal{D}^0 lie in the first quadrant. More precisely, for $1 \leq i, j \leq m$, the center of D_i^- is $c_i^- = (-2 - (i - 1)\omega, 0)$ and

the center of D_j^+ is $c_j^+ = (2 + (j - 1)\omega, 0)$, and for $1 \leq k \leq m$, the center of D_k^0 is $(2 - 2\cos(k\theta), 2\sin(k\theta))$. See Fig. 4.6(a).

We claim that for every triple i, j, k with $1 \leq i, j, k \leq m$, there is a disk touching D_i^- and D_j^+ from the outside and D_k^0 from the inside and not containing any disk of $\mathcal{D}^- \cup \mathcal{D}^+ \cup \mathcal{D}^0$ in its interior.

First of all, this is true for $i = j = 1$ and $1 \leq k \leq m$. Note that D_1^- is centered at $(-2, 0)$, D_1^+ is centered at $(2, 0)$, and D_k^0 touches D_1^+ from the outside. Since the radius of D_1^- and D_1^+ is the same, the locus b_{11} of the centers of disks that simultaneously touch D_1^- and D_1^+ from the outside is the bisector of their centers, i.e., b_{11} is y -axis. Fix a value k . It is easy to see that the disk D_{11k}^* centered at $(0, 2\tan(k\theta))$ with the radius $\frac{2}{\cos(k\theta)} - 1$ touches D_1^- and D_1^+ from the outside and D_k^0 from the inside. Furthermore, we show that D_{11k}^* does not contain disks in \mathcal{D}^0 in its interior (obvious for $\mathcal{D}^- \cup \mathcal{D}^+$). Since every disk in \mathcal{D}^0 touches D_1^+ from the outside, D_{11k}^* containing a disk of \mathcal{D}^0 in its interior would imply that D_{11k}^* intersects the interior of D_1^+ , a contradiction.

Next, we show that it holds for $1 < i, j \leq m$ and $1 \leq k \leq m$. The key idea is that D_i^- (resp. D_j^+) got placed by translating (“perturbing”) D_1^- (resp. D_1^+) so little that the disk D_{ijk}^* touching D_i^- and D_j^+ from the outside and D_k^0 from the inside does not contain any disk of $\mathcal{D}^- \cup \mathcal{D}^+ \cup \mathcal{D}^0$ in its interior as for the case when $i = j = 1$. We argue this using some elementary geometry. See Fig. 4.6(b). Let v_{ijk} and v'_{ijk} be the two intersection points of ∂D_{ijk}^* and ∂D_1^+ , for $1 \leq i, j, k \leq m$. Such two intersection points coincide with each other when $i = j = 1$, and furthermore, they always exist due to our construction. Note that v_{11k} is also the intersection point of ∂D_k^0 and ∂D_1^+ . It is trivial to see that as the parameter ω gets smaller, v_{ijk} and v'_{ijk} lie closer to v_{11k} along ∂D_1^+ . They all coincide with v_{11k} when ω becomes 0. Since ω is a sufficiently small positive number, we are assured that v_{ijk} lies between $v_{11(k-1)}$ and v_{11k} along

∂D_1^+ , i.e., D_{ijk}^* does not contain D_{k-1}^0 , not to mention D_1^0, \dots, D_{k-2}^0 . Similarly, D_{ijk}^* does not contain D_{k+1}^0, \dots, D_m^0 . Moreover, D_{ijk}^* does not contain any disk of $\mathcal{D}^- \cup \mathcal{D}^+$. Hence, there is a disk touching D_i^- and D_j^+ from the outside and D_k^0 from the inside and not containing any disk of $\mathcal{D}^- \cup \mathcal{D}^+ \cup \mathcal{D}^0$ in its interior, for $1 \leq i, j, k \leq m$.

This proves our claim, and finishes our $\Omega(n^3)$ lower bound construction when the disks have the same radius. \square

Next, if the disks in \mathcal{D} are pairwise disjoint and the ratio of the radii of the largest to the smallest disk is bounded by λ , then we prove a refined bound on the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ that depends on λ .

Lemma 4.2.8. *If $\mathcal{P} = \{P_1, \dots, P_n\}$ is a set of n uncertain points in \mathbb{R}^2 whose uncertainty regions are pairwise-disjoint disks with radii in the range $[1, \lambda]$, a pair of curves in Γ intersects in $O(\lambda)$ points.*

Proof. Fix a pair of curves γ_1 and γ_2 , let D_1 and D_2 be the corresponding disks, and let c_1 and c_2 be their centers, respectively. By applying rotation and translation to the plane, we can assume D_1 and D_2 are centered on the x -axis, with D_1 to the left of D_2 .

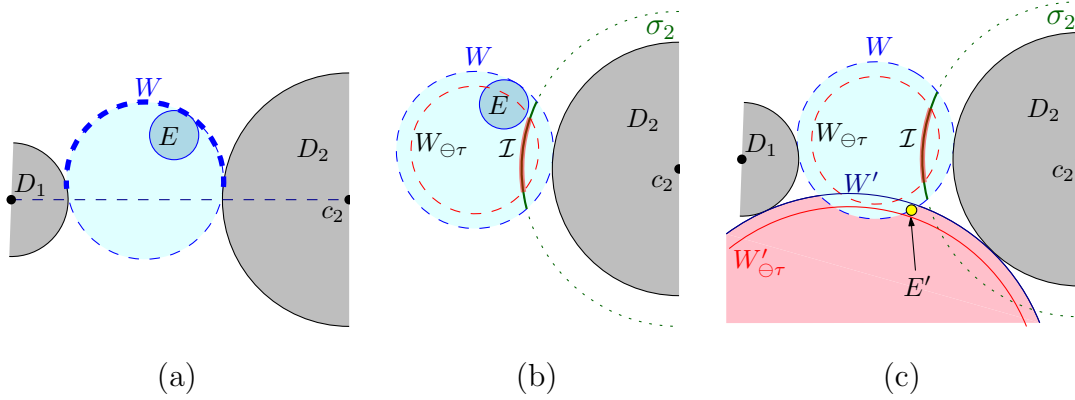


FIGURE 4.7. An illustration for the proof of Lemma 4.2.8.

For a parameter t , $1 \leq t \leq \lambda$, let \mathcal{D} denote the set of all the disks associated with \mathcal{P} , excluding D_1 and D_2 , with radii between t and $2t$. An intersection point $q \in \gamma_1 \cap \gamma_2$

corresponds to a *witness* disk W centered at q that touches both D_1 and D_2 from the outside, touches exactly one other disk $E \in \mathcal{D}$ from the inside, and properly contains no disks of \mathcal{D} . The family of disks that touch both D_1 and D_2 from the outside is a *pencil*, which sweeps over a portion of the plane as the tangency points with D_1 and D_2 move continuously (see Fig. 4.5(b)). A disk of \mathcal{D} can contribute at most two intersection points to $\gamma_1 \cap \gamma_2$, as its boundary gets swept over at most twice by the circles of the pencil.

We break ∂W into two curves, *top* and *bottom*, at W 's tangency points with D_1 and D_2 . For a disk $E \in \mathcal{D}$, if its tangency point with its witness disk W is on the top portion of W , then it is a *top tangency event*, otherwise it is a *bottom tangency event*. See Fig. 4.7(a). Let \mathcal{D}_1 (resp. \mathcal{D}_2) be the set of disks in \mathcal{D} that are closer to D_1 (resp. D_2).

Below we bound the number of top tangency events involving disks in \mathcal{D}_2 . Other tangency events are handled by a symmetric argument.

We remove from \mathcal{D}_2 all the disks at distance at most $T = \xi t$ from D_2 , where ξ is a sufficiently large constant. The ring with outer radius $r(D_2) + 4T$ and inner radius $r(D_2)$ has area

$$\alpha = \pi((r(D_2) + 4T)^2 - (r(D_2))^2) = O(Tr(D_2) + T^2) = O(t^2 + \lambda t),$$

as $r(D_2) \leq \lambda$. Disks removed from \mathcal{D}_2 have the following properties:

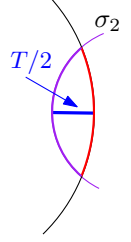
- (i) they are interior-disjoint, and
- (ii) their radii lie in the interval $[t, 2t]$, and
- (iii) they are contained in the aforementioned ring, and
- (iv) the area of each such disk is at least πt^2 .

Hence the number of removed disks is $O((t^2 + \lambda t)/t^2) = O(\lambda/t)$.

Consider the circle σ_2 of radius $r(D_2) + T/2$ centered at c_2 . Consider any disk $E \in \mathcal{D}_2$ and its witness disk W touching both D_1 and D_2 from the outside. Note

that since E has not been removed from \mathcal{D}_2 , $r(W) \geq (T + 2t)/2$; in particular it is larger than $T/2$. Let $W_{\ominus\tau}$ be the disk concentric with W with radius $r(W) - \tau$, where $\tau = 4t$. The disk $W_{\ominus\tau}$ is interior-disjoint from all disks in \mathcal{D}_2 , as E touches W from inside and W cannot fully contain any other disks from \mathcal{D}_2 .

The witness disk W covers an arc of length at least $T/2$ on σ_2 . Indeed, neither of these two disks contains the center of the other, and the inner distance between the two intersection arcs is $T/2$, see figure on the right. Similarly, let $J(E)$ be the arc $W_{\ominus\tau} \cap \sigma_2$. By the same argument, we have that $J(E)$ is of length at least $T/2 - \tau = \Omega(t)$.



The circumference of σ_2 is $2\pi(r(D_2) + T/2) = O(\lambda)$, so if the arcs $J(E)$, for $E \in \mathcal{D}_2$, are pairwise disjoint, we are done, as this implies that there could be at most $\lambda/(T/2 - \tau) = O(\lambda/t)$ such arcs and thus the size of the original \mathcal{D}_2 is $O(\lambda/t)$. See Fig. 4.7(b). We will therefore proceed to prove that any two such arcs are disjoint.

So, consider two disks $E, E' \in \mathcal{D}_2$, both realizing a top tangency event. Let W (resp. W') be the witness disk that is tangent to D_1, D_2 and E (resp. E'). Assume that the tangency of W with D_2 is clockwise to the tangency of W' with D_2 (i.e., E is “above” E'). If $W_{\ominus\tau}$ and $W'_{\ominus\tau}$ are disjoint then their corresponding arcs are disjoint. Otherwise, as we already observed, E' and $W_{\ominus\tau}$ are disjoint. Furthermore, it is easy to verify that E' must lie in the region “between” σ_2 , $W_{\ominus\tau}$, and $W'_{\ominus\tau}$, and therefore the arcs $J(E)$ and $J(E')$ are disjoint, as claimed; refer to Fig. 4.7(c).

We now repeat the above counting argument, for $t = 1, 2, 4, \dots, 2^m$, where $m = \lceil \log_2 \lambda \rceil$, concluding that the number of intersection points between γ_1 and γ_2 is bounded by $\sum_{i=1}^m O(\lambda/2^i) = O(\lambda)$, as claimed. \square

Theorem 4.2.9. *Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n uncertain points in \mathbb{R}^2 such that their uncertainty regions are pairwise-disjoint disks and that the ratio of the largest and the smallest radii of the disks is at most λ . Then the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ is*

$O(\lambda n^2)$, and it can be computed in $O(n^2 \log n + \mu)$ expected time, where μ is the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$. Furthermore, there exists such a set \mathcal{P} of uncertain points for which $\mathcal{V}_{\neq 0}(\mathcal{P})$ has $\Omega(n^2)$ complexity.

Proof. The upper bound on the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ follows from Lemma 4.2.8. By the same argument as in the proof of Theorem 4.2.5, $\mathcal{V}_{\neq 0}(\mathcal{P})$ can be computed in $O(n^2 \log n + \mu)$ time, where μ is the number of vertices in $\mathcal{V}_{\neq 0}(\mathcal{P})$.

Next we show that there exists a set \mathcal{P} of n uncertain points in \mathbb{R}^2 such that $\mathcal{V}_{\neq 0}(\mathcal{P})$ has $\Omega(n^2)$ vertices. Assume that $n = 2m$ for some positive integer m . All the disks D_i have the same radius 1, centered at $c_i = (4(i - m) - 2, 0)$, for $1 \leq i \leq 2m$. Any pair (P_i, P_j) satisfying that $j - i \geq 2$ and $j + i$ is even determines 2 vertices: $v_1 = (2(i + j - 2m - 1), (j - i)^2 - 1)$, and $v_2 = (2(i + j - 2m - 1), 1 - (j - i)^2)$, of $\mathcal{V}_{\neq 0}$ (realized with $P_k, k = \frac{j+i}{2}$) (Fig. 4.8). Any pair (P_i, P_j) satisfying that $j - i \geq 2$ and $j + i$ is odd determines 2 vertices: $v_1 = (2(i + j - 2m - 1), (j - i)\sqrt{(j - i)^2 - 4})$, and $v_2 = (2(i + j - 2m - 1), (i - j)\sqrt{(j - i)^2 - 4})$, of $\mathcal{V}_{\neq 0}$ (realized with $P_k, k = \lfloor \frac{j+i}{2} \rfloor$ or $k = \lceil \frac{j+i}{2} \rceil$). One can verify that $\delta_i(v) = \delta_j(v) = \Delta_k(v) \leq \Delta_l(v)$, for $1 \leq l \leq n$, $v \in \{v_1, v_2\}$. Hence we obtain a lower bound of $\Omega(n^2)$ for the complexity of $\mathcal{V}_{\neq 0}$. \square

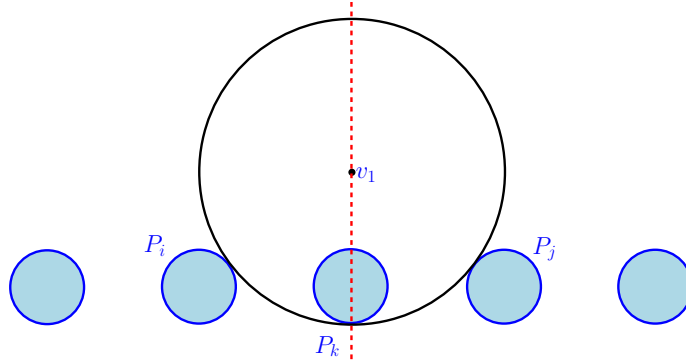


FIGURE 4.8. Any pair (P_i, P_j) satisfying $j - i \geq 2$ determines 2 vertices of $\mathcal{V}_{\neq 0}$. Only the vertex v_1 is shown.

Storing \mathcal{P}_ϕ 's for $\mathcal{V}_{\neq 0}(\mathcal{P})$. We store the index i of each uncertain point P_i instead of P_i itself. If we store \mathcal{P}_ϕ for each cell ϕ of $\mathcal{V}_{\neq 0}(\mathcal{P})$ explicitly, the size increases by

a factor of n . However, we observe that for two adjacent cells ϕ, ϕ' of $\mathcal{V}_{\neq 0}(\mathcal{P})$, i.e., two cells that share a common edge, $|\mathcal{P}_\phi \oplus \mathcal{P}_{\phi'}| = 1$, where \oplus denotes the symmetric difference of two sets. Therefore, using a persistent data structure [54], we can store \mathcal{P}_ϕ for all cells of $\mathcal{V}_{\neq 0}(\mathcal{P})$ in $O(\mu)$ space, where μ is the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$, so that for any cell ϕ , \mathcal{P}_ϕ can be retrieved in $O(\log n + |\mathcal{P}_\phi|)$ time.² By combining this with a planar point-location data structure [52], we obtain the following:

Theorem 4.2.10. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 , and let μ be the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$. Then $\mathcal{V}_{\neq 0}(\mathcal{P})$ can be preprocessed in $O(\mu \log \mu)$ time into a data structure of size $O(\mu)$ so that for a query point $q \in \mathbb{R}^2$, $\text{NN}_{\neq 0}(q, \mathcal{P})$ can be computed in $O(\log n + t)$ time, where t is the output size.*

Remarks. This bound can be extended to the case when each uncertainty region is a convex α -fat semialgebraic set of constant description complexity. A convex set C is called α -fat, if there exist two concentric disks, $D \subseteq C \subseteq D'$, such that the ratio between the radii of D' and D is at most α . The constant of proportionality also depends on α and the number and maximum degree of polynomials defining the uncertainty regions.

4.2.2. Discrete case

We now analyze the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ when the distribution of each point P_i in \mathcal{P} is discrete. Let $P_i = \{p_{i1}, \dots, p_{ik}\}$. For $1 \leq j \leq k$, let $w_{ij} = \Pr[P_i \text{ is } p_{ij}]$. As in the previous section, for a point x , let $\Delta_i(x) = \max_{1 \leq j \leq k} d(x, p_{ij})$ and $\delta_i(x) = \min_{1 \leq j \leq k} d(x, p_{ij})$. Note that the projection of the graph of Δ_i (resp. δ_i) onto the xy -plane is the farthest-point (resp. nearest-point) Voronoi diagram of P_i . Let $\Delta(x) = \min_{1 \leq i \leq n} \Delta_i(x)$. For each i , let $\gamma_i = \{x \in \mathbb{R}^2 \mid \delta_i(x) = \Delta(x)\}$, and set

²If the curves of Γ intersect transversally at every vertex, it suffices to store \mathcal{P}_ϕ for each cell of $\mathcal{V}_{\neq 0}(\mathcal{P})$. Otherwise one may have to store \mathcal{P}_ϕ for edges and vertices of $\mathcal{V}_{\neq 0}(\mathcal{P})$. This does not affect the asymptotic performance of the data structure.

$\Gamma = \{\gamma_1, \dots, \gamma_n\}$. Then $\mathcal{V}_{\neq 0}(\mathcal{P})$ is the planar subdivision induced by Γ , as defined above. We need the following lemma to bound the complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$.

Lemma 4.2.11. *For any pair i, j , $1 \leq i \neq j \leq n$, let $\gamma_{ij} = \{x \in \mathbb{R}^2 \mid \delta_i(x) = \Delta_j(x)\}$, then γ_{ij} is a convex polygonal curve with $O(k)$ vertices.*

Proof. Fix a point $p \in \mathbb{R}^2$, and we define a linear function $g: \mathbb{R}^2 \rightarrow \mathbb{R}$ as

$$g(x) = d^2(x, p) - \|x\|^2 = \|p\|^2 - 2\langle x, p \rangle. \quad (4.5)$$

For $1 \leq i \leq n$, define $\varphi_i(x) = \min_{1 \leq j \leq k} g(x, p_{ij})$ and $\Phi_i(x) = \max_{1 \leq j \leq k} g(x, p_{ij})$. Then for any pair i, j , $\delta_i(x) = \Delta_j(x)$ if and only if $\varphi_i(x) = \Phi_j(x)$. Hence, γ_{ij} is also the zero set of the function $\Phi_j(x) - \varphi_i(x)$.

Φ_j is the upper envelope of k linear functions, and thus is a piecewise-linear convex function. Similarly, φ_i , the lower envelope of k linear functions, is a piecewise-linear concave function. Hence $\Phi_j(x) - \varphi_i(x)$ is a piecewise-linear convex function, which implies that $\gamma_{ij} = \{x \in \mathbb{R}^2 \mid \Phi_j(x) = \varphi_i(x)\}$ is a convex polygonal curve. Since γ_{ij} is the projection of the intersection curve of the graphs of Φ_j and φ_i , each of which is the surface of an unbounded convex polyhedron with at most k faces, γ_{ij} has $O(k)$ vertices. \square

Theorem 4.2.12. *Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n uncertain points in \mathbb{R}^2 , where each P_i has a discrete distribution of size at most k . The complexity of $\mathcal{V}_{\neq 0}(\mathcal{P})$ is $\mu = O(kn^3)$, and it can be computed in $O(n^2 \log n + \mu)$ expected time. Furthermore, it can be preprocessed in additional $O(\mu)$ time into a data structure of size $O(\mu)$ so that an $\text{NN}_{\neq 0}(q)$ query can be answered in $O(\log \mu + t)$, where t is the output size.*

Proof. We follow the same argument as in the proof of Theorem 4.2.5. We need to bound the number of intersection points between a pair of curves γ_i and γ_j . Fix an index u . Let $\gamma_{iu} = \{x \in \mathbb{R}^2 \mid \delta_i(x) = \Delta_u(x)\}$ and $\gamma_{ju} = \{x \in \mathbb{R}^2 \mid \delta_j(x) = \Delta_u(x)\}$.

By Lemma 4.2.11, each of γ_{iu} and γ_{ju} is a convex polygonal curve in \mathbb{R}^2 with $O(k)$ vertices. Since two convex polygonal curves in general position with n_1 and n_2 vertices intersect in at most $n_1 + n_2$ points, γ_{iu} and γ_{ju} intersect at $O(k)$ points. Hence γ_i and γ_j intersect at $O(kn)$ points, implying that $\mathcal{V}_{\neq 0}(\mathcal{P})$ has $O(kn^3)$ vertices. The running time follows from the proof of Theorem 4.2.5. \square

4.3. Data Structures for $\text{NN}_{\neq 0}$ Queries

With the maximum size of $\mathcal{V}_{\neq 0}$ being $\Theta(n^3)$ or $\Theta(n^2)$, we present data structures with less space that can answer $\text{NN}_{\neq 0}$ queries in poly-logarithmic or sublinear time. We consider both continuous and discrete cases.

An $\text{NN}_{\neq 0}(q)$ query is answered in two stages. The first stage computes $\Delta(q)$, and the second stage computes all points $P_i \in \mathcal{P}$ for which $\delta_i(q) < \Delta(q)$. We build a separate data structure for each stage.

Continuous case. We assume that the uncertainty region of each point P_i is a disk D_i of radius r_i centered at c_i . Recall from Section 4.2 that the projection of the graph of the function Δ onto the xy -plane, a planar subdivision \mathbb{M} , is the (additive-weighted) Voronoi diagram of the points c_1, \dots, c_n , and it has linear complexity. Hence \mathbb{M} can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n)$ so that for a query point $q \in \mathbb{R}^2$, $\Delta(q)$ can be computed in $O(\log n)$ time.

Next we wish to report all points $P_i \in \mathcal{P}$ for which $\delta_i(q) < \Delta(q)$. Note that the projection of the graph of the lower envelope of $\{\delta_1, \dots, \delta_n\}$ is also an (additive) weighted Voronoi diagram of the points c_1, \dots, c_n and has linear complexity. Therefore, using the data structure by Agarwal *et al.* [10, Thm 4.1], the above query can be answered in $O(\log n + t)$ time. The preprocessing time and the size of the data structure are $O(n^{1+\varepsilon})$, for any constant $\varepsilon > 0$. We thus obtain the following:

Theorem 4.3.1. *Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n uncertain points in \mathbb{R}^2 so*

that the uncertainty region of each P_i is a disk. Then \mathcal{P} can be preprocessed into a data structure of size $O(n^{1+\varepsilon})$, for any constant $\varepsilon > 0$, so that an $\text{NN}_{\neq 0}(q)$ query can be answered in $O(\log n + t)$ time, where t is the output size and the hiding constants in the big O notation depend on ε .

Remarks. (i) Note that Theorem 4.3.1 gives a better result than Theorem 4.2.10 if the uncertainty regions of \mathcal{P} are allowed to intersect, but the Voronoi–diagram-based data structure is much simpler and practical.

(ii) If we use L_1 or L_∞ metric to compute the distance between points and use disks in L_1 or L_∞ metric (i.e., a diamond or a square), then an $\text{NN}_{\neq 0}(q)$ query can be answered in $O(\log^2 n + t)$ time using $O(n \log^2 n)$ space: the first stage remains the same and the second stage reduces to reporting a set of axis-aligned squares that intersect a query (axis-aligned) square [3].

Discrete case. If the distribution of each P_i is discrete, then the functions Δ_i and δ_i are complex and thus the data structure for $\text{NN}_{\neq 0}(q)$ queries is more involved. First we observe that the problem of reporting all points $\mathcal{P}_i \in \mathcal{P}$ such that $\delta_i(q) \leq R$ for a query point $q \in \mathbb{R}^2$ and $R > 0$, can be formulated as a colored disk range reporting. Namely, we color all k points of P_i with color i . Let $S = \bigcup_{i=1}^n P_i$. Then given a disk D of radius R centered at q , we wish to report the colors of all points in S that lie inside D — each color should be reported only once. Using the results in [68], this can be done with $O(\log(nk) + t)$ query time using $O((nk)^{2+\varepsilon})$ space (for any constant $\varepsilon > 0$), or with $O((nk)^{1/2+\varepsilon} + t)$ query time using $O(nk)$ space.

It thus suffices to describe how we compute $\Delta(q)$ for a query point $q \in \mathbb{R}^2$. Recall that the projection of Δ_i onto the xy -plane, for $1 \leq i \leq n$, is the farthest-point Voronoi diagram of P_i , and that Δ is the lower envelope of $\Delta_1, \dots, \Delta_n$. Following the same argument as by Huttenlocher *et al.* [74], where they studied the upper envelope of $\delta_1, \dots, \delta_n$, we can prove the following.

Lemma 4.3.2. *The xy -projection of the graph of the function Δ is a planar subdivision with $O(n^2k\alpha(nk))$ vertices, and it can be computed in $O(n^2k \log(nk))$ time, where $\alpha(\cdot)$ is the inverse Ackerman function.*

Hence by preprocessing the projection of Δ for point-location queries, $\Delta(q)$ can be computed in $O(\log(nk))$ time, for any query point q .

If we wish to construct a linear-size data structure, we do not construct Δ . Instead we proceed as follows. As in the proof of Lemma 4.2.11 (cf. Eq. (4.5)), the graph of each Δ_i can be represented as a piecewise-linear xy -monotone surface. By triangulating each face of Δ_i , we can assume Δ_i is a triangulated xy -monotone surface in \mathbb{R}^3 . Let ℓ_q be the oriented line in the direction of the $(+z)$ -axis. Our goal is to find the first triangle of a surface in $\mathbf{\Delta} = \{\Delta_1, \dots, \Delta_n\}$ intersected by ℓ_q . The total number of triangles in $\mathbf{\Delta}$ is $O(nk)$.

As shown in [9,81], this so-called *vertical ray shooting* problem is equivalent to the *point-enclosure* problem in the xy -projections of triangles in $\mathbf{\Delta}$, under the semi-group model. Namely, report all triangles (in \mathbb{R}^2) of the projection that contain a query point, as the union of a small set of *canonical* subsets; a family of canonical subsets is constructed at the preprocessing stage which does not depend on the choice of a query point. Using multi-level partition trees, a point-enclosure query, and thus a vertical ray shooting query, can be answered in $O((nk)^{1/2+\varepsilon})$, for any constant $\varepsilon > 0$, using $O(nk)$ space and $O(nk \log(nk))$ preprocessing [3]. Putting everything together, we conclude the following.

Theorem 4.3.3. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 , each of size at most k . \mathcal{P} can be preprocessed into a data structure of size $O((nk)^{2+\varepsilon})$, for any constant $\varepsilon > 0$, so that an $\text{NN}_{\neq 0}(q)$ query can be answered in $O(\log(nk) + t)$ time, or into a data structure of size $O(nk)$ with $O((nk)^{1/2+\varepsilon} + t)$ query time, where t is the output size. The preprocessing times are $O((nk)^{2+\varepsilon})$ and $O(nk \log(nk))$ time, respectively.*

The hiding constants in the big O notation depend on ε .

4.4. Quantification Probabilities

We begin with exact algorithms for uncertain point sets, in which each uncertain point has k possible locations. We can build a structure called the *probabilistic Voronoi diagram* $\mathcal{V}_{\text{Pr}}(\mathcal{P})$ that decomposes \mathbb{R}^2 into a set of cells, so that any point q in a cell has the same $\pi_i(q)$ value for all $P_i \in \mathcal{P}$; that is, for any point q in this cell, we know exactly the probability of each point $P \in \mathcal{P}$ being the NN of q .

Lemma 4.4.1. *For a set \mathcal{P} of n uncertain points in \mathbb{R}^2 , each with k possible locations, the complexity of $\mathcal{V}_{\text{Pr}}(\mathcal{P})$ is $O(n^4 k^4)$. Moreover, there exists a set \mathcal{P} of n uncertain points in \mathbb{R}^2 with $k = 2$ such that $\mathcal{V}_{\text{Pr}}(\mathcal{P})$ has size $\Omega(n^4)$.*

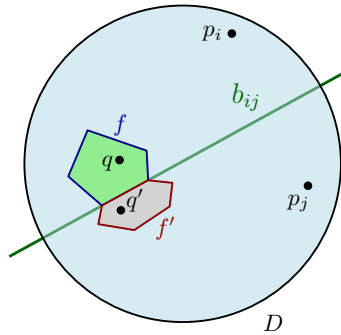


FIGURE 4.9. An illustration of the proof. Inside the unit disk D , two adjacent faces f and f' share a portion of the bisector b_{ij} defined by p_i and p_j .

Proof. We first prove the upper bound. There are nk possible locations. Each pair of possible locations determines a bisector, resulting in $O(n^2 k^2)$ bisectors. These bisectors partition the plane into $O(n^4 k^4)$ convex cells so that the order of all distances to each of the nk possible locations, and thus by Eq. (4.2) also all the qualification probabilities, are preserved within each cell. Therefore the resulting planar subdivision is a refinement of $\mathcal{V}_{\text{Pr}}(\mathcal{P})$, and thus $O(n^4 k^4)$ is an upper bound on the complexity of $\mathcal{V}_{\text{Pr}}(\mathcal{P})$.

Next, we show that there exists a set \mathcal{P} of n uncertain points in \mathbb{R}^2 with $k = 2$ such that $\mathcal{V}_{\text{Pr}}(\mathcal{P})$ has size $\Omega(n^4)$. For simplicity, we describe a degenerate configuration of points, but the argument can be generalized to a non-degenerate configuration as well, by being more careful. For every $1 \leq i \leq n$, $P_i \in \mathcal{P}$ has two possible locations p_i and p'_i , each with probability 0.5. Let D be the unit disk centered at the origin. We choose p_1, \dots, p_n inside D so that the bisector b_{ij} of every pair p_i, p_j , for $i < j$, is a distinct line and all pairs of bisectors intersect inside D . We place all p'_i 's at the same location far away from D , say, at $\bar{p} = (100, 0)$. Note that the bisector of \bar{p} and p_i , for any $i \leq n$, does not intersect inside D , so for any point $q \in D$, $d(p_i, q) < d(\bar{p}, q)$.

Let \mathcal{A} be the arrangement of the bisectors $\{b_{ij} \mid 1 \leq i < j \leq n\}$. Since all pairs of bisectors intersect inside D , $\mathcal{A} \cap D$ has $\Theta(n^4)$ faces. Let f, f' be any two adjacent faces of \mathcal{A} inside D , let b_{ij} be the bisector separating f and f' , and let q, q' be arbitrary points in the interior of f, f' , respectively. Without loss of generality, assume that $d(p_i, q) < d(p_j, q)$, then $d(p_i, q') > d(p_j, q')$. See Fig. 4.9. Suppose there are r , $0 \leq r < n - 1$, points of $\{p_1, \dots, p_n\}$ that are closer to q than p_i , i.e., p_i (resp. p_j) is the $(r + 1)$ -st NN of q (resp. q') among $\{p_1, \dots, p_n\}$. Then, by Eq. (4.2),

$$\pi_i(q) = 0.5 \cdot (1 - 0.5)^r + 0.5 \cdot (1 - 0.5)^{n-1} = 0.5^{r+1} + 0.5^n.$$

$$\pi_j(q) = 0.5 \cdot (1 - 0.5)^{r+1} + 0.5 \cdot (1 - 0.5)^{n-1} = 0.5^{r+2} + 0.5^n.$$

Symmetrically, $\pi_i(q') = 0.5^{r+2} + 0.5^n$ and $\pi_j(q') = 0.5^{r+1} + 0.5^n$. In particular $\pi_i(q) \neq \pi_i(q')$ and $\pi_j(q) \neq \pi_j(q')$. In other words, any two adjacent faces of \mathcal{A} inside D have distinct quantification probability vectors, implying that $\mathcal{V}_{\text{Pr}}(\mathcal{P})$ has $\Omega(n^4)$ complexity. \square

$\mathcal{V}_{\text{Pr}}(\mathcal{P})$ is too large to store in practice, so we explore how to approximate $\pi_i(q)$.

4.4.1. A Monte-Carlo algorithm

In this section we describe a simple Monte-Carlo approach to build a data structure for quickly computing $\widehat{\pi}_i(q)$ for all P_i for any query point q , which approximates the quantification probability $\pi_i(q)$. For a fixed value s , to be specified later, the preprocessing step works in s rounds. In the j -th round the algorithm creates a sample $R_j = \{r_{j1}, r_{j2}, \dots, r_{jn}\} \subseteq \mathbb{R}^2$ by choosing each r_{ji} using the distribution of P_i . For each $j \in \{1, \dots, s\}$, we construct the Voronoi diagram $\text{Vor}(R_j)$ in $O(n \log n)$ time and preprocess it for point-location queries in additional $O(n \log n)$ time.

To estimate quantification probabilities of a query q , we initialize a counter $c_i = 0$ for each point P_i . For each $j \in \{1, \dots, s\}$, we find the point $r_{ji} \in R_j$ whose cell in $\text{Vor}(R_j)$ contains the query point q , and increment c_i by 1. Finally we estimate $\widehat{\pi}_i(q) = c_i/s$. Note that at most s distinct c_i 's have nonzero values, so we can implicitly set the others to 0.

Discrete case. If each $P_i \in \mathcal{P}$ has a discrete distribution of size k , then this algorithm can be implemented very efficiently. Each r_i can be selected in $O(\log k)$ time after preprocessing each P_i , in $O(k)$ time, into a balanced binary tree [99]. Thus total preprocessing takes $O(s(n(\log n + \log k)) + nk) = O(nk + sn \log(nk))$ time and $O(sn)$ space, and each query takes $O(s \log n)$ time.

It remains to determine the value of s so that $|\pi_i(q) - \widehat{\pi}_i(q)| \leq \varepsilon$ for all P_i and all queries q , with probability at least $1 - \delta$. For fixed q , P_i , and instantiation R_j , let X_i be the random indicator variable, which is 1 if r_i is the NN of q and 0 otherwise. Since $\mathbb{E}[X_i] = \pi_i(q)$ and $X_i \in \{0, 1\}$, applying a Chernoff-Hoeffding bound [99] to

$$\widehat{\pi}_i(q) = \frac{c_i}{s} = \frac{1}{s} \sum X_i,$$

we obtain that

$$\Pr [|\widehat{\pi}_i(q) - \pi_i(q)| \geq \varepsilon] \leq 2 \exp(-2\varepsilon^2 s). \quad (4.6)$$

For each cell of $\mathcal{V}_{\text{Pr}}(\mathcal{P})$, we choose one point, and let Q be the resulting set of points. If $|\widehat{\pi}_i(q) - \pi_i(q)| \leq \varepsilon$ for every point $q \in Q$, then $|\widehat{\pi}_i(q) - \pi_i(q)| \leq \varepsilon$ for every point $q \in \mathbb{R}^2$. Since there are n different values of i , by applying the union bound to Eq. (4.6), the probability that there exist a point $q \in \mathbb{R}^2$ and an index $i \in \{1, \dots, n\}$ with $|\widehat{\pi}_i(q) - \pi_i(q)| \geq \varepsilon$ is at most $2n|Q| \exp(-2\varepsilon^2 s)$. Hence, by setting

$$s = \frac{1}{2\varepsilon^2} \ln \frac{2n|Q|}{\delta},$$

$|\widehat{\pi}_i(q) - \pi_i(q)| \leq \varepsilon$ for all $q \in \mathbb{R}^2$ and for all $i \in \{1, \dots, n\}$, with probability at least $1 - \delta$. By Lemma 5.4.1, $|Q| = O(n^4 k^4)$, so we obtain the following result.

Theorem 4.4.2. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 , each with a discrete distribution of size k , and let $\varepsilon, \delta \in (0, 1)$ be two parameters. \mathcal{P} can be preprocessed, in*

$$O(nk + (n/\varepsilon^2) \log(nk) \log(nk/\delta))$$

time, into a data structure of size $O((n/\varepsilon^2) \log(nk/\delta))$, which computes, for any query point $q \in \mathbb{R}^2$, in $O((1/\varepsilon^2) \log(nk/\delta) \log n)$ time, a value $\widehat{\pi}_i(q)$ for every P_i such that $|\pi_i(q) - \widehat{\pi}_i(q)| \leq \varepsilon$ for all i with probability at least $1 - \delta$.

Continuous case. There are two technical issues in extending this technique and analysis to continuous distributions. First, we instantiate a certain point r_i from each P_i . Herein we assume the representation of the pdf is such that this can be done in constant time for each P_i .

Second, we need to bound the number of distinct queries that need to be considered to apply the union bound as we did above. Since $\pi_i(q)$ may vary continuously with the query location, unlike the discrete case, we cannot hope for a bounded number of distinct results. However, we just need to define a finite set \overline{Q} of query points so that for any point $q \in \mathbb{R}^2$, there is a point $q' \in \overline{Q}$ such that $\max_i |\pi_i(q) - \pi_i(q')| \leq \varepsilon/2$. Then we can choose s large enough so that it permits at most $\varepsilon/2$ error on each query

in \bar{Q} . Specifically, choosing $s = O((1/\varepsilon^2) \log(n|\bar{Q}|/\delta))$ is sufficient, so all that remains is to bound $|\bar{Q}|$.

To choose \bar{Q} , we show that each pdf of P_i can be approximated with a discrete distribution of size $O((n^2/\varepsilon^2) \log(n/\delta))$, and then reduce the problem to the discrete case.

For parameters $\alpha > 0$ and $\delta' \in (0, 1)$, set

$$k(\alpha) = \frac{c}{\alpha^2} \log \frac{1}{\delta'},$$

where c is a constant. For each $i \in \{1, \dots, n\}$, we choose a random sample $\bar{P}_i \subset P_i$ of size $k(\alpha)$, according to the distribution defined by the location pdf f_i of P_i . We regard \bar{P}_i as an uncertain point with uniform location probability. Set $\bar{\mathcal{P}} = \{\bar{P}_1, \dots, \bar{P}_n\}$.

For a point $q \in \mathbb{R}^2$, let $\bar{G}_{q,i}$ denote the cdf of the distance between q and \bar{P}_i , i.e., $\bar{G}_{q,i}(r) = \Pr[d(q, \bar{P}_i) \leq r]$, or equivalently, it is the probability of \bar{P}_i lying in the disk of radius r centered at q . A well-known result in the theory of random sampling [89, 121] implies that for all $r \geq 0$,

$$|G_{q,i}(r) - \bar{G}_{q,i}(r)| \leq \alpha, \tag{4.7}$$

with probability at least $1 - \delta'$, provided that the constant c in $k(\alpha)$ is chosen sufficiently large.

Let $\bar{\pi}_i(q)$ denote the probability of \bar{P}_i being the NN of q in $\bar{\mathcal{P}}$. We prove the following:

Lemma 4.4.3. *For any $q \in \mathbb{R}^2$ and for any fixed $i \in \{1, \dots, n\}$,*

$$|\pi_i(q) - \bar{\pi}_i(q)| \leq \alpha n,$$

with probability at least $1 - \delta'$.

Proof. Recall that by Eq. (4.1),

$$\pi_i(q) = \int_0^\infty g_{q,i}(r) \prod_{j \neq i} (1 - G_{q,j}(r)) dr.$$

Using Eq. (6.2), and the fact that $G_{q,j}(r), \bar{G}_{q,j}(r) \in [0, 1]$ for all j , we obtain

$$\pi_i(q) \leq \int_0^\infty g_{q,i}(r) \prod_{j \neq i} (1 - \bar{G}_{q,j}(r)) dr + (n-1)\alpha.$$

Note that $\prod_{j \neq i} (1 - \bar{G}_{q,j}(r))$ is the probability that the closest point of q in $\bar{\mathcal{P}} \setminus \{\bar{P}_i\}$ is at least distance r away from q . Let $h_{q,i}$ be the pdf of the distance between q and its closest point in $\bar{\mathcal{P}} \setminus \{\bar{P}_i\}$. Then

$$\prod_{j \neq i} (1 - \bar{G}_{q,j}(r)) = \int_r^\infty h_{q,i}(\theta) d\theta.$$

Therefore

$$\pi_i(q) \leq \int_0^\infty \int_r^\infty g_{q,i}(r) h_{q,i}(\theta) d\theta dr + (n-1)\alpha.$$

By reversing the order of integration, we obtain

$$\begin{aligned} \pi_i(q) &\leq \int_0^\infty \int_0^\theta h_{q,i}(\theta) g_{q,i}(r) dr d\theta + (n-1)\alpha \\ &= \int_0^\infty h_{q,i}(\theta) G_{q,i}(\theta) d\theta + (n-1)\alpha \\ &\leq \int_0^\infty h_{q,i}(\theta) (\bar{G}_{q,i}(\theta) + \alpha) d\theta + (n-1)\alpha \\ &\quad \text{(using Eq. (6.2))} \\ &= \int_0^\infty h_{q,i}(\theta) \bar{G}_{q,i}(\theta) d\theta + n\alpha \\ &= \bar{\pi}_i(q) + n\alpha. \end{aligned}$$

A similar argument shows that $\pi_i(q) \geq \bar{\pi}_i(q) - n\alpha$. This completes the proof of the lemma. \square

Thus by setting $\alpha = \varepsilon/2n$, a random sample \bar{P}_i of size $O((n^2/\varepsilon^2) \log(n/\delta))$ from each P_i ensures that

$$|\pi_i(q) - \bar{\pi}_i(q)| \leq \varepsilon/2 \tag{4.8}$$

for all queries. By choosing $\delta' = \delta/2n$, Eq. (4.8) holds for all $i \in \{1, \dots, n\}$ with probability at least $1 - \delta/2$.

We consider $\mathcal{V}_{\text{Pr}}(\overline{\mathcal{P}})$, choose one point from each of its cells, and set $\overline{\mathcal{Q}}$ to be the resulting set of points. For a point $q \in \mathbb{R}^2$, let $\bar{q} \in \overline{\mathcal{Q}}$ be the representative point of the cell of $\mathcal{V}_{\text{Pr}}(\overline{\mathcal{P}})$ that contains q . Then $|\pi_i(q) - \bar{\pi}_i(\bar{q})| < \varepsilon/2$ for all points $q \in \mathbb{R}^2$ and $i \in \{1, \dots, n\}$, with probability at least $1 - \delta/2$.

Now applying the analysis for the discrete case on the point set $\overline{\mathcal{P}}$, if we choose

$$s = O\left(\frac{1}{\varepsilon^2} \log \frac{n|\overline{\mathcal{Q}}|}{\delta}\right),$$

then $|\bar{\pi}_i(q) - \hat{\pi}_i(q)| < \varepsilon$ for all points $q \in \mathbb{R}^2$ and for all $i \in \{1, \dots, n\}$ with probability at least $1 - \delta/2$. Since

$$|\overline{P}_i| = k\left(\frac{\varepsilon}{2n}\right) = O\left(\frac{n^2}{\varepsilon^2} \log \frac{n}{\delta}\right),$$

by Lemma 5.4.1,

$$|\overline{\mathcal{Q}}| = O\left(n^4 \left(k\left(\frac{\varepsilon}{2n}\right)\right)^4\right) = O\left(\frac{n^{12}}{\varepsilon^8} \log^4 \frac{n}{\delta}\right).$$

Putting everything together, we obtain the following.

Theorem 4.4.4. *Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n uncertain points in \mathbb{R}^2 so a random instantiation of P_i can be performed in $O(1)$ time, let $0 < \varepsilon, \delta < 1$. \mathcal{P} can be preprocessed in $O((n/\varepsilon^2) \log(n/\varepsilon\delta) \log n)$ time into a data structure of size $O((n/\varepsilon^2) \log(n/\varepsilon\delta))$, which computes for any query point $q \in \mathbb{R}^2$, in $O((1/\varepsilon^2) \log(n/\varepsilon\delta) \log n)$ time, a value $\hat{\pi}_i(q)$ for every P_i such that $|\pi_i(q) - \hat{\pi}_i(q)| \leq \varepsilon$ for all i with probability at least $1 - \delta$.*

4.4.2. A spiral-search algorithm

If the distribution of each point in \mathcal{P} is discrete, then there is an alternative approach to approximate the quantification probabilities for a given query q : set a parameter

$m > 1$, choose m points of $S = \bigcup_{i=1}^n P_i$ that are closest to q , and use only these m points to estimate $\pi_i(q)$ for each P_i . We show this works for a small value of m when, for each P_i , each location is approximately equally likely, but is not efficient if the location probabilities vary significantly.

Let $P_i = \{p_{i1}, \dots, p_{ik}\}$ and $w_{ij} = \Pr[P_i = p_{ij}]$. Set $S = \bigcup_{i=1}^n P_i$. We refer to the quantity

$$\rho = \frac{\max_{i,j} w_{ij}}{\min_{i,j} w_{ij}} \quad (4.9)$$

as the *spread* of location probabilities. Set

$$m(\rho, \varepsilon) = \rho k \ln(\rho/\varepsilon) + k - 1.$$

Fix a query point $q \in \mathbb{R}^2$, and let $\bar{S} \subseteq S$ be the $m(\rho, \varepsilon)$ nearest neighbors of q in S . Let $\bar{P}_i = \bar{S} \cap P_i$, and $\bar{\mathcal{P}} = \{\bar{P}_1, \dots, \bar{P}_n\}$. Note that $\bar{w}_i = \sum_{p_{i,a} \in \bar{P}_i} w_{i,a}$ is not necessarily equal to 1, so we cannot regard \bar{P}_i as an uncertain point in our model, but still it will be useful to think of \bar{P}_i as an uncertain point that does not exist with probability $1 - \bar{w}_i$.

For a set Y of points and another point $\xi \in \mathbb{R}^2$, let

$$Y[\xi] = \{p \in Y \mid d(q, p) \leq d(q, \xi)\}.$$

Then for a point $p := p_{i,a} \in P_i$, $\pi_p(q)$, the probability that p is the NN of q in \mathcal{P} is

$$\pi_p(q) = w_{i,a} \prod_{j \neq i} \left(1 - \sum_{p_{j,\ell} \in P_j[p]} w_{j,\ell}\right). \quad (4.10)$$

Moreover,

$$\pi_i(q) = \sum_{p_{i,a} \in P_i} \pi_{p_{i,a}}(q). \quad (4.11)$$

For each $i \leq n$, we analogously define a quantity $\hat{\pi}_i(q)$ using Eq. (4.10) and Eq. (4.11) but replacing P_j with \bar{P}_j for every $j \in \{1, \dots, n\}$. Intuitively, if $\bar{\mathcal{P}}$ were a family of uncertain points, then $\hat{\pi}_i(q)$ would be the probability of \bar{P}_i being the NN of q in $\bar{\mathcal{P}}$.

Lemma 4.4.5. For all $i \in \{1, \dots, n\}$,

$$|\pi_i(q) - \widehat{\pi}_i(q)| \leq \varepsilon.$$

Proof. Fix a point $p \in P_i$. Set $x_j = |P_j[p]|$ and $m = \sum_{j \neq i} x_j$. Note that each $w_{j,a}$ satisfies

$$1/\rho k \leq w_{j,a} \leq \rho/k.$$

Then for a point $p := p_{i,a} \in P_i$, we obtain using Eq. (4.10)

$$\begin{aligned} \pi_p(q) &= w_{i,a} \prod_{j \neq i} \left(1 - \sum_{p_\ell \in P_j[p]} w_{j,\ell}\right) \\ &\leq \frac{\rho}{k} \prod_{j \neq i} \left(1 - \frac{x_j}{\rho k}\right) \\ &\leq \frac{\rho}{k} \prod_{j \neq i} \exp(-x_j/\rho k) = \frac{\rho}{k} \exp(-m/\rho k). \end{aligned}$$

Thus any point $p \in P_i$ that has $m \geq \rho k \ln(\rho/\varepsilon)$ points in $\mathcal{P} \setminus \{P_i\}$ closer to q than itself, has probability at most ε/k of being the closest point to q . Since each $P_i \in \mathcal{P}$ consists of at most k points, the combined effect of all of these far points cannot contribute more than ε to the total probability that P_i is the nearest neighbor. Also $k-1$ points from P_i may also be closer to q than p . Thus if p is not an $m(\rho, \varepsilon)$ -nearest neighbor of q in S , i.e., $p \notin \overline{P}_i$, then $\pi_p(q) < \varepsilon/k$. Hence,

$$\pi_i(q) \leq \sum_{p \in \overline{P}_i} \pi_p(q) + \varepsilon = \sum_{p \in \overline{P}_i} \widehat{\pi}_p(q) + \varepsilon = \widehat{\pi}_i(q) + \varepsilon.$$

This completes the proof of the lemma. □

For any i , if $P_i \cap \overline{S}(q) = \emptyset$, then we can implicitly set $\widehat{\pi}_i(q)$ to 0. Finally, the following result shows that the m nearest neighbors of q in S can be chosen efficiently in \mathbb{R}^2 .

Lemma 4.4.6 (Afshani and Chan [2]). *Given a set of N points in \mathbb{R}^2 , with $O(N \log N)$ expected preprocessing time and $O(N)$ space, the m nearest neighbors of a query point can be reported in $O(m + \log N)$ time.*

We thus obtain the following result.

Theorem 4.4.7. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 , ρ be the spread of its location probabilities as in Eq. (4.9), and $\varepsilon \in (0, 1)$ be a parameter. \mathcal{P} can be preprocessed in $O(nk \log(nk))$ expected time into a data structure of size $O(nk)$, so that for a query point $q \in \mathbb{R}^2$ and a parameter $\varepsilon \in (0, 1)$, it can compute, in time $O(\rho k \log(\rho/\varepsilon) + \log(nk))$, values $\hat{\pi}_i(q)$ for all $P_i \in \mathcal{P}$ such that $|\pi_i(q) - \hat{\pi}_i(q)| \leq \varepsilon$ for all $i \in \{1, \dots, n\}$.*

Remarks. (i) Unfortunately, this approach is not efficient when the spread of location probabilities is unbounded. In this case, one may have to retrieve $\Omega(n)$ points. Another approach may be to ignore points with weight smaller than ε/k , since even k such weights from a single uncertain point P_i cannot contribute more than ε to $\pi_i(q)$. However, the union of all such points may distort other probabilities.

Consider the following example. Let $p_1 \in P_1 \in \mathcal{P}$ be the closest point to the query point q . Let $w(p_1) = 3\varepsilon$. Let the next $n/2$ closest points $p_3, \dots, p_{n/2+2}$ be from different uncertain points $P_3, \dots, P_{n/2+2}$ and each have weights $w(p) = 2/(n+2) \ll \varepsilon/k$. Let the next closest point $p_2 \in P_2 \in \mathcal{P}$ have weight $w(p_2) = 5\varepsilon$. With probability $\pi_{p_1}(q) = 3\varepsilon$ the nearest neighbor is p_1 . The probability that p_2 is the nearest neighbor is $\pi_{p_2}(q) = (5\varepsilon)(1 - 3\varepsilon)(1 - 2/n)^{n/2} < (5\varepsilon)(1 - 3\varepsilon)(1/e) < 2\varepsilon$. Thus p_1 is more likely to be the nearest neighbor than p_2 . However, if we ignore points $p_3, \dots, p_{n/2+2}$ because they have small weights, then we calculate p_2 has probability $\hat{\pi}_{p_2}(q) = (1 - 3\varepsilon)(5\varepsilon) > 4\varepsilon$ for being the nearest neighbor. So $\pi_2(q)$ will be off by more than 2ε and it would incorrectly appear that p_2 is more likely to be the nearest neighbor than p_1 .

(ii) Though Lemma 4.4.6 is optimal theoretically, it is too complex to be implemented. Instead, one may use order- m Voronoi diagram to retrieve the m closest points (in unsorted order) to q . This would yield a data structure with $O(m(nk - m))$ space and $O(m(nk - m) \log(nk) + nk \log^3(nk))$ expected preprocessing time [8], while preserving the query time $(\log(nk) + m)$, where $m = O(\rho k \log(\rho/\varepsilon))$. Alternatively, one may use quad-trees and a branch-and-bound algorithm to retrieve m points of S closest to q [70].

4.5. Experimental Results

We have conducted experiments on synthetic datasets to demonstrate the efficacy of our methods for estimating qualification probabilities.

Experimental setup. We assume each uncertain point has a discrete distribution of size s . We set $r = \frac{c}{\sqrt{n}}$, where $c > 0$ is a parameter. The value of c indicates the level of uncertainty: the bigger value c is, the larger uncertainty each uncertain point has. We synthetically generated n uncertain points in two steps as follows: (1) For each uncertain point, we first generate a disk of radius r whose center is randomly chosen inside the unit square $[0, 1]^2$. (2) We then choose s possible locations within the disk of each uncertain point. We chose s possible locations uniformly inside the disk (we also tried Gaussian distribution and got similar results). In our experiments, we set $n = 1000$, $s = 5$, and $c \in \{0.5, 1.0\}$.

Measuring the effectiveness. We test how effective the Monte Carlo method and the spiral-search methods are in computing the most likely nearest neighbor, NN_L , and the estimates of qualification probabilities. In the experiments, 1000 queries were issued for each input, and we measured the following three quantities:

- (i) The percentage of trials in which the algorithm returns the correct NN_L .
- (ii) The error in estimated qualification probability of NN_L . Specifically for each

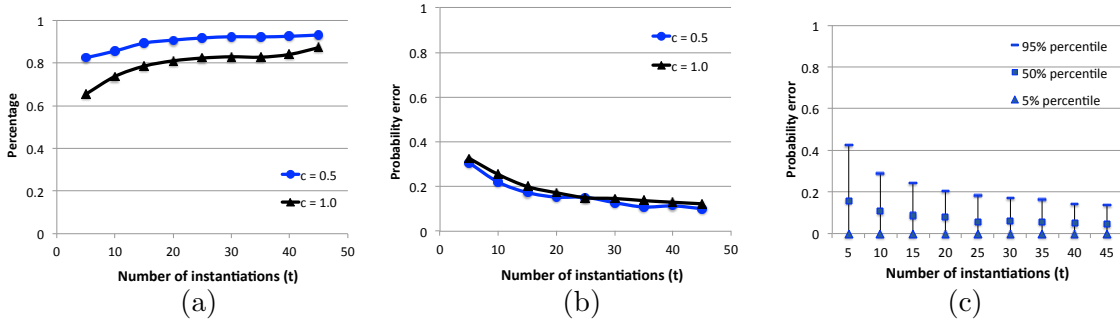


FIGURE 4.10. Monte Carlo method: (a) percentage of NN_L ; (b) probability error of NN_L ; (c) probability error of all points.

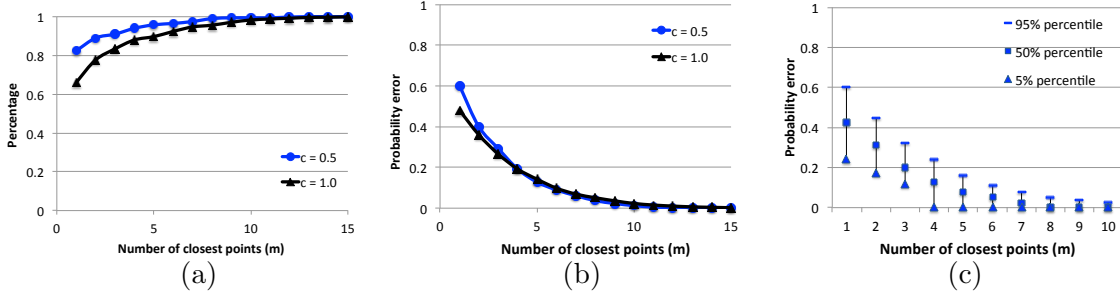


FIGURE 4.11. Spiral search method: (a) percentage of NN_L ; (b) probability error of NN_L ; (c) probability error of all points.

query, suppose P_i is the true NN_L , then the NN_L probability error is $|\pi_i - \hat{\pi}_i|$; we report the 90% percentile of these errors.

- (iii) For each query q , we compute $\max_i |\pi_i(q) - \hat{\pi}_i(q)|$, the maximum error in probability. Among all 1000 trials, we report the 5%, 50%, and 95% percentiles of these errors. We only used $c = 0.5$ for quantity (iii).

Monte Carlo method. We tested how quantities (i)-(iii) changed as we varied t , the number of instantiations t . Fig. 4.10(a)–(c) show these quantities as t varies from 5 to 45. Not surprisingly, as t increases, the percentage of correct NN_L increases, and the probability errors decrease. The smaller uncertainty (as denoted by c) we have, the better performance. For both $c = 0.5$ and $c = 1.0$, the NN_L is returned correctly at least 80% of the times if $t \geq 20$, and this also generally provides probability error less than 0.15.

Spiral search method. We also tested how (i) – (iii) changed as we varied m , the number of closest points retrieved to estimate the qualification probabilities. Fig. 4.11(a)–(c) show these quantities as m varies from 1 to 15 (or 10 in Fig. 4.11(c)).

Compared to the Monte Carlo approach, the spiral search method accuracy seems to converge much faster (although t versus m is not directly comparable). After only $m = 9$ closest points are retrieved, the NN_L is found more than 95% of the time, and the probability error goes to practically 0. Recall $s = 5$ so from these experiments it appears retrieving only $2s$ points to be effective. This method also seems less affected by the scale of uncertainty (parameter c). Since many practical k -nearest-neighbor algorithms exist, we believe this has the potential for practical use.

4.6. Conclusion

In this chapter, we investigated NN queries in a probabilistic framework in which the location of each input point is specified as a probability distribution function. We presented efficient methods for returning all the non-zero probability points, estimating the quantification probabilities and using it for threshold NN queries. We also conducted some preliminary experiments to demonstrate the effectiveness of our methods. We conclude by mentioning two open problems:

- (i) Analogously to the probabilistic Voronoi diagram defined in Section 4.4, one can define the most likely Voronoi diagram as a planar subdivision in which all points in a cell have the same most likely NN. What is the complexity of the most likely Voronoi diagram? The bound proved in Lemma 5.4.1 is an obvious upper bound but no matching lower bound is known.
- (ii) Can we extend the spiral search method to continuous distributions (at least for some simple, well-behaved distributions such as Gaussian), so that the query time is always sublinear?

Probabilistic Convex Hull

5.1. Introduction

In this chapter, we study the convex-hull problem, which asks for the smallest convex set that contains a given point set, in a probabilistic setting.

Uncertainty models. We focus on two models of uncertainty: unipoint and multipoint. In the *unipoint model*, each input point has a fixed location but it only exists probabilistically. Specifically, the input \mathcal{P} is a set of pairs $\{(p_1, \gamma_1), \dots, (p_n, \gamma_n)\}$ where each p_i is a point in \mathbb{R}^d and each γ_i is a real number in the range $(0, 1]$ denoting the probability of p_i 's existence. The existence probabilities of different points are independent; $P = \{p_1, \dots, p_n\}$ denotes the set of sites in \mathcal{P} .

In the *multipoint model*, each point probabilistically exists at one of multiple possible sites. Specifically, \mathcal{P} is a set of pairs $\{(P_1, \Gamma_1), \dots, (P_m, \Gamma_m)\}$ where each P_i is a set of n_i points and each Γ_i is a set of n_i real values in the range $(0, 1]$. The set $P_i = \{p_i^1, \dots, p_i^{n_i}\}$ describes the possible sites for the i th point of \mathcal{P} and the set $\Gamma_i = \{\gamma_i^1, \dots, \gamma_i^{n_i}\}$ describes the associated probability distribution. The probabilities γ_i^j correspond to disjoint events and therefore sum to at most 1. By allowing the

sum to be less than one, this model also accounts for the possibility of the point not existing (i.e. the *null* location)—thus, the multipoint model generalizes the unipoint model. In the multipoint model, $P = \bigcup_{i=1}^m P_i$ refers to the set of all sites and $n = |P|$.

Our results. The main results of our chapter can be summarized as follows.

- (A) We show (in Section 5.2) that the membership probability of a query point $q \in \mathbb{R}^d$, namely, the probability of q being inside the convex hull of \mathcal{P} , can be computed in $O(n \log n)$ time for $d = 2$. For $d \geq 3$, assuming the input and the query point are in general position, the membership probability can be computed in $O(n^d)$ time. (This result is shown in Section 5.3.) The results hold for both unipoint and multipoint models.
- (B) Next we describe two algorithms (in Section 5.4) to preprocess \mathcal{P} into a data structure so that for a query point its membership probability in \mathcal{P} can be answered quickly. The first algorithm constructs a *probability map* $\mathbb{M}(\mathcal{P})$, a partition of \mathbb{R}^d into convex cells, so that all points in a single cell have the same membership probability. We show that $\mathbb{M}(\mathcal{P})$ has size $\Theta(n^{d^2})$, and for $d = 2$ it can be computed in optimal $O(n^4)$ time. The second one is a sampling-based Monte Carlo algorithm for constructing a near-linear-size data structure that can approximate the membership probability with high likelihood in sublinear time for any fixed dimension.
- (C) We show (in Section 5.5) a connection between the membership probability and the Tukey depth, which can be used to approximate cells of high membership probabilities. For $d = 2$, this relationship also leads to an efficient data structure.
- (D) Finally, we introduce the notion of β -*hull* (in Section 5.6) as another approximate representation for uncertain convex hulls in the multipoint model: for $\beta \in [0, 1]$, a convex set C is called β -*dense* for \mathcal{P} , if C contains at least β fraction of each uncertain point. The β -hull of \mathcal{P} is the intersection of all β -dense sets for \mathcal{P} . We

show that for $d = 2$, the β -hull of \mathcal{P} can be computed in $O(n \log^5 n)$ time.

Related work. There is extensive and ongoing research in the database community on uncertain data; see [51] for a survey. In the computational geometry community, the early work relied on deterministic models for uncertainty (see e.g. [58, 92]), but more recently probabilistic models of uncertainty, which are closer to the models used in statistics and machine learning, have been explored [1, 6, 7, 9, 79, 80, 104, 105, 116, 125]. The convex-hull problem over uncertain data has received some attention very recently. Suri *et al.* [116] showed that the problem of computing the most likely convex hull of a point set in the multipoint model is NP-hard. Even in the unipoint model, the problem is NP-hard for $d \geq 3$. They also presented an $O(n^3)$ -time algorithm for computing the most likely convex hull under the unipoint model in \mathbb{R}^2 . Zhao *et al.* [129] investigated the problem of computing the probability of each uncertain point lying on the convex hull, where they aimed to return the set of (uncertain) input points whose probabilities of being on the convex hull are at least some threshold. Jørgensen *et al.* [77] showed that the distribution of properties, such as areas or perimeters, of the convex hull of \mathcal{P} may have $\Omega(\prod_{i=1}^m n_i)$ complexity.

5.2. Membership Probability in the Plane

In this section, we describe how to compute the probability that a query point lies inside the convex hull of a given uncertain point set in the plane. For simplicity, we assume that the input is non-degenerate, meaning that all possible point sites, including the query point q , are in general position: no two sites have the same coordinate along any dimension and no three sites are collinear. We defer the discussion of how to handle such degeneracies to Section 5.2.3. We begin our discussion with the unipoint case.

5.2.1. The unipoint model

Let $\mathcal{P} = \{(p_1, \gamma_1), \dots, (p_n, \gamma_n)\}$ be a set of n uncertain points in \mathbb{R}^2 under the unipoint model. We let P denote the set of all sites of \mathcal{P} , namely, $\{p_1, \dots, p_n\}$. Let \mathbf{R} denote the outcome of the probabilistic experiment by choosing each p_i with probability γ_i ; \mathbf{R} is a random subset of P . For a subset $B \subseteq P$, let $\gamma(B) = \Pr[\mathbf{R} = B]$. Then

$$\gamma(B) = \prod_{p_i \in B} \gamma_i \times \prod_{p_i \notin B} \bar{\gamma}_i,$$

where $\bar{\gamma}_i$ is the complementary probability $(1 - \gamma_i)$. Given a query point q , we want to compute its membership probability, denoted by $\mu(q)$, the probability that q lies in the convex hull of \mathbf{R} . Let $\text{CH}(B)$ denote the convex hull of B . By definition, $\mu(q)$ can be written as

$$\mu(q) = \sum_{B \subseteq P \mid q \in \text{CH}(B)} \gamma(B), \quad (5.1)$$

which unfortunately involves an exponential number of terms (possible subsets B). Our polynomial-time scheme for computing $\mu(q)$ builds on the following simple observation: q is *outside* $\text{CH}(\mathbf{R})$ if and only if q is a vertex of the convex hull $\text{CH}(\mathbf{R} \cup \{q\})$. For ease of reference, let C denote $\text{CH}(\mathbf{R} \cup \{q\})$ and V denote the set of vertices of C . Then the probability we want is $\mu(q) = 1 - \Pr[q \in V]$.

If $\mathbf{R} = \emptyset$, then clearly $C = \{q\}$ and $q \in V$. Otherwise, $|V| \geq 2$ and $q \in V$ implies that q is an endpoint of exactly two edges on the boundary of C .¹ In this case, we define the first edge following q in the counter-clockwise order of C as the *witness edge* of $q \notin \text{CH}(\mathbf{R})$. (See Fig. 5.1 (A).) It is easy to see that $q \in V$ if and only if $\mathbf{R} = \emptyset$ or (exclusively) qp_i is a witness edge for some $1 \leq i \leq n$. Let

¹If B consists of a single site p_i , then C is the line segment qp_i . In this case, we consider the boundary of C to be a cycle formed by two edges: one going from q to p_i , and one going from p_i back to q .

$\pi_i(q) = \Pr[qp_i \text{ is a witness edge}]$. Thus,

$$1 - \mu(q) = \Pr[q \in V] = \Pr[\mathbf{R} = \emptyset] + \sum_{p_i \in P} \pi_i(q). \quad (5.2)$$

Let $G_i \subseteq P$ be the set of sites lying to the right of the oriented line ℓ_i , spanned by the vector $\overrightarrow{qp_i}$. We observe that qp_i is a witness edge if and only if (i) $p_i \in \mathbf{R}$ and (ii) $\mathbf{R} \cap G_i = \emptyset$. Therefore

$$\pi_i(q) = \gamma_i \cdot \prod_{p_j \in G_i} \overline{\gamma_j},$$

This expression can be computed in $O(n)$ time. Since $\Pr[\mathbf{R} = \emptyset]$ can be computed in linear time, $\mu(q)$ can be computed in $O(n^2)$ time. The computation time can be improved to $O(n \log n)$ as described below.

Improving the running time. The main idea is to compute the witness edge probabilities in radial order around q . We sort P in counter-clockwise order around q . Without loss of generality, assume that the circular sequence $\langle p_1, \dots, p_n \rangle$ is the resulting order. (See Fig. 5.1 (B).) We first compute $\pi_1(q)$ in $O(n)$ time. Next for $i > 1$, we compute $\pi_i(q)$ from $\pi_{i-1}(q)$ in $O(1)$ amortized time as follows: Let W_i denote the set of sites in the open wedge bounded by the rays emanating from q in directions $\overrightarrow{p_{i-1}q}$ and $\overrightarrow{p_iq}$. (See Fig. 5.1 (C).) Notice that $G_i = G_{i-1} \cup \{p_{i-1}\} \setminus W_i$. It follows that

$$\pi_i(q) = \frac{\gamma_i}{\gamma_{i-1}} \cdot \frac{\overline{\gamma_{i-1}}}{\prod_{p_j \in W_i} \overline{\gamma_j}} \cdot \pi_{i-1}(q).$$

The amortized cost of a single update is $O(1)$ because each site of P enters G_i at most twice. Finally, notice that we can easily keep track of the set W_i during our radial sweep, as changes to this set follow the same radial order.

Theorem 5.2.1. *Given a set of n uncertain points in \mathbb{R}^2 under the unipoint model, the membership probability of a query point q can be computed in $O(n \log n)$ time.*

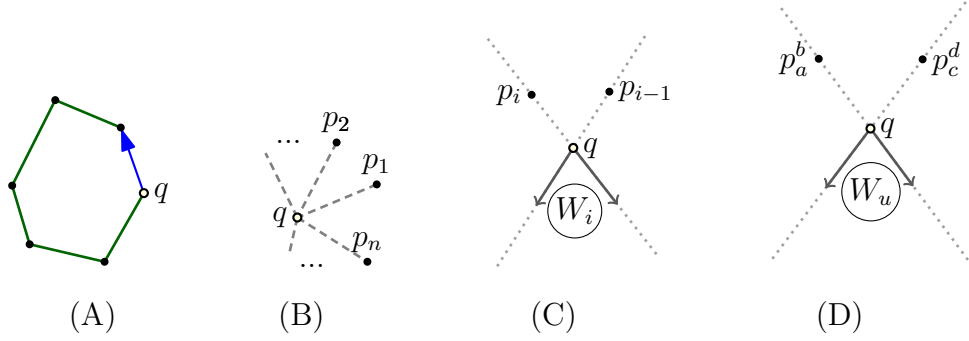


FIGURE 5.1. (A) A witness edge. (B) Sites in radial order around q . (C) The set W_i . (D) The set W_u .

5.2.2. The multipoint model

Let $\mathcal{P} = \{(P_1, \Gamma_1), \dots, (P_m, \Gamma_m)\}$ be a set of uncertain points in the multipoint model, as defined in the previous section. Recall that $P = \bigcup_{i=1}^m P_i$ is the set of all sites. Let \mathbf{R} be the outcome of the experimental outcome in which exactly one point of P_i is chosen randomly; p_i^j is chosen with probability γ_i^j .

For a subset $B \subseteq P$, let $\gamma(B) = \Pr[\mathbf{R} = B]$. As in the unipoint model, the definition of $\gamma(B)$ involves a product of existence probabilities for all sites in B . The sites that are not in B , however, contribute to $\gamma(B)$ in a different way. Specifically, let p_i^j be a site that is not in B . If B contains another site p_i^ℓ from the i th point, then the non-existence probability of p_i^j is irrelevant to $\gamma(B)$, because existence of p_i^ℓ already implies non-existence of p_i^j . If there is no such site p_i^ℓ , then no site from the tuple of the i th point is in B . In that case, we just consider the probability that i th point does not exist at all, which is $1 - \sum_{1 \leq j \leq |P_i|} \gamma_i^j$. Finally, notice that if B contains two sites from the same uncertain point, then it cannot be the outcome of an experiment.

For a point $q \in \mathbb{R}^2$, we again define $\mu(q) = \Pr[q \in \text{CH}(\mathbf{R})]$. We now describe how $\mu(q)$ is computed in the multipoint model. Let C and V be defined as above. As in the unipoint model, $q \in \text{CH}(\mathbf{R})$ if and only if $q \notin V$, thus $\mu(q) = 1 - \Pr[q \in V]$. Let $\pi_{ij}(q)$ denote the probability that qp_i^j is a witness edge. We follow a similar strategy

and decompose $\Pr[q \in V]$ as follows:

$$\Pr[q \in V] = \Pr[\mathbf{R} = \emptyset] + \sum_{\substack{1 \leq i \leq m \\ 1 \leq j \leq |P_i|}} \pi_{ij}(q).$$

The first term is trivial to compute in $O(n)$ time; $\pi_{ij}(q)$ is computed as follows. Let $G_{i,j}$ be the set of sites to the right of the oriented line spanning the vector $\overrightarrow{qp_i^j}$. As in the unipoint model, the segment qp_i^j is a witness edge if and only if $p_i^j \in \mathbf{R}$ and $\mathbf{R} \cap G_{i,j} = \emptyset$. Hence,

$$\begin{aligned} \pi_{ij}(q) &= \Pr[p_i^j \in \mathbf{R}] \times \Pr[\mathbf{R} \cap G_{i,j} = \emptyset \mid p_i^j \in \mathbf{R}] \\ &= \Pr[p_i^j \in \mathbf{R}] \times \prod_{1 \leq k \leq m} \Pr[\mathbf{R} \cap G_{i,j} \cap P_k = \emptyset \mid p_i^j \in \mathbf{R}] \\ &= \Pr[p_i^j \in \mathbf{R}] \times \prod_{\substack{1 \leq k \leq m \\ k \neq i}} \Pr[\mathbf{R} \cap P_k \cap G_{i,j} = \emptyset] \\ &= \gamma_i^j \times \prod_{\substack{1 \leq k \leq m \\ k \neq i}} \left(1 - \sum_{l \mid p_k^l \in G_{i,j}} \gamma_k^l \right). \end{aligned}$$

This expression can be easily computed in $O(n)$ time. It follows that one can compute $\Pr[q \in V]$, thus $\mu(q)$, in $O(n^2)$ time.

As before, the computation time can be improved to $O(n \log n)$ by computing the witness edge probabilities in a radial order around q . Let the circular sequence p'_1, p'_2, \dots, p'_n be the counter-clockwise order of all sites around q , where each p'_u is a distinct site p_a^b . We first compute the probability that qp'_1 is the witness edge in $O(n)$ time and also remember the values of the intermediate factors used in the computation. (The factors inside the $\prod_{1 \leq k \leq m}$ expression.) Then, for increasing values of u from 2 to n , we compute the probability that qp'_u is the witness edge by updating the probability for qp'_{u-1} . As a first step to this update, we update the values of the intermediate factors. To be more specific, let W_u denote the set of sites in the

open wedge bounded by the rays emanating from q in directions $\overrightarrow{p'_u q}$ and $\overrightarrow{p'_{u-1} q}$. Also, for simplicity, assume that $p'_u = p_a^b$ and $p'_{u-1} = p_c^d$. (See Fig. 5.1 (D).) Notice that $G_{a,b} = G_{c,d} \cup \{p_c^d\} \setminus W_u$. Then, for each site p_e^f in W_u , the e th factor increases by γ_e^f . Also, the c th factor decreases by γ_c^d . Finally, we temporarily set the value of the a th factor to 1 (to cover the case $k \neq i$ in the expression). Then, we can compute the witness edge probability for qp'_u by multiplying the probability of qp'_{u-1} with γ_a^b/γ_c^d and the multiplicative change in each intermediate factor. The cost of a single update is $O(1)$ amortized, as each site can contribute to at most 4 updates as in the unipoint case.

Theorem 5.2.2. *Given a set \mathcal{P} of uncertain points in the multipoint model with n sites in total and a point q in \mathbb{R}^2 , the probability of q being in the convex hull of \mathcal{P} can be computed in $O(n \log n)$ time using linear space.*

5.2.3. Dealing with degeneracies

In this section, we briefly explain how our algorithm for the planar case can be adapted to handle degeneracies. There are two main issues to be handled in the presence of degeneracies:

1. **A site may coincide with the query point q :** If this is the case, then the existence of such a site in R implies that $q \in \text{CH}(R)$. We separately compute the probability that q coincides with a site in R . The remaining portion of $\mu(q)$ is computed as before, but conditioned on the non-existence of all sites that coincide with q . To be precise, we again compute the probability $1 - \Pr[q \in V]$, but this time on the reduced set of sites that does not involve the sites coinciding q . (In the multipoint model, this also requires adjusting the probabilities of the sites which belong to the same uncertain point with another site coinciding q .) Once this probability is computed, we further multiply it with the probability

that no site coinciding q exists in R .

2. **q might be collinear with two or more other sites:** In this case, we need to re-define G_i in a more careful manner. As before, let ℓ_i be the oriented line spanned by the vector $\overrightarrow{qp_i}$. Let $G_i^1 \subseteq P$ be the set of sites lying to the right of or on ℓ_i . Let $G_i^2 \subseteq P$ be the set of sites lying on the segment qp_i . Then $G_i = G_i^1 - G_i^2$. A similar approach also applies to the multipoint model.

5.3. Membership Probability in Higher Dimensions

In this section, we describe our algorithm for computing the membership probability for dimensions higher than two. This algorithm works correctly only for non-degenerate input. To be precise, we require that all sites, including the query point q , are in general position in the following sense: for $2 \leq k \leq d$, the projection of no $k+1$ points of $P \cup \{q\}$ on a subspace spanned by any subset of k coordinates lies on a $(k-1)$ -hyperplane. We note that computing the membership probability for degenerate point sets in high dimensions is still an open problem. Our work can be considered a first step into understanding the complexity of the membership probability in high dimensions.

5.3.1. The unipoint model

The difficulty in extending the above to higher dimensions is an appropriate generalization of witness edges, which allow us to implicitly sum over exponentially many outcomes without over-counting. The following discussion explains the main idea we use for this generalization.

As in Section 5.2.1, let R denote the probabilistic outcome, $C = \text{CH}(R \cup \{q\})$ its convex hull, and V the vertices of C . Let $\lambda(R \cup \{q\})$ denote the point with the lowest x_d -coordinate in $R \cup \{q\}$. Clearly, if q is $\lambda(R \cup \{q\})$ then $q \in V$; otherwise, we

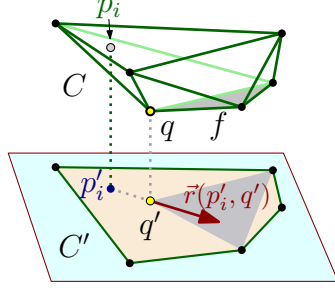


FIGURE 5.2. A three-dimensional example of an p_i -escaping face f for q .

condition the probability based on which point of R is $\lambda(R \cup \{q\})$. In particular, we have

$$\Pr[q \in V] = \Pr[q = \lambda(R \cup \{q\})] + \sum_{1 \leq i \leq n} \Pr[p_i = \lambda(R \cup \{q\}) \wedge q \in V].$$

It is easy to compute the first term. We show below how to compute each term of the summation in $O(n^{d-1})$ time, which gives the desired bound of $O(n^d)$.

Let p_i be an arbitrary point in R . We use p_i as a reference point known to be contained in the hull $C = \text{CH}(R \cup \{q\})$. Let R', p'_i and q' denote the projections of R , p_i and q respectively on the hyperplane $x_d = 0$, which we identify with \mathbb{R}^{d-1} . Let us define $C' = \text{CH}(R' \cup \{q'\}) \subset \mathbb{R}^{d-1}$, and let V' be the vertices of C' .

Let $\vec{r}(p'_i, q')$ denote the open ray emanating from q' in the direction of the vector $\overrightarrow{p'_i q'}$ (that is, this ray is moving “away” from p'_i). A facet f of C is a p_i -escaping facet for q if q is a vertex of f and the projection of f on \mathbb{R}^{d-1} intersects $\vec{r}(p'_i, q')$. See Fig. 5.2 for a three-dimensional example. The following lemma is key to our algorithm. The points of C projected into $\partial C'$ form the *silhouette* of C .

Lemma 5.3.1. (A) q has at most one p_i -escaping facet on C .

(B) The point q is a non-silhouette vertex of the convex-hull C if and only if q has a (single) p_i -escaping facet on C .

Proof. (A) If q has a p_i -escaping facet then it is a vertex of the convex-hull C . Consider the union of facets adjacent to q , and observe that the projection of this

“tent” can fold over itself in the projection only if q is on the silhouette. Specifically, if q is not on the silhouette then the claim immediately holds. Otherwise, q is on the silhouette, the open ray $\vec{r}(p'_i, q')$ does not intersect C' , and there are no p_i -escaping facets.

(B) The second claim follows immediately from the observation that the projected “tent” surrounds q' and as such one of the facets must be an escaping facets for p_i . \square

Given a subset of sites $P_\alpha \subseteq P \setminus \{p_i\}$ of size $(d - 1)$, define $f(P_\alpha)$ to be the $(d - 1)$ -dimensional simplex $\text{CH}(P_\alpha \cup \{q\})$. Since $p_i = \lambda(\mathbf{R} \cup \{q\})$ implies $p_i \in \mathbf{R}$, we can use Lemma 5.3.1 to decompose the i th term as follows:

$$\begin{aligned} \Pr\left[p_i = \lambda(\mathbf{R} \cup \{q\}) \wedge q \in V\right] &= \Pr\left[p_i = \lambda(\mathbf{R} \cup \{q\}) \wedge q' \in V'\right] \\ &+ \sum_{\substack{P_\alpha \subseteq P \setminus \{p_i\} \\ |P_\alpha| = (d-1) \\ f(P_\alpha) \text{ is } p_i\text{-escaping for } q}} \Pr\left[p_i = \lambda(\mathbf{R} \cup \{q\}) \wedge f(P_\alpha) \text{ is a facet of } C\right]. \end{aligned}$$

The first term is an instance of the same problem in $(d - 1)$ dimensions (for the point q' and the projection of P), and thus is computed recursively. For the second term, we compute the probability that $f(P_\alpha)$ is a facet of C as follows. Let $G_1 \subseteq P$ be the subset of sites which are on the other side of the hyperplane supporting $f(P_\alpha)$ with respect to p_i . Let $G_2 \subseteq P$ be the subset of sites that are below p_i along the x_d -axis. Clearly, $f(P_\alpha)$ is a facet of C (and $p_i = \lambda(\mathbf{R} \cup \{q\})$) if and only if all points in P_α and p_i exist in \mathbf{R} , and all points in $G_1 \cup G_2$ are absent from \mathbf{R} . The corresponding probability can be written as

$$\gamma_i \times \prod_{p_j \in P_\alpha} \gamma_j \times \prod_{p_j \in G_1 \cup G_2} \overline{\gamma_j}.$$

This formula is valid only if $P_\alpha \cap G_2 = \emptyset$ and p_i has a lower x_d -coordinate than q ; otherwise we set the probability to zero. This expression can be computed in linear time, and the whole summation term can be computed in $O(n^d)$ time. Then, by

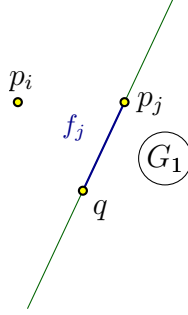


FIGURE 5.3. A facet f_j projected to the orthogonal complement plane.

induction, the computation of the i th term takes $O(n^d)$ time. Notice that the base case of our induction requires computing the probability $\Pr[p_i = \lambda(\mathbf{R} \cup \{q\}) \wedge q^{(d-2)} \in V^{(d-2)}]$ (where $^{(d-2)}$ indicates a projection to \mathbb{R}^2). Computing this probability is essentially a two-dimensional membership probability problem on q and P , but is conditioned on the existence of p_i and the non-existence of all sites below p_i along d th axis. Our two dimensional algorithm can be easily adapted to solve this variation in $O(n \log n)$ time as well.

Similar to the planar case, we can improve the computation time for the i th term to $O(n^{d-1})$ by considering the facets $f(P_\alpha)$ in radial order. In essence, the main idea is to fix $(d-2)$ sites of a given P_α and then change its $(d-1)$ th site in radial order around the fixed sites in order to obtain different P_α 's, such that the probability for the next P_α is easily computed using the probability for the previous P_α . In particular, let $L_\beta \subseteq P \setminus \{p_i\}$ be a subset of $(d-2)$ sites. Let f_j denote the $(d-1)$ -dimensional simplex $f(L_\beta \cup \{p_j\})$ where $p_j \notin L_\beta$ and $p_j \neq p_i$. (Note that $L_\beta \cup p_j$ is a possible P_α and $f_j = f(P_\alpha)$.) We can compute the probability that f_j is a facet of C for all facets f_j in constant amortized time as follows. We project all sites to the two-dimensional plane passing through q and orthogonal to the $(d-2)$ -dimensional hyperplane defined by $L_\beta \cup \{q\}$. (Such a plane is known as an orthogonal complement.) The hyperplane defined by $L_\beta \cup \{q\}$ projects onto q on this plane. Moreover, each facet f_j projects to a line segment extending from q . When we need to compute the probability that f_j is

a facet of C , the set G_1 includes the sites on the other side of the line supporting f_j 's projection with respect to p_i . (See Fig. 5.3.) We compute probabilities for the facets f_j based on their radial order around q . The probability for the next facet in the sweep can be computed by modifying the probability of the previous facet in constant amortized time as we have done for the planar case, as we can efficiently track how G_1 changes. It follows that the probability for all facets f_j (based on a single L_β) can be computed in $O(n)$ time. By iterating through all possible L_β , we can compute the probability for all facets $f(P_\alpha)$ in $O(n^{d-1})$ time. As a final note, we point out that the total cost of all sorting required for the radial sweeps is $O(n^{d-1} \log n)$ which is less than the overall cost of $O(n^d)$.

Theorem 5.3.2. *Let \mathcal{P} be a set of n uncertain points in the unipoint model in \mathbb{R}^d , and let q be a point in \mathbb{R}^d . If the input sites and q are in general position, then the probability of q being in the convex hull of \mathcal{P} can be computed in $O(n^d)$ time using linear space.*

5.3.2. The multipoint model

As in the planar case, the d -dimensional algorithm easily extends to the multipoint model. As before, we compute $\mu(q)$ by computing the probability $\Pr[q \in V]$. Following the earlier strategy, we decompose it as

$$\Pr[q \in V] = \Pr[q = \lambda(\mathbf{R})] + \sum_{1 \leq i \leq m} \left(\sum_{1 \leq j \leq |P_i|} \Pr[p_i^j = \lambda(\mathbf{R}) \wedge q \in V] \right).$$

It is trivial to compute the first term in $O(n)$ time. We now show how to compute each term inside the summations in $O(n^{d-1})$ time. This implies a total time of $O(n^d)$.

Clearly, Lemma 5.3.1 extends to the multipoint model, so we can use p_i^j -escaping facets to decompose our probability. Given a subset of sites $P_\alpha \subseteq P \setminus \{p_i^j\}$ of size

$(d - 1)$, define $f(P_\alpha)$ to be the $(d - 1)$ -dimensional simplex whose vertices are the points in P_α and q . Then,

$$\begin{aligned} \Pr\left[p_i^j = \lambda(\mathbf{R} \cup \{q\}) \wedge q \in V\right] &= \Pr\left[p_i^j = \lambda(\mathbf{R} \cup \{q\}) \wedge q' \in V'\right] \\ &+ \sum_{\substack{P_\alpha \subseteq P \setminus \{p_i^j\} \\ |P_\alpha| = (d-1) \\ f(P_\alpha) \text{ is } p_i^j\text{-escaping for } q}} \Pr\left[p_i^j = \lambda(\mathbf{R} \cup \{q\}) \wedge f(P_\alpha) \text{ is a facet of } C\right]. \end{aligned}$$

The first term is computed recursively. We compute each term of the summation as follows. Let I_α be the set of uncertain point indices of the sites in P_α , i.e., $I_\alpha = \{u \mid \exists v \cdot p_u^v \in P_\alpha\}$. As before, let $G_1 \subseteq P$ be the subset of sites which are on the other side of the hyperplane supporting $f(P_\alpha)$ with respect to p_i^j . Let $G_2 \subseteq P$ be the subset of sites that are below p_i^j along the x_d -axis. As done for the unipoint model, we write the desired probability as the probability that all points in P_α and p_i^j exist in \mathbf{R} , and all points in $G_1 \cup G_2$ are absent from \mathbf{R} . This probability is clearly zero, if any of the following conditions hold:

- $P_\alpha \cap G_2 \neq \emptyset$.
- p_i^j has a higher x_d -coordinate than q .
- P_α contains any two sites from the same uncertain point P_k .
- P_α contains any site from P_i .

Otherwise, we can write the probability as follows:

$$\begin{aligned} &\Pr\left[p_i^j \in \mathbf{R} \wedge P_\alpha \subseteq \mathbf{R} \wedge \mathbf{R} \cap (G_1 \cup G_2) = \emptyset\right] \\ &= \Pr\left[p_i^j \in \mathbf{R}\right] \times \Pr\left[P_\alpha \subseteq \mathbf{R} \mid p_i^j \in \mathbf{R}\right] \times \\ &\quad \Pr\left[\mathbf{R} \cap (G_1 \cup G_2) = \emptyset \mid p_i^j \in \mathbf{R} \wedge P_\alpha \subseteq \mathbf{R}\right] \\ &= \Pr\left[p_i^j \in \mathbf{R}\right] \times \Pr\left[P_\alpha \subseteq \mathbf{R}\right] \times \\ &\quad \Pr\left[\mathbf{R} \cap (G_1 \cup G_2) = \emptyset \mid p_i^j \in \mathbf{R} \wedge P_\alpha \subseteq \mathbf{R}\right] \end{aligned}$$

$$\begin{aligned}
&= \Pr[p_i^j \in \mathbf{R}] \times \Pr[P_\alpha \subseteq \mathbf{R}] \times \\
&\quad \prod_{\substack{1 \leq u \leq m \\ u \neq i \\ u \notin I_\alpha}} \left(\Pr[P_u \cap \mathbf{R} \cap (G_1 \cup G_2) = \emptyset] \right) \\
&= \gamma_i^j \times \prod_{u, v | p_u^v \in P_\alpha} \gamma_u^v \times \prod_{\substack{1 \leq u \leq m \\ u \neq i \\ u \notin I_\alpha}} \left(1 - \sum_{v | p_u^v \in (G_1 \cup G_2)} \gamma_u^v \right).
\end{aligned}$$

The expression takes linear time to compute and thus the summation term can be computed in $O(n^d)$ time. Then, by induction, the computation of the term for the site p_i^j takes $O(n^d)$ time. As before, we can improve this computation time to $O(n^{d-1})$ by considering the facets $f(P_\alpha)$ in radial order. This implies a total complexity of $O(n^d)$ for the algorithm.

Theorem 5.3.3. *Let \mathcal{P} be a set of n uncertain points in the multipoint model in \mathbb{R}^d and a point $q \in \mathbb{R}^d$ for $d \geq 3$, such that q and the sites of \mathcal{P} are in general position. The probability of q being in the convex hull of \mathcal{P} can be computed in $O(n^d)$ time.*

5.4. Membership Queries

In this section, we describe two algorithms – one deterministic and one Monte Carlo – for preprocessing a set of uncertain points for efficient membership-probability queries. We begin with the deterministic algorithm, which is based on a structure called the probability map.

5.4.1. Probability map

The *probability map* $\mathbb{M}(\mathcal{P})$ is the subdivision of \mathbb{R}^d into maximal connected regions so that the membership probability is the same for all query points in a region. The following lemma gives a tight bound on the size of $\mathbb{M}(\mathcal{P})$.

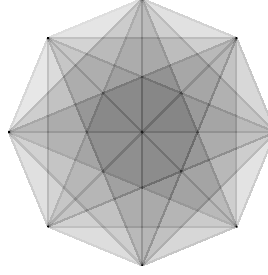


FIGURE 5.4. The probability map for a set of uncertain points arranged as vertices of a regular polygon.

Lemma 5.4.1. *The worst-case complexity of the probability map of a set of uncertain points in \mathbb{R}^d is $\Theta(n^{d^2})$, under both the unipoint and the multipoint model, where n is the total number of sites in the input.*

Proof. We prove the result for the unipoint model, as the extension to the multipoint model is straightforward. For the upper bound, consider the set H of $O(n^d)$ hyperplanes formed by all d -tuples of points in \mathcal{P} . In the arrangement $\mathcal{A}(H)$ formed by these planes, each (open) cell has the same value of $\mu(q)$. This arrangement, which is a refinement of $\mathbb{M}(\mathcal{P})$, has size $O((n^d)^d) = O(n^{d^2})$, establishing the upper bound.

For the lower bound, consider the problem in two dimensions; extension to higher dimensions is straightforward. We choose the sites to be the vertices p_1, \dots, p_n of a regular n -gon, where each site exists with probability w , $0 < w < 1$. See Fig. 5.4. Consider the arrangement \mathcal{A} formed by the line segments $p_i p_j$, $1 \leq i < j \leq n$, and treat each face as relatively open. If $\mu(f)$ denotes the membership probability for a face f of \mathcal{A} , then for any two faces f_1 and f_2 of \mathcal{A} , where f_1 bounds f_2 (i.e., $f_1 \subset \partial f_2$), we have $\mu(f_1) \geq \mu(f_2)$, and $\mu(f_1) > \mu(f_2)$ if $w < 1$. Thus, the size of the arrangement \mathcal{A} is also a lower bound on the complexity of $\mathbb{M}(\mathcal{P})$. This proves that the worst-case complexity of $\mathbb{M}(\mathcal{P})$ in \mathbb{R}^d is $\Theta(n^{d^2})$. \square

We now describe an algorithm for computing the probability map $\mathbb{M}(\mathcal{P})$. For simplicity, we describe the algorithm for the unipoint model, and then briefly explain

how to extend it to the multipoint model. Let H be the set of $O(n^2)$ lines passing through a pair of sites in \mathcal{P} , and let $\mathcal{A}(H)$ be the arrangement of H . $\mathcal{A}(H)$ contains $O(n^4)$ cells, edges and vertices. By Lemma 5.4.1, $\mathcal{A}(H)$ is a refinement of $\mathbb{M}(\mathcal{P})$. We first construct $\mathcal{A}(H)$ in $O(n^4)$ time, using an algorithm by Edelsbrunner *et al.* [57]. Since the membership probability of all points on an edge or cell is the same, let $\mu(f)$ denote this quantity for a vertex, edge, or cell f of $\mathcal{A}(H)$.

Next, we compute the membership probability of one of the cells in the arrangement, say C , in $O(n \log n)$ time (cf. Theorem 5.2.1). We compute the membership probabilities of the vertices, edges and cells neighboring C , in $O(1)$ time per each, by modifying $\mu(C)$.² We then apply the same process for all cells neighboring C . By repeatedly expanding into the neighboring cells, we can compute the probabilities for all of the arrangement in $O(n^4)$ time.

We now show how to compute $\mu(C')$ by using the already computed $\mu(C)$, where C' is one of the neighboring cells of C . We later explain how this algorithm can be adapted to compute the probability of neighboring edges and vertices.

Without loss of generality, assume that C and C' are separated by a vertical line ℓ passing through the sites p_i and p_j and C is to the left of C' . Notice that the common edge of C and C' , contained in ℓ , does not contain p_i or p_j . Now imagine that a point q moves through this boundary, crossing from C to C' . It is easy to see that the change in the membership probability of q is due to the changes in witness edge probabilities of the segments qp_i and qp_j , as other sites are irrelevant. Let $G_i(C)$ denote the set of sites lying to the right of the line $\overrightarrow{qp_i}$ for some $q \in C$. By construction, $G_i(C)$ is the same for all $q \in C$. Similarly we define $G_i(C')$, $\pi_i(C) = \Pr[qp_i \text{ is a witness edge} \mid q \in C]$ and $\pi_i(C') = \Pr[qp_i \text{ is a witness edge} \mid q \in C']$. We describe the change in the

²For ease of presentation, we assume that the arrangement is non-degenerate. It is straightforward to apply our technique on degenerate arrangements by using standard techniques (such as perturbation) to create a non-degenerate arrangement. We note that, even if we perturb the points to create a non-degenerate arrangement, we still use the old coordinates of the points and utilize the degeneracy handling rules of Section 5.2.3 while computing probabilities.

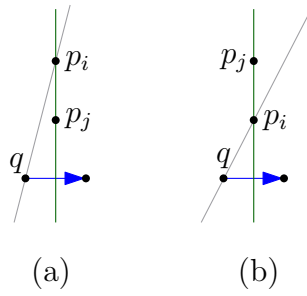


FIGURE 5.5. The cases to consider for computing the probability of C' from C .

witness edge probability of qp_i . The probability of qp_j changes analogously. The change in the probability of qp_i happens differently for two cases (See Fig. 5.5):

- (a) p_i is above p_j : In this case $G_i(C') = G_i(C) \setminus \{p_j\}$, therefore $\pi_i(C') = \pi_i(C)/\overline{\gamma}_j$.
- (b) p_i is below p_j : In this case $G_i(C') = G_i(C) \cup \{p_j\}$ and $p_j \notin G_i(C)$, therefore $\pi_i(C') = \pi_i(C) \cdot \overline{\gamma}_j$.

The changes clearly require constant time operations, and thus the membership probability of C' can be computed in $O(1)$ time.

We now describe how to compute $\mu(e)$ by using the already computed $\mu(C)$, where e is one of the bounding edges of C . Notice that any query point q on an edge e is degenerate with respect to the uncertain points. Therefore, we have to make use of the degeneracy handling rules from Section 5.2.3. Without loss of generality, assume that e is on a vertical line passing through the sites p_i and p_j and C is to the left of e . Notice that e is only a segment of the vertical line and does not contain p_i or p_j . Now imagine that a point q moves from C onto e . Again, the change in the membership probability of q is due to the changes in witness edge probabilities of the segments qp_i and qp_j . Similarly to $G_i(C)$ and $\pi_i(C)$, we define $G_i(e)$, and $\pi_i(e) = \Pr[qp_i \text{ is a witness edge} \mid q \in e]$. We describe the change in the witness edge probability of qp_i , the change for qp_j is analogous. We consider six different cases based on the vertical order of the points q, p_i and p_j (See Fig. 5.6):

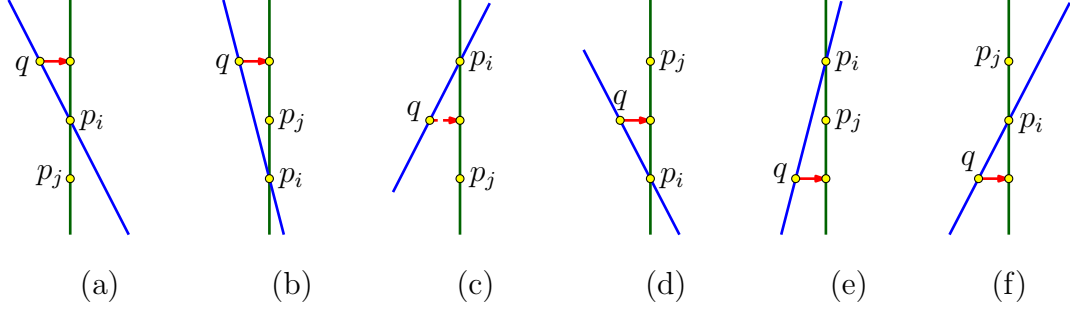


FIGURE 5.6. The cases to consider for computing the probability of e from C .

- (a) **Order** q, p_i, p_j : In this case $G_i(e) = G_i(C)$, therefore $\pi_i(e) = \pi_i(C)$.
- (b) **Order** q, p_j, p_i : In this case $G_i(e) = G_i(C)$, therefore $\pi_i(e) = \pi_i(C)$.
- (c) **Order** p_i, q, p_j : In this case $G_i(e) = G_i(C)$, therefore $\pi_i(e) = \pi_i(C)$.
- (d) **Order** p_j, q, p_i : In this case $G_i(e) = G_i(C) \cup \{p_j\}$ and $p_j \notin G_i(C)$, therefore $\pi_i(e) = \pi_i(C) \cdot \overline{\gamma}_j$.
- (e) **Order** p_i, p_j, q : In this case $G_i(e) = G_i(C) \setminus \{p_j\}$, therefore $\pi_i(e) = \pi_i(C) / \overline{\gamma}_j$.
- (f) **Order** p_j, p_i, q : In this case $G_i(e) = G_i(C) \cup \{p_j\}$ and $p_j \notin G_i(C)$, therefore $\pi_i(e) = \pi_i(C) \cdot \overline{\gamma}_j$.

For all cases, the change to the witness edge probability is easily computed in $O(1)$ time.

Finally, we explain how to compute the probability of the vertices. Let v be a vertex of $\mathcal{A}(H)$, which is the common endpoint of two edges e_1 and e_2 of a cell C . Then $\mu(v)$ is computed by applying the same probability changes that is applied for e_1 and e_2 . In other words,

$$\mu(v) = \frac{\mu(e_1) \cdot \mu(e_2)}{\mu(C)}.$$

The only exception to this is when v coincides a site p_i . In that case, we compute the membership probability of v from scratch in $O(n \log n)$ time. Since the number of such vertices is linear, it does not increase our overall cost of $O(n^4)$.

The extension of our technique to the multipoint model is straightforward. The

only major difference is that we need to remember (similar to what is done in Section 5.2.2) the intermediate factors when computing cell probabilities, as updating the witness edge probabilities requires updating these factors first. The total cost of a single update remains $O(1)$ because it requires updating one intermediate factor of two witness edge probabilities.

Theorem 5.4.2. *Given a set \mathcal{P} of uncertain points in the plane with n sites in total, the probability map $\mathbb{M}(\mathcal{P})$ can be computed in $O(n^4)$ time.*

Once $\mathbb{M}(\mathcal{P})$ is computed, it can be preprocessed in $O(n^4)$ time into a data structure of size $O(n^4)$ so that the vertex, edge, or face of $\mathbb{M}(\mathcal{P})$ containing a query point can be found in $O(\log n)$ time. We thus conclude the following.

Theorem 5.4.3. *Let \mathcal{P} be a set of uncertain points in \mathbb{R}^2 , with a total of n sites. \mathcal{P} can be preprocessed in $O(n^4)$ time into a data structure of size $O(n^4)$ so that for any point $q \in \mathbb{R}^d$, $\mu(q)$ can be computed in $O(\log n)$ time.*

Remark. For $d \geq 3$, due to our general position assumption, we can compute the membership probability only for d -faces of $\mathbb{M}(\mathcal{P})$, and not for the lower-dimensional faces. In that case, by utilizing a point-location technique in [38], one can build a structure that can report the membership probability of a query point (inside a d -face) in $O(\log n)$ time, with a preprocessing cost of $O(n^{d^2+d})$.

5.4.2. A Monte-Carlo algorithm

The size of the probability map may be prohibitive even for $d = 2$, so we describe a simple, space-efficient Monte Carlo approach for quickly approximating the membership probability, within absolute error. Fix a parameter $s > 1$, to be specified later. The preprocessing consists of s rounds, where the algorithm creates an outcome A_j of \mathcal{P} in each round j . Each A_j is preprocessed into a data structure so that for a query point $q \in \mathbb{R}^d$, we can determine whether $q \in \text{CH}(A_j)$.

For $d \leq 3$, we can build each $\text{CH}(A_j)$ explicitly and use linear-size point-location structures with $O(\log n)$ query time. For $d = 2$, we also apply fractional cascading to $\text{CH}(A_j)$, for $1 \leq j \leq s$, so that for a point $q \in \mathbb{R}^2$, all values of $j \leq s$ for which $q \in \text{CH}(A_j)$ can be reported in a total of $O(\log n + s)$ time. This leads to total preprocessing time $O(sn \log n)$ and space $O(sn)$. For $d \geq 4$, we use the data structure in [97] for determining whether $q \in \text{CH}(A_j)$, for all $1 \leq j \leq s$. For a parameter t such that $n \leq t \leq n^{\lfloor d/2 \rfloor}$ and for any constant $\sigma > 0$, using $O(st^{1+\sigma})$ space and preprocessing, it can compute in $O(\frac{sn}{t^{\lfloor d/2 \rfloor}} \log^{2d+1} n)$ time whether $q \in \text{CH}(A_j)$ for every j .

Given a query point $q \in \mathbb{R}^d$, we check whether $q \in \text{CH}(A_j)$, for every $j \leq s$. If q lies in k of them, we return $\hat{\mu}(q) = k/s$ as our estimate of $\mu(q)$. Thus, the query time is $O(\frac{sn}{t^{\lfloor d/2 \rfloor}} \log^{2d+1} n)$ for $d \geq 4$, $O(s \log n)$ for $d = 3$, and $O(\log n + s)$ for $d = 2$.

It remains to determine the value of s so that $|\mu(q) - \hat{\mu}(q)| \leq \varepsilon$ for all queries q , with probability at least $1 - \delta$. For a fixed q and outcome A_j , let X_i be the random indicator variable, which is 1 if $q \in \text{CH}(A_j)$ and 0 otherwise. Since $\mathbf{E}[X_i] = \mu(q)$ and $X_i \in \{0, 1\}$, using a Chernoff-Hoeffding bound [99] on

$$\hat{\mu}(q) = k/s = (1/s) \sum_i X_i,$$

we observe that

$$\Pr[|\hat{\mu}(q) - \mu(q)| \geq \varepsilon] \leq 2 \exp(-2\varepsilon^2 s) \leq \delta'.$$

By Lemma 5.4.1, we need to consider $O(n^{d^2})$ distinct queries. If we set $1/\delta' = O(n^{d^2}/\delta)$ and $s = O((1/\varepsilon^2) \log(n/\delta))$, we obtain the following theorem.

Theorem 5.4.4. *Let \mathcal{P} be a set of uncertain points in \mathbb{R}^d under the multipoint model with a total of n sites, and let $\varepsilon, \delta \in (0, 1)$ be parameters. \mathcal{P} can be preprocessed into a data structure so that with probability at least $1 - \delta$, for any query point $q \in \mathbb{R}^2$, a non-negative value $\hat{\mu}(q)$ satisfying $|\mu(q) - \hat{\mu}(q)| \leq \varepsilon$ can be returned.*

- For $d = 2$, the query time, the size, and the preprocessing time of the data structure are $O(\frac{1}{\varepsilon^2} \log \frac{n}{\delta})$, $O(\frac{n}{\varepsilon^2} \log \frac{n}{\delta})$, and $O(\frac{n}{\varepsilon^2} \log \log \frac{n}{\delta} \log n)$, respectively.
- For $d = 3$, the query time, the size, and the preprocessing time of the data structure are $O(\frac{1}{\varepsilon^2} \log(\frac{n}{\delta}) \log n)$, $O(\frac{n}{\varepsilon^2} \log \frac{n}{\delta})$, and $O(\frac{n}{\varepsilon^2} \log \log \frac{n}{\delta} \log n)$, respectively.
- For $d \geq 4$ and a parameter t such that $n \leq t \leq n^{\lfloor d/2 \rfloor}$, the query time, the size, and the preprocessing time of the data structure are $O(\frac{n}{t^{\lfloor d/2 \rfloor} \varepsilon^2} \log \frac{n}{\delta} \log^{2d+1} n)$, $O((t^{1+\sigma}/\varepsilon^2) \log \frac{n}{\delta})$, and $O((t^{1+\sigma}/\varepsilon^2) \log \frac{n}{\delta})$, respectively, for any constant $\sigma > 0$.

5.5. Tukey Depth and Convex Hull

The membership probability is neither a convex nor a continuous function, as suggested by the example in the proof of Lemma 5.4.1. In this section, we establish a helpful structural property of membership-probability function, intuitively showing that the probability stabilizes once we go deep enough into the “region”. Specifically, we show a connection between the Tukey depth of a point q with its membership probability; in two dimensions, this also results in an efficient data structure for approximating $\mu(q)$ quickly within a small absolute error.

Estimating $\mu(q)$. Let Q be a set of weighted points in \mathbb{R}^d . For a subset $A \subseteq Q$, let $w(A)$ be the total weight of points in A . Then the *Tukey depth* of a point $q \in \mathbb{R}^d$ with respect to Q , denoted by $\tau(q, Q)$, is $\min w(Q \cap H)$ where the minimum is taken over all halfspaces H that contain q .³ If Q is obvious from the context, we use $\tau(q)$ to denote $\tau(q, Q)$. Before bounding $\mu(q)$ in terms of $\tau(q, Q)$, we prove the following lemma.

Lemma 5.5.1. *Let Q be a finite set of points in \mathbb{R}^d . For any $p \in \mathbb{R}^d$, there is a set $\mathcal{S} = \{S_1, \dots, S_T\}$ of d -simplices formed by Q such that (i) each S_i contains p in its*

³If the points in Q are unweighted, then $\tau(q, Q)$ is simply the minimum number of points that lie in a closed halfspace that contains q .

interior; (ii) no pair of them shares a vertex; and (iii) $T \geq \lceil \tau(p, Q)/d \rceil$.

Proof. If $\tau(p, Q) > 0$, then $p \in \text{CH}(Q)$, and by Carathéodory Theorem [56], there is a d -simplex S with its $d + 1$ vertices in Q such that $p \in S$. Remove the vertices of S from Q , and repeat the argument. Let S_1, \dots, S_T be the resulting simplices. Observe that at most d vertices of S can be in an halfspace passing through p , which implies that the Tukey depth of p drops by at most d after each iteration of this algorithm. Hence $T \geq \lceil \tau(p, Q)/d \rceil$. \square

We now use Lemma 5.5.1 to bound $\mu(p)$ in terms of $\tau(p, P)$, but we first need a definition. Let X be a set of n points in \mathbb{R}^d . A subset $N \subseteq X$ is called an ε -net, for $\varepsilon \in [0, 1]$, if for every halfspace h with $|h \cap X| \geq \varepsilon n$, $N \cap h \neq \emptyset$. Haussler and Welzl [72] proved that a random subset of X of size $\frac{cd}{\varepsilon} \ln \frac{1}{\varepsilon\delta}$ is an ε -net with probability at least $1 - \delta$; here c is a constant.⁴ Their argument can be adapted to prove that if each point of X is chosen with probability $p(\varepsilon) \geq \frac{cd}{\varepsilon n} \ln \frac{1}{\varepsilon\delta}$, then the resulting subset is an ε -net with probability at least $1 - \delta$.

Theorem 5.5.2. *Let \mathcal{P} be a set of n uncertain points in the uniform unipoint model, that is, each point is chosen with the same probability $\gamma > 0$. Let P be the set of sites in \mathcal{P} . There is a constant $c > 0$ such that for any point $p \in \mathbb{R}^d$ with $\tau(p, P) \geq t$, we have $(1 - \gamma)^t \leq 1 - \mu(p) \leq d \exp\left(-\frac{\gamma t}{cd^2}\right)$.*

Proof. For the first inequality, fix a closed halfspace H that contains t points of P . If none of these t points is chosen then p does not appear in the convex hull of the outcome, so $1 - \mu(p) \geq (1 - \gamma)^t$.

Next, let p be a point with $\tau(p, P) \geq t$, \mathcal{S} the set of T simplices of Lemma 5.5.1, and V its set of vertices, where $T \geq \lceil t/d \rceil$. The uniform model chooses each point of V

⁴Haussler and Welzl [72] defined ε -net for general range spaces and proved the bound on the size of ε -nets for range spaces with finite VC-dimension but we need their result for this special case.

with probability γ . Let $R \subseteq V$ be the set of chosen points. If $\gamma = \frac{cd(d+1)}{|V|} \ln \frac{d}{\delta}$, then R is a $(\frac{1}{d+1})$ -net with probability at least $1 - \delta$. Since any halfspace H containing p contains at least one vertex of each simplex in \mathcal{S} , $|H \cap R| \geq \frac{|V|}{d+1}$. Therefore with probability at least $1 - \delta$, every halfspace containing p contains at least one point of R , implying that $p \in \text{CH}(R)$. Consequently, $\mu(p) \geq 1 - \delta$. Since $\gamma = \frac{cd(d+1)}{|V|} \ln \frac{d}{\delta} = \frac{cd}{T} \ln \frac{d}{\delta} \leq \frac{cd^2}{t} \ln \frac{d}{\delta}$, we have $\delta \leq d \exp(-\frac{\gamma t}{cd^2})$, as desired. \square

Data structure. Let \mathcal{P} be a set of uncertain points in the uniform unipoint model in \mathbb{R}^2 , i.e., each point appears with probability γ . Let P denote the set of all sites of \mathcal{P} . We now describe a data structure to estimate $\mu(q)$ for a query point $q \in \mathbb{R}^2$, within additive error $1/n^{c_0}$, for some constant c_0 . We fix a parameter $t_0 = \frac{c}{\gamma} \ln n$ for some constant $c > 0$. Let $\mathcal{T} = \{x \in \mathbb{R}^2 \mid \tau(x, \mathcal{P}) \geq t_0\}$ be the set of all points whose Tukey depth in \mathcal{P} is at least t_0 . If $t_0 \leq n/3$, $\mathcal{T} \neq \emptyset$ and \mathcal{T} is a convex polygon with $O(n)$ vertices [95]. We assume that $\gamma \geq \frac{3c}{n} \ln n$, so that $\mathcal{T} \neq \emptyset$.

By Theorem 5.5.2, $\mu(q) \geq 1 - 1/n^{c'}$ for all points $q \in \mathcal{T}$, where c' is a constant dependent on c . For a point $q \in \mathbb{R}^2$, let $\hat{\mu}(q) = \Pr[q \in \text{CH}(\mathcal{T} \cup R)]$, where R , as earlier, is the random subset of \mathcal{P} obtained by choosing each point of \mathcal{P} with probability γ . Note that $\hat{\mu}(q) \geq \mu(q)$. We describe a data structure for computing $\hat{\mu}(q)$ and argue that $\hat{\mu}(q) - \mu(q) \leq 1/n^{c'}$.

We construct \mathcal{T} and preprocess P for halfspace range reporting queries [41]. \mathcal{T} can be computed in time $O(n \log^3 n)$ [95], and constructing the half-plane range reporting data structure takes $O(n \log n)$ time [41]. So the total preprocessing time is $O(n \log^3 n)$, and the size of the data structure is linear.

A query is answered as follows. Given a query point $q \in \mathbb{R}^2$, we first test in $O(\log n)$ time whether $q \in \mathcal{T}$. If the answer is yes, we simply return 1 as $\mu(q)$. If not, we compute in $O(\log n)$ time the two tangents ℓ_1, ℓ_2 of \mathcal{T} from q . For $i = 1, 2$, let $\xi_i = \ell_i \cap \mathcal{T}$, and let h_{ℓ_i} be the *closed* half-plane bounded by ℓ_i that does not contain

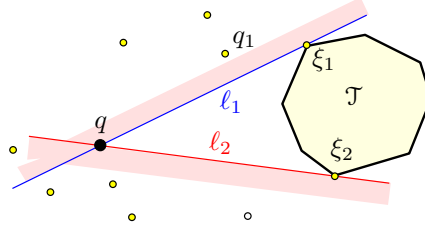


FIGURE 5.7. Illustration for the halfplanes h_{ℓ_1} and h_{ℓ_2} .

\mathcal{T} . Without loss of generality, assume that \mathcal{T} lies to the left (resp. right) of the vector $\overrightarrow{q\xi_1}$ (resp. $\overrightarrow{q\xi_2}$). (See Fig. 5.7.) Set $P_q = P \cap (h_{\ell_1} \cup h_{\ell_2})$ and $n_q = |P_q|$.

By querying the half-plane range reporting data structure with each of these two tangent lines, we compute the set P_q in time $O(\log n + n_q)$. Let $P_q^0 = P_q \cap (h_{\ell_1} \setminus h_{\ell_2})$. Note that for a point $p \in P \setminus P_q^0$, \overrightarrow{qp} cannot be a witness edge of $q \notin \text{CH}(\mathbb{R} \cup \mathcal{T})$. Adapting Eq. (5.2) in Section 5.2, we can write

$$1 - \hat{\mu}(q) = \sum_{p_i \in P_q^0} \hat{\pi}_i(q). \quad (5.3)$$

where $\hat{\pi}_i(q) = \Pr[qp_i \text{ is a witness edge}]$.

Note that for any $p_i \in P_q^0$, the points lying to the right of the line $\overrightarrow{qp_i}$ belong to P_q . Therefore $\hat{\pi}_i(q)$ can be computed by just considering the points of P_q . By sorting P_q around q and then performing an angular sweep, as in Section 5.2, $\hat{\pi}_i(q)$, for all $p_i \in P_q$, can be computed in a total of $O(n_q \log n_q)$ time. Hence, $\hat{\mu}(q)$ can be computed in $O(n_q \log n_q)$ time.

The correctness of the algorithm follows from the following lemma.

Lemma 5.5.3. *For any point $q \notin \mathcal{T}$, $|\hat{\mu}(q) - \mu(q)| \leq 1/n^{c-1}$, where c is the hiding constant of proportionality in the value of t_0 .*

Proof. We first note that $\pi_i(q) = \hat{\pi}_i(q)$ for all $p_i \in P_q^0$. Therefore Eq. (5.2) and Eq. (5.3) imply

$$\hat{\mu}(q) - \mu(q) = \sum_{p_i \notin P_q^0} \pi_i(q) = \sum_{p_i \notin P_q^0} \gamma(1 - \gamma)^{|G_i|},$$

where $G_i \subseteq P$, as defined in Section 5.2, is the set of points lying to the right of $\overrightarrow{qp_i}$.

For $p_i \notin P_q^0$, the halfplane lying to the right of the line $\overrightarrow{qp_i}$ intersects \mathcal{T} , therefore, by definition of \mathcal{T} , $|G_i| \geq t_0$. Hence,

$$\hat{\mu}(q) - \mu(q) \leq \sum_{p_i} \gamma(1 - \gamma)^{t_0} \leq n \cdot \exp(-\gamma t_0) \leq n \cdot \exp\left(-\gamma \frac{c}{\gamma} \ln n\right) = 1/n^{c-1}.$$

□

The efficiency of the algorithm follows from the following lemma.

Lemma 5.5.4. *For any point $q \notin \mathcal{T}$, $n_q < 4t_0 = O(\gamma^{-1} \log n)$.*

Proof. Fix any edge $e = (u, v)$ of \mathcal{T} . Let ℓ_e be the bounding line of e , and $h_{\ell_e}^-$ be the *open* half-plane bounded by ℓ_e that does not contain \mathcal{T} . By definition of \mathcal{T} , $|P \cap h_{\ell_e}^-| < t_0$.

Next we show that $|P \cap h_{\ell_1}| < 2t_0$. Let $e_1 = (\xi_1, u)$ and $e'_1 = (\xi_1, u')$ be the two edges adjacent to the vertex ξ_1 of \mathcal{T} . Then $|P \cap h_{\ell_{e_1}}^-| \leq t_0 - 1$ and $|P \cap h_{\ell_{e'_1}}^-| \leq t_0 - 1$.

Noticing that $P \cap h_{\ell_1} \subseteq P \cap (h_{\ell_{e_1}}^- \cup h_{\ell_{e'_1}}^- \cup \{\xi_1\})$, we have

$$|P \cap h_{\ell_1}| \leq \left| P \cap (h_{\ell_{e_1}}^- \cup h_{\ell_{e'_1}}^- \cup \{\xi_1\}) \right| \leq \left| P \cap h_{\ell_{e_1}}^- \right| + \left| P \cap h_{\ell_{e'_1}}^- \right| + 1 < 2t_0.$$

Similarly, $|P \cap h_{\ell_2}| < 2t_0$. The lemma now follows from that $n_q = |P \cap (h_{\ell_1} \cup h_{\ell_2})| \leq |P \cap h_{\ell_1}| + |P \cap h_{\ell_2}| < 4t_0$.

□

By Lemma 5.5.4, $n_q = O(\gamma^{-1} \log n)$, so the query takes $O(\gamma^{-1} \log(n) \log \log n)$ time. We thus obtain the following.

Theorem 5.5.5. *Let \mathcal{P} be a set of n uncertain points in \mathbb{R}^2 in the unipoint model, where each point appears with probability γ . Given a constant $c > 0$, \mathcal{P} can be preprocessed in $O(n \log^3 n)$ time into a linear-size data structure so that, for any*

point $q \in \mathbb{R}^2$, a value $\hat{\mu}(q)$ satisfying $|\hat{\mu}(q) - \mu(q)| \leq 1/n^c$ can be computed in $O(\gamma^{-1} \log(n) \log \log n)$ time, provided that $\gamma \geq \frac{c' \ln n}{n}$ where c' is a constant depending on c .

5.6. β -Hull

In this section, we consider the multipoint model, i.e., $\mathcal{P} = \{(P_1, \Gamma_1), \dots, (P_m, \Gamma_m)\}$. A convex set $C \subseteq \mathbb{R}^2$ is called β -dense with respect to \mathcal{P} if it contains β -fraction of each (P_i, Γ_i) , i.e., $\sum_{p_i^j \in C} \gamma_i^j \geq \beta$ for all $i \leq m$. The β -hull of \mathcal{P} , denoted by $\text{CH}_\beta(\mathcal{P})$, is the intersection of all convex β -dense sets with respect to \mathcal{P} . See Fig. 5.8(a) for an example. Note that for $m = 1$, $\text{CH}_\beta(\mathcal{P})$ is the set of points whose Tukey depth is at least $1 - \beta$. We first prove an $O(n)$ upper bound on the complexity of $\text{CH}_\beta(\mathcal{P})$ and then describe an algorithm for computing it.

Theorem 5.6.1. *Let $\mathcal{P} = \{(P_1, \Gamma_1), \dots, (P_m, \Gamma_m)\}$ be a set of m uncertain points in \mathbb{R}^2 under the multipoint model with $P = \bigcup_{i=1}^m P_i$ and $|P| = n$. For any $\beta \in [0, 1]$, $\text{CH}_\beta(\mathcal{P})$ has $O(n)$ vertices.*

Proof. We call a convex β -dense set C *minimal* if there is no convex β -dense set C' such that $C' \subset C$. A convex β -dense set C is minimal if $C = \text{CH}(P \cap C)$. Therefore C is a convex polygon whose vertices are a subset of P . Obviously $\text{CH}_\beta(\mathcal{P})$ is the intersection of minimal convex β -dense sets. Therefore each edge of $\text{CH}_\beta(\mathcal{P})$ lies on a line passing through a pair of points of P . Let L be the set of lines supporting the edges of $\text{CH}_\beta(\mathcal{P})$. We prove that $|L| \leq 2n$.

Fix a point $p \in P$. We claim that L contains at most two lines that pass through p . Indeed if $p \in \text{int}(\text{CH}_\beta(\mathcal{P}))$, then no line of L passes through p ; if $p \in \partial(\text{CH}_\beta(\mathcal{P}))$, then at most two lines of L pass through p ; and if $p \notin \text{CH}_\beta(\mathcal{P})$, then the only lines of L that can pass through p are the two tangents of $\text{CH}_\beta(\mathcal{P})$ from p . Hence at most two lines of L pass through p , as claimed. \square

Algorithm. We describe the algorithm for computing the upper boundary \mathcal{U} of $\text{CH}_\beta(\mathcal{P})$. The lower boundary of $\text{CH}_\beta(\mathcal{P})$ can be computed analogously. We call a line ℓ passing through a point $p \in P_i$ β -tangent of P_i at p if the *open* half-plane lying above ℓ contains less than β -fraction of points of P_i but the *closed* half-plane lying below ℓ contains at least β -fraction of points.

It will be easier to work in the dual plane. The dual of a point $p = (a, b)$ is the line $p^* : y = ax - b$, and the dual of a line $\ell : y = mx + c$ is the point $\ell^* = (m, -c)$. The point p lies above/below/on the line ℓ if and only if the dual point ℓ^* lies above/below/on the dual line p^* . Set $P_i^* = \{p_i^{j*} \mid p_i^j \in P_i\}$ and $P^* = \bigcup_{i=1}^m P_i^*$. For a point $q \in \mathbb{R}^2$ and for $i \leq m$, let $\kappa(q, i) = \sum \gamma_i^j$, where the summation is taken over all points $p_i^j \in P_i$ such that q lies below the dual line p_i^{j*} . We define the β -level Λ_i of P_i^* to be the upper boundary of the region $\{q \in \mathbb{R}^2 \mid \kappa(q, i) \geq \beta\}$. Λ_i is an x -monotone polygonal chain composed of the edges of the arrangement $\mathcal{A}(P_i^*)$; the dual line of a point on Λ_i is a β -tangent line of P_i . Let Λ be the lower envelope of $\Lambda_1, \dots, \Lambda_m$. See Fig. 5.8(b).

Let ℓ be the line supporting an edge of \mathcal{U} . It can be proved that the dual point ℓ^* is a vertex of Λ : first, ℓ is a supporting line for some minimal convex β -dense set, hence ℓ is a β -tangent line for some P_i and contains at least β -fraction of each (P_i, Γ_i) , that is, $\ell^* \in \Lambda$; second, as in the proof of Theorem 5.6.1, ℓ passes through a pair of points of P , hence ℓ^* is an intersection vertex of two lines of P^* . Next, let q be a vertex of \mathcal{U} , then q cannot lie above any β -tangent line of any P_i (since for any β -tangent line ℓ_0 , there exists a convex β -dense set with ℓ_0 bounding its upper part, and \mathcal{U} has to lie no above ℓ_0), which implies that the dual line q^* passes through a pair of vertices of Λ and does not lie below any vertex of Λ . Hence, each vertex of \mathcal{U} corresponds to an edge of the upper boundary of the convex hull of Λ . See Fig. 5.8(c). This observation suggests that \mathcal{U}^* , the dual of \mathcal{U} , can be computed by adapting an

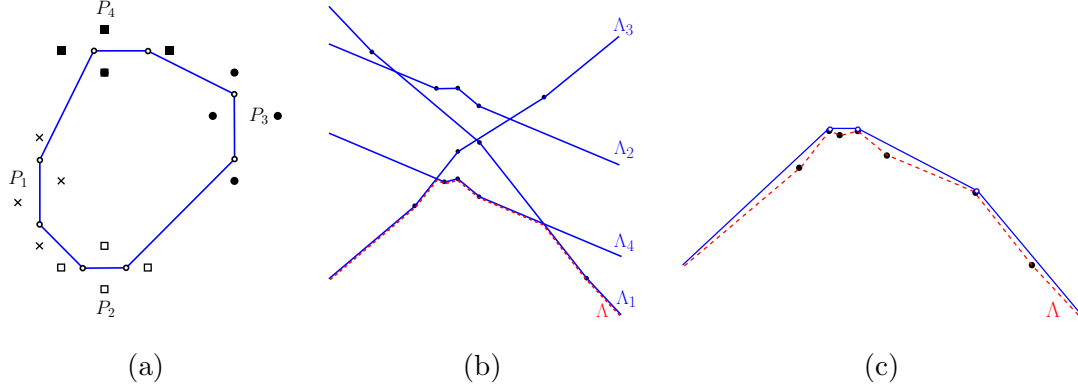


FIGURE 5.8. (a) An example of β -hull. \mathcal{P} has 4 uncertain points (marked with different shapes); each uncertain point has 4 possible locations with probability 0.25 each. Here $\beta = 0.75$. (b) The β -levels Λ_i (blue solid) of P_i^* , and their lower envelope Λ (red dashed). (c) The upper boundary (blue solid) of the convex hull of Λ (red dashed).

algorithm for computing the convex hull of a level in an arrangement of lines [16, 95]. We begin by describing a simple procedure, which will be used as a subroutine in the overall algorithm.

Lemma 5.6.2. *Given a line ℓ , the intersection points of ℓ and Λ can be computed in $O(n \log n)$ time.*

Proof. We sort the intersections of the lines of P^* with ℓ in $O(n \log n)$ time. Let $\langle q_1, \dots, q_u \rangle$, $u \leq n$, be the sequence of these intersection points. For every $i \leq m$, $\kappa(q_1, i)$ can be computed in a total of $O(n)$ time. Given $\{\kappa(q_{j-1}, i) \mid 1 \leq i \leq m\}$, $\{\kappa(q_j, i) \mid 1 \leq i \leq m\}$ can be computed in $O(1)$ time because if q_j lies on a line of P_i^* , then only $\kappa(q_j, i)$ is different from $\kappa(q_{j-1}, i)$. A point $q_j \in \Lambda$ if $q_j \in \Lambda_i$ for some i and lies below Λ'_i for all other i' . This completes the proof of the lemma. \square

The following two procedures can be developed by plugging Lemma 5.6.2 into the parametric-search technique [16, 95].

- (A) Given a point q , determine whether q lies below \mathcal{U}^* or return the tangent lines of \mathcal{U}^* from q .
- (B) Given a line ℓ , compute the edges of \mathcal{U}^* that intersect ℓ .

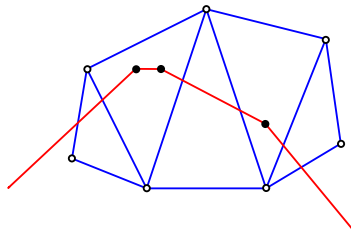


FIGURE 5.9. A $(1/r)$ -cutting Ξ with respect to the upper boundary of the convex hull of Λ .

Using Lemma 5.6.2 and the parametric search technique described in [16, Section 2], (A) can be performed in $O(n \log^2 n)$ time. Using (A) as a procedure and the parametric search technique described in [16, Section 2], (B) can be done in $O(n \log^4 n)$ time.⁵ Given (B), we can now compute \mathcal{U}^* as follows.

With the procedure (B) at hand, we are now ready to describe the algorithm. It is a recursive algorithm: each subproblem is defined with respect to a triangle Δ , and has as input a subset L_Δ of lines of P^* and a vector β_Δ , where $\partial\Delta$ is intersected by two disjoint edges e_1, e_2 of \mathcal{U}^* each of which has one vertex inside Δ , and L_Δ, β_Δ are defined recursively as follows.

Let v_1 (resp. v_2) be the vertex of e_1 (resp. e_2) that lies inside Δ , respectively. Let $L_{\Delta'}$ denote the set of lines from the previous recursive call. L_Δ is the set of lines ℓ in $L_{\Delta'}$ that cross Δ such that:⁶

- (i) ℓ crosses the segment v_1v_2 , or
- (ii) ℓ lies above the segment v_1v_2 and the slope of ℓ is between the slopes of e_1 and e_2 .

The vector $\beta_\Delta = \langle \beta_1^\Delta, \dots, \beta_m^\Delta \rangle$ indicates for β_i^Δ that Λ_i is the β_i^Δ -level of $P_i^* \cap L_\Delta$, and it can be obtained as follows. Let $\beta_{\Delta'} = \langle \beta_1^{\Delta'}, \dots, \beta_m^{\Delta'} \rangle$ denote the vector from the previous recursive call, and $\kappa(\Delta) = \sum \gamma_i^j$ where the summation is taken over

⁵In [16, Section 2], the running time for (B) was stated as $O(n \text{ polylog}(n))$ since the authors did not aim for the most efficient implementation.

⁶It can happen that $\partial\Delta$ is intersected by two disjoint edges e_1 and e_2 of \mathcal{U}^* each of which has one vertex inside Δ , and another two disjoint edges e_3 and e_4 of \mathcal{U}^* each of which has one vertex inside Δ . One can define L_Δ in this case similarly, and it does not affect our analysis.

all points $p_i^j \in P_i$ such that $p_i^{j*} \in L_{\Delta'}$ and Δ lies below p_i^{j*} . For $i \leq m$, we set $\beta_i^\Delta = \beta_i^{\Delta'} - \kappa(\Delta)$.

Next, we present the overall algorithm. In the initial call, we have as input $L = P^*$ and $\beta = \langle \beta, \dots, \beta \rangle$. Fix a parameter $r > 1$.⁷ We compute a $(1/r)$ -cutting $\Xi = \{\Delta_1, \dots, \Delta_u\}$ of L , where $u = O(r^2)$. For each $\Delta \in \Xi$, we do the following. By executing (B) with L and β , we compute the edges of \mathcal{U}^* that intersect $\partial\Delta$. We can then deduce whether Δ contains more than one vertex of \mathcal{U}^* . If $\partial\Delta$ is intersected by two disjoint edges e_1 and e_2 of \mathcal{U}^* each of which has one vertex inside Δ , then the answer is yes, and we solve the problem recursively in Δ with L_Δ and β_Δ as input, where L_Δ and β_Δ are defined previously. See Fig. 5.9.

Let $n_\Delta = |L_\Delta|$, for each $\Delta \in \Xi$. Since \mathcal{U}^* is convex, each line ℓ crosses \mathcal{U}^* at most twice for case (i), and each line ℓ satisfies case (ii) only once. Furthermore, if case (ii) holds for a triangle Δ , then case (i) does not hold for any triangle. Hence, $\sum_{\Delta \in \Xi} n_\Delta \leq 2n$, where $n = |L|$.

Let $T(n_\Delta, \mu_\Delta)$ denote an upper bound on the running time of the recursive call within Δ , where $n_\Delta = |L_\Delta|$, and μ_Δ is the number of vertices of \mathcal{U}^* lying inside Δ . The overall running time will be $T(n, \mu)$, where $\mu = O(n)$ is the complexity of \mathcal{U}^* . In the initial call, we assume $\mu > 1$, otherwise it is trivial. The following recurrence can be derived.

$$T(n, \mu) = \begin{cases} \sum_{\Delta \in \Xi} T(n_\Delta, \mu_\Delta) + O(n \log^4 n) & \text{if } \mu > 1, \\ O(n \log^4 n) & \text{if } \mu \leq 1. \end{cases}$$

where $n_\Delta \leq n/r$, $\sum_{\Delta \in \Xi} n_\Delta \leq 2n$ and $\sum_{\Delta \in \Xi} \mu_\Delta \leq \mu = O(n)$. The above recurrence solves to $T(n, \mu) = O(n \log^5 n)$. We conclude the following.

Theorem 5.6.3. *Given a set \mathcal{P} of uncertain points in \mathbb{R}^2 under the multipoint model with a total of n sites, and a parameter $\beta \in [0, 1]$, the β -hull of \mathcal{P} can be computed in*

⁷A $(1/r)$ -cutting of L is a triangulation Ξ of \mathbb{R}^2 such that each triangle of Ξ crosses at most n/r lines of L , where $n = |L|$.

$O(n \log^5 n)$ time.

Remarks. (1) The procedure (B) can be performed in $O(n \log^3 n)$ expected time by using randomized search (see e.g. [14]) instead of parametric search. Consequently, the β -hull of \mathcal{P} can be computed in $O(n \log^4 n)$ expected time.

(2) Let $k = \max_{1 \leq i \leq m} |P_i|$. Note that the β -level Λ_i has $O(k^2)$ complexity, and the lower envelope Λ of $\Lambda_1, \dots, \Lambda_m$ has $O(mk^2) = O(nk)$ complexity. Thus, the upper hull of Λ , hence \mathcal{U}^* and the β -hull of \mathcal{P} , can be computed in $O(nk \log n)$ time. This approach is more straightforward than using parametric search, and improves the running time for computing β -hull of \mathcal{P} in Theorem 5.6.3 if $k = O(\log^4 n)$.

5.7. Conclusion

In this chapter we studied the convex-hull problem in a probabilistic setting. We presented efficient algorithms for computing the probability of a point lying inside the convex hull of a set of uncertain points, and we also presented data structures for answering membership-probability queries. There are a few natural open problems:

- (i) Extend our membership-probability algorithm in high dimensions to handle degeneracies in the input.
- (ii) The size of the probability map is quite high. Is there a small-size approximate probability map? More precisely, given a parameter $\varepsilon > 0$, can we compute a small-size subdivision of \mathbb{R}^d and associate a number $\hat{\mu}_f$ with each cell of the subdivision so that for all points $q \in f$, $|\mu(q) - \hat{\mu}_f| \leq \varepsilon$. What is the size of such a subdivision and how quickly can it be computed?
- (iii) Can the data structure described in Section 5.5 be extended to higher dimensions?

Probabilistic Contour Tree

6.1. Introduction

In this chapter, we study contour trees of terrains in a probabilistic setting. Terrain is generally defined as the vertical and horizontal dimension of land surface, the understanding of which is important in many areas, including but not limited to, geographic information systems, agriculture, hydrology, and aviation. One commonly-studied type of terrain is represented as the graph of a piecewise-linear triangulated surface in \mathbb{R}^3 , known as *triangulated irregular network* (TIN). This is also the type of our interest in this chapter. Due to the inherent measurement errors, it is reasonable to assume that the height of each vertex of the underlying triangulation defining a terrain is described probabilistically.

Contour tree is a fundamental structure for topological analysis and data visualizations on large volume data sets, such as terrains and images. Efficient algorithms have been devised for computing contour trees of terrains in memory [33, 117, 119], I/O-efficiently [5], and for maintaining contour trees of dynamic terrains [4], where terrains are represented as TIN. When the height of each terrain vertex is repre-

sented probabilistically, a natural question raises: what is the contour tree of such a terrain? As there can be exponential number of contour tree instances, can we compute/estimate some statistics among these contour tree instances? For example, what is the probability of two points p, q lying on an edge of the contour tree? What is the expected distance of two points p, q on the contour tree, where the distance of p, q on a contour tree is defined to be the difference between the maximum height and the minimum height on the unique path from p to q on the contour tree, as defined in [24]? We look into some of the computational challenges related to contour trees of terrains imposed by the uncertainty on the vertex heights.

Terrains. We consider a terrain in \mathbb{R}^3 induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where V, E, F denote the set of vertices, edges, faces (triangles), respectively. Let $n = |V|$. Let $h : \mathbb{R}^2 \rightarrow \mathbb{R}$ be a piecewise-linear function defined on \mathbb{M} by first defining $h(v)$ for each $v \in V$, and then linearly interpolate each triangle $f \in F$. Alternately, h is characterized by the vector $\langle h(v_1), \dots, h(v_n) \rangle$. The graph of h is a terrain, which we denote by Σ . Σ is a piecewise-linear triangulated surface in \mathbb{R}^3 whose triangulation is induced by \mathbb{M} , i.e., if we project Σ onto \mathbb{R}^2 , we get exactly \mathbb{M} . Abusing the notations a little bit, we denote Σ by $\Sigma = (V, E, F)$ as well, and we use Σ and h interchangeably whenever the context is clear. We assume that $h(u) \neq h(v)$ for any two distinct vertices $u, v \in V$.

Critical points. For a vertex $v \in V$ of \mathbb{M} , the set of neighbors of v , denoted by N_v , is the set of vertices $u \in V$ such that u shares the same triangle with v . A higher (resp. lower) neighbor of v is a neighbor $u \in N_v$ such that $h(u) > h(v)$ (resp. $h(u) < h(v)$). v is called a local minimum (resp. local maximum) of \mathbb{M} if v has no lower neighbor (resp. higher neighbor). v is called a saddle vertex if there exist two higher neighbors and two lower neighbors which alternate around v .

More precisely, there exist $u_1, u_2, u_3, u_4 \in N_v$ in the counterclockwise order around v such that $h(u_1) > h(v)$, $h(u_2) < h(v)$, $h(u_3) > h(v)$ and $h(u_4) < h(v)$, or such that $h(u_1) < h(v)$, $h(u_2) > h(v)$, $h(u_3) < h(v)$ and $h(u_4) > h(v)$. All the local minimums, local maximums, and saddle vertices are called critical points.

Contours. Given any value $\ell \in \mathbb{R}$, the ℓ -level set of \mathbb{M} , denoted as \mathbb{M}_ℓ , is defined as $\mathbb{M}_\ell = \{x \in \mathbb{R}^2 \mid h(x) = \ell\}$. A connected component of \mathbb{M}_ℓ is called a contour. A local minimum or maximum v has only one contour, which only consists of the single point v . A saddle vertex v can have two or more contours, where v is the only point shared by these two or more contours.

Height level maps. Introduced by de Berg et al. [26], a *height level map* \mathbb{M}_h , for a given terrain Σ , is defined to be the planar map consisting of all the contours of all the critical points; see Fig. 6.1 for an example. Since for a given saddle vertex, the complexity of all its contours can be linear, and there can be linear number of saddle vertices, \mathbb{M}_h can have quadratic complexity in the worst case. See [26] for more details.

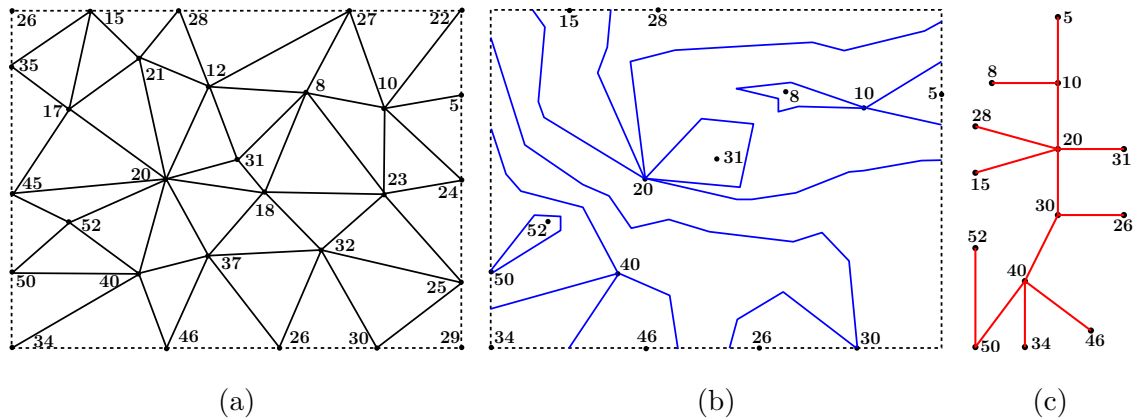


FIGURE 6.1. (a) A terrain Σ ; (b) its height level map \mathbb{M}_h ; (c) its contour tree \mathbb{T}_h .

Contour trees. As we change the value of ℓ from $-\infty$ to $+\infty$, contours can appear at local minimums, split/merge at saddle vertices, and disappear at local maximums. Contour tree was defined to track the topological changes of contours, where each node of a contour tree corresponds to a contour at a critical vertex, and an edge (v, w) represents the contour that appears at v and disappears at w . For a terrain Σ , we denote its contour tree by T_h . T_h can be defined directly from M_h , by taking all the critical points as nodes, and connecting two nodes u and v if and only if there is a region in M_h bounded by the contours of u and v . That is, an edge in T_h corresponds to a region in M_h . T_h has only linear size while M_h has quadratic size in the worst size. See Fig. 6.1. Note that here we are defining T_h based on M_h , as in [119]. One can also define T_h from the perspective of contour evolutions; see e.g. [4, 33].

Uncertainty. Here we assume at each vertex $v \in V$, $h(v)$ has k possible values, each with some probability being chosen. Formally, $h(v) \in \{h_v^1, \dots, h_v^k\}$ and $\Pr[h(v) = h_v^i] = w_v^i$, where $w_v^j \in [0, 1]$ and $\sum_{j=1}^k w_v^j = 1$, and we say that $h(v)$ has a discrete distribution of *description complexity* k . We assume that for any two distinct vertices u and v of \mathbb{M} , $h_u^i \neq h_v^j$, for any $1 \leq i, j \leq k$. We also consider the case where $h(v)$ is represented as a continuous distribution defined by a probability density function (pdf) $h(v) : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$. Examples of $h(v)$ include uniform distributions, Gaussian distributions, and histograms.

Our results. The main results of this chapter can be summarized as follows.

- (A) We show (in Section 6.2) that if the height of each terrain vertex is described by a discrete distribution of description complexity k , then after sorting the height values for each vertex v in total $O(nk \log k)$ time, the probability that v is a critical point can be computed in $O(kt_v^2)$ time, where t_v is the number of neighbors of v . Consequently, the expected number of nodes (resp. edges) of the

counter tree can be computed in additional $O(n)$ time.

- (B) We show (in Section 6.3) that the probability of two points p, q in \mathbb{R}^2 lying on an edge of the contour tree can be estimated within additive error with high probability using a near-linear-size data structure. The results hold both for discrete and continuous distributions to represent the height of each terrain vertex.
- (C) We define the distance of two points p, q on a contour tree to be the difference between the maximum height and the minimum height on the unique path from p to q on the contour tree, as in [24]. We show (in Section 6.4) that two distance statistics, the expected distance of p, q in \mathbb{R}^2 and the probability that the distance of p, q is at least ℓ on the contour tree, can be estimated within additive error and/or relative error with high probability using a near-linear-size data structure.

Related work. The contour-tree problem under uncertainty has received some attention recently, see e.g. [67, 84, 98]. Kraus [84] studied the visualization of uncertain contour trees, where he showed how to determine (by using grayscale morphology) and visually convey the uncertainty of the elements of a contour tree, and how to combine multiple contour trees of different versions of a data set in one visualization. Mihai and Westermann [98] studied the visualization of the stability of critical points in uncertain scalar fields, where they derived measures for the likelihood of a critical point occurring around a given location. Günther et al. [67] studied *mandatory critical points* of 2D uncertain scalar fields, where a mandatory critical point is represented by a *critical component* as well a *critical interval* such that any realization has at least one critical point of a given type present in the critical component and taking a value in the critical interval. The mandatory critical points can be interpreted as the

common topological denominator of all the realizations of the uncertain data.

Furthermore, there has been some work on terrain analysis in the presence of data uncertainty. The shortest-path problems have been studied on terrains in a probabilistic setting, termed as *uncertain terrains*, see e.g. [62, 63, 82]. Gray and Evans [63] showed that finding the *optimistic* shortest path on uncertain terrains is NP-hard, where for any vertex in the underlying triangulation defining an uncertain terrain, its height is represented as an interval and can take any value in the given interval, a path is characterized in \mathbb{R}^2 , its length with respect to a terrain instance is defined in \mathbb{R}^3 by lifting it onto the terrain instance, and its length with respect to an uncertain terrain is defined to be the *minimum* length of this path among all terrains instances of the uncertain terrain. Later, Gary [62] extended the hardness result to the *pessimistic*-shortest-path problem where the length of a path with respect to an uncertain terrain is the *maximum* length of this path among all terrain instances of the uncertain terrain. Furthermore, Kholondyrev and Evans [82] showed that if we can walk only on the terrain edges, i.e., the (lifted) path is restricted on the terrain edges, then finding the pessimistic shortest path on uncertain terrains remains NP-hard and there exists a fully-polynomial time approximation scheme for it, while the optimistic version is polynomial-time solvable.

6.2. Probability of a Vertex Being a Critical Point

In this section, we describe how to compute the probability that v is a critical point, which we denote by $\phi(v)$, and then show how to use $\phi(v)$ to compute the expected number of nodes (resp. edges) in the contour tree.

We assume that the height $h(v)$ of each vertex v is described by a discrete distribution of description complexity k . Let Σ denote the *random* terrain outcome of the probabilistic experiment, in which exactly one height value h_v^j of v is chosen — h_v^j is chosen randomly with probability w_v^j , and let \mathbf{H} denote the *random* height

function of Σ . Let Σ denote one possible outcome in which v has height value h_v^i , and $\gamma(\Sigma) = \gamma(h) = \Pr\{\mathbf{H} = h\}$, where h is the height function of Σ . Then

$$\gamma(\Sigma) = \gamma(h) = \prod_{v \in V} w_v^i.$$

By definition, $\phi(v)$ can be written as

$$\phi(v) = \sum_{v \text{ is critical in } \Sigma} \gamma(\Sigma).$$

Note that there are $O(k^n)$ terrain instances Σ , implying that the above expression involves an exponential number of terms. Nevertheless, we show that $\phi(v)$ can be computed in polynomial time. For simplicity, we compute its complement probability, $1 - \phi(v)$, the probability that v is not a critical point. v is called a *regular vertex* in such cases. It is easy to see that v is a regular vertex if and only if, in the counterclockwise order around v , all the neighbors of v form a non-empty sequence of higher neighbors followed by a non-empty sequence of lower neighbors, or vice versa. There are only $O(t_v^2)$ such cases, where t_v denotes the number of neighbors of v , and the probability of each case occurring can be computed in amortized $O(1)$ time, given a fixed value of v and sorted height values of each vertex. Therefore it takes $O(kt_v^2)$ time to compute $1 - \phi(v)$, hence $\phi(v)$. We conclude the following.

Lemma 6.2.1. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of each vertex v is described by a discrete distribution of description complexity k . After sorting the height values for each vertex $v \in V$ in total $O(nk \log k)$ time, the probability that v is a critical point can be computed in $O(kt_v^2)$ time, where t_v is the number of neighbors of v .*

Let $T_{\mathbf{H}}$ denote the *random* contour tree of the random height function \mathbf{H} (or the random outcome Σ). Next, we describe how to compute the expected number

of nodes (resp. edges) in the contour tree T_H , which we denote by $\mathsf{EN}(\mathsf{T}_H)$ (resp. $\mathsf{EE}(\mathsf{T}_H)$), where the expectation is taken over all possible contour tree instances T_h .

Since the number of edges is always one less than the number of nodes in a tree, we have that $\mathsf{EE}(\mathsf{T}_H) = \mathsf{EN}(\mathsf{T}_H) - 1$. Now we show how to evaluate $\mathsf{EN}(\mathsf{T}_H)$. Note that for an outcome Σ , a vertex v is a node in T_h if and only if v is a critical point in Σ . By linearity of expectation,

$$\mathsf{EN}(\mathsf{T}_H) = \sum_{v \in V} \phi(v).$$

Thus, we have the following corollary.

Corollary 6.2.2. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of each vertex v is described by a discrete distribution of description complexity k . The expected number of nodes (resp. edges) in its contour tree can be computed in $O(n)$ time given the probabilities of all the vertices in \mathbb{M} being critical points.*

Remarks. If the height $h(v)$ of each vertex v is described by a continuous distribution, computing $\phi(v)$ requires complex high-dimensional integration, which is often expensive. Instead, one may sample each continuous distribution first to estimate $\phi(v)$ within additive error in polynomial running time with high probability.

6.3. Probability of Two Points Lying on an Edge of the Contour Tree

In this section, we describe our algorithm for estimating the probability of any two points $p, q \in \mathbb{R}^2$ lying on an edge of the contour tree T_H , which we denote by $\pi(p, q)$, within additive error ε with probability at least $1 - \delta$, for any two given parameters $\varepsilon, \delta \in (0, 1)$. We start with the discrete case where the height $h(v)$ of each vertex v is described by a discrete distribution of description complexity k , then proceed with the continuous case.

Discrete case. We have $O(k^n)$ possible instances of terrains Σ , each of which defines a contour tree \mathbb{T}_h , resulting in $O(k^n)$ contour trees. $\pi(p, q)$ can be written as

$$\pi(p, q) = \sum_{p, q \text{ lies on an edge of } \mathbb{T}_h} \gamma(h).$$

Furthermore, we have the following lemma.

Lemma 6.3.1. *There exists a terrain in the probabilistic setting with at most two height values for each of its n vertices such that it has $\Omega(2^n)$ distinct contour trees.*

Proof. Assume $n = 9m^2$ for some positive integer m . Consider a grid G of size $\sqrt{n} \times \sqrt{n} = 3m \times 3m$ with n vertices. See Fig. 6.2. Note that G has m^2 pairwise-disjoint grids of size 3×3 . Let h^-, h^0, h^+ be three height values such that $h^- < h^0 < h^+$. For each such 3×3 grid, its center vertex has only one height value h^0 , and each of its other eight vertices has two possible height values h^-, h^+ , with probability 0.5 each. Finally, we triangulate each grid of size 1×1 arbitrarily, resulting in a triangulated grid \mathbb{M} . It is no hard to see that each center vertex can be a critical or non-critical vertex, and it is independent with other center vertices. Hence it has $\Omega(2^{m^2})$ distinct sets of critical vertices, resulting in $\Omega(2^{m^2})$ contour trees. Since $m^2 = n/9$, hence this construction gives a lower bound of $\Omega(2^n)$ on the number of contour trees. \square

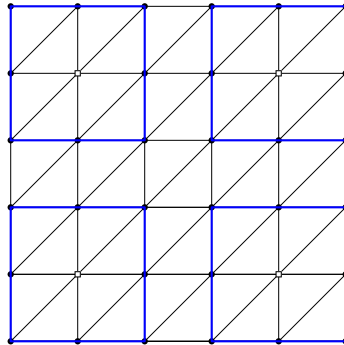


FIGURE 6.2. Lower bound construction with $n = 36$ vertices placed on a 6×6 grid. It has 4 pairwise-disjoint grids of size 3×3 .

Lemma 6.3.1 suggests that it is unlikely to avoid summing over exponential number of terms to compute $\pi(p, q)$ exactly. Therefore, we turn to approximate $\pi(p, q)$. To this end, we rely on the height level maps \mathbf{M}_h , defined in Section 6.1. We show that although there can be k^n height level maps \mathbf{M}_h , each of which has complexity $\Theta(n^2)$, the overlay of all k^n such height level maps, which we denote by $\widehat{\mathbf{M}}_h$, only has complexity $O(n^3k^8)$.

Lemma 6.3.2. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of each vertex v is described by a discrete distribution of description complexity k . $\widehat{\mathbf{M}}_h$ has complexity $O(n^3k^8)$, and it can be constructed in time proportionally to its size.*

Proof. Note that in the exact case, each triangle is crossed at most once by all the contours of a saddle vertex, i.e., each triangle contributes at most one edge to all the contours of a saddle vertex in \mathbf{M}_h . There are n possible saddle vertices, each taking k possible values, giving us nk possible saddle values. There are k^3 vertex value combinations for the three vertices of a triangle, hence there are totally $O(nk^4)$ edges inside each triangle. The arrangement of these $O(nk^4)$ edges has complexity $O(n^2k^8)$. Since we have n triangles, and edges are disjoint between any two triangles, therefore the complexity of the resulting planar map, which we denote by $\widetilde{\mathbf{M}}_h$, is $O(n^3k^8)$. Furthermore, it is no hard to see that $\widetilde{\mathbf{M}}_h$ is finer than $\widehat{\mathbf{M}}_h$, hence $\widehat{\mathbf{M}}_h$ has complexity $O(n^3k^8)$.

To construct $\widehat{\mathbf{M}}_h$, we need to remove redundant edges in $\widetilde{\mathbf{M}}_h$. First notice that each edge e in $\widetilde{\mathbf{M}}_h$ is determined by three vertices v_1, v_2, v_3 of a triangle in \mathbb{M} and a saddle value h_v of some vertex v in \mathbb{M} . We test whether e is valid in $\widehat{\mathbf{M}}_h$, by checking the following two conditions:

- (i) Whether it is possible for v to be a saddle vertex with saddle value h_v conditioning on the values that v_1, v_2, v_3 have taken. If v coincides with one of v_1, v_2, v_3 , say

v_1 , then v_1 must take the same value as v . It can be checked in constant time assuming v is adjacent to constant number of triangles (hence vertices) in \mathbb{M} . If e is not valid, we remove e from $\tilde{\mathbb{M}}_h$. We do this for every edge e in $\tilde{\mathbb{M}}_h$.

- (ii) Whether the edge e is able to reach v through a path of the same height as e . We can find all these edges in a BFS (breadth-first search) manner starting from v , and then eliminate those edges e that cannot reach v through a path of the same height as e . Those eliminated edges are components of the level sets of $h(v)$ but do not lie in the same contour as v .

Finally, for every vertex v in \mathbb{M} , we check whether it has a chance to be a local minimum/maximum. If it does, we add v back into $\tilde{\mathbb{M}}_h$ if v is removed in the phase of validating the edges in $\tilde{\mathbb{M}}_h$. It is no hard to argue that the resulting map is $\hat{\mathbb{M}}_h$.

Since validating all edges in $\tilde{\mathbb{M}}_h$ takes $O(n^3k^8)$ time, which dominates the process of validating all vertices in \mathbb{M} , we conclude that it takes $O(n^3k^8)$ time to construct $\hat{\mathbb{M}}_h$. The theorem thus follows. \square

It is now easy to derive the following lemmas.

Lemma 6.3.3. *For any two points p and q in the same region of $\hat{\mathbb{M}}_h$, $\pi(p, q) = 1$, i.e., p and q always lie on an edge of \mathbb{T}_H .*

Proof. Note that for each region R of $\hat{\mathbb{M}}_h$, R is the intersection of all the k^n regions R_1, R_2, \dots, R_{k^n} that contain R in the k^n height level maps. Now $p \in R$ and $q \in R$ implies that $p \in R_i$ and $q \in R_i, \forall i, 1 \leq i \leq k^n$. Hence p and q lie in the same region for all k^n height level maps, i.e., p and q always lie on an edge of \mathbb{T}_H . This completes the proof. \square

Lemma 6.3.4. *Given two distinct regions R and R' of $\hat{\mathbb{M}}_h$, any two points p, q of R , and any two points p', q' of R' , $\pi(p, p') = \pi(q, q')$.*

Proof. This follows from transitivity. By Lemma 6.3.3, p and q always lie on an edge of T_H , and p' and q' always lie on an edge of T_H . So the event that p and p' lie on an edge of T_H is equivalent to the event that q and q' lie on an edge of T_H . The lemma hence follows. \square

Lemma 6.3.4 gives us the following corollary.

Corollary 6.3.5. *The size of the set $\{\pi(p, q) \mid p, q \in \mathbb{R}^2\}$ is $O(n^6 k^{16})$.*

Proof. For each region of \widehat{M}_h , we choose an arbitrary representative point, resulting in $O(n^3 k^8)$ representative points (by Lemma 6.3.2). For any pair of p and q in \mathbb{R}^2 , $\pi(p, q)$ can be answered using a pair of such representative points (by Lemma 6.3.4). Hence it follows. \square

A Monte-Carlo algorithm. We describe a simple Monte Carlo approach to build a data structure for quickly approximating $\pi(p, q)$, for any two points p, q in \mathbb{R}^2 . More precisely, it will output a value $\hat{\pi}(p, q)$ such that $|\hat{\pi}(p, q) - \pi(p, q)| \leq \varepsilon$, with probability at least $1 - \delta$, for some parameters $0 < \varepsilon, \delta < 1$. For a fixed value s , to be specified later, the preprocessing step has s rounds. In the j -th round the algorithm creates an instance Σ_j using $h_j(v) \in h(v)$ by choosing each $h_j(v)$ using the distribution of $h(v)$. For each $j \leq s$, we construct the linear-size data structure on Σ_j in $O(n \log n)$ time such that given two points p and q , one can determine in $O(\log n)$ time whether p and q lie in the same region of the height level map M_{h_j} , i.e., lie on an edge of the contour tree T_{h_j} , as in [26].

To estimate $\pi(p, q)$, the probability that p and q lie on an edge of T_H , we initialize a counter $c = 0$. For each Σ_j , if p and q lie on an edge of T_{h_j} , we increment c by 1. Finally we use $\hat{\pi}(p, q) = c/s$ to estimate $\pi(p, q)$.

Since each vertex v has k possible values, this algorithm can be implemented very efficiently. Each $h_j(v)$ can be selected in $O(\log k)$ time after preprocessing each $h(v)$, in

$O(k)$ time, into a balanced binary tree with total weight calculated for each subtree [99]. Thus total preprocessing takes $O(s(n(\log n + \log k)) + nk) = O(nk + sn \log(nk))$ time and $O(sn)$ space, and each query takes $O(s \log n)$ time.

It remains to determine the value of s so that $|\hat{\pi}(p, q) - \pi(p, q)| \leq \varepsilon$ for all pairs of p and q in \mathbb{R}^2 , with probability at least $1 - \delta$. For fixed p, q , and instance Σ_j , let \mathbf{X} be the random indicator variable, which is 1 if p and q lie on an edge of \mathbb{T}_{h_j} and 0 otherwise. Since $\mathbb{E}[\mathbf{X}] = \pi(p, q)$ and $\mathbf{X} \in \{0, 1\}$, applying a Chernoff-Hoeffding bound to

$$\hat{\pi}(p, q) = \frac{c}{s} = \frac{1}{s} \sum \mathbf{X},$$

we observe that

$$\Pr [|\hat{\pi}(p, q) - \pi(p, q)| \geq \varepsilon] \leq 2 \exp(-2\varepsilon^2 s). \quad (6.1)$$

For each region of $\widehat{\mathbb{M}}_h$, we choose one representative point, and let P be the resulting set of points. Let $Q = P \times P$ denote all possible combinations of pairs of points in P . By Corollary 6.3.5, if $|\hat{\pi}(p, q) - \pi(p, q)| \leq \varepsilon$ for all pairs $(p, q) \in Q$, then it holds for any two points p, q in \mathbb{R}^2 . By applying the union bound to Eq. (6.1), the probability that there exist two points $p, q \in \mathbb{R}^2$ and $|\hat{\pi}(p, q) - \pi(p, q)| \leq \varepsilon$ is at most $2|Q| \exp(-2\varepsilon^2 s)$. Hence, by setting

$$s = \frac{1}{2\varepsilon^2} \ln \frac{2|Q|}{\delta},$$

$|\hat{\pi}(p, q) - \pi(p, q)| \leq \varepsilon$ for all pairs of points $p, q \in \mathbb{R}^2$ with probability at least $1 - \delta$. By Corollary 6.3.5, $|Q| = O(n^6 k^{16})$, so we obtain the following result.

Theorem 6.3.6. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of each vertex v is described by a discrete distribution of description complexity k , and let $\varepsilon, \delta \in (0, 1)$ be two parameters. It can be preprocessed, in*

$$O(nk + (n/\varepsilon^2) \log(nk) \log(nk/\delta))$$

time, into a data structure of size $O((n/\varepsilon^2)\log(nk/\delta))$, which computes, for any two points $p, q \in \mathbb{R}^2$, in $O((1/\varepsilon^2)\log(nk/\delta)\log n)$ time, a value $\hat{\pi}(p, q)$ such that $|\hat{\pi}(p, q) - \pi(p, q)| \leq \varepsilon$ with probability at least $1 - \delta$.

Continuous case. There are two technical issues in extending this technique and analysis to continuous distributions. First, we instantiate a certain height value from each continuous distribution $h(v)$. Herein we assume the representation of the pdf is such that this can be done in constant time for each $h(v)$.

Second, we need to bound the number of distinct queries that need to be considered to apply the union bound as we did above. Since $\pi(p, q)$ may vary continuously with the locations of p, q , unlike the discrete case, we cannot hope for a bounded number of distinct results. However, we just need to define a finite set \bar{Q} of query points so that for any points $p, q \in \mathbb{R}^2$, there are two points $p', q' \in \bar{Q}$ such that $|\pi(p, q) - \pi(p', q')| \leq \varepsilon/2$. Then we can choose s large enough so that it permits at most $\varepsilon/2$ error on each pair of query points in \bar{Q} . Specifically, choosing $s = O((1/\varepsilon^2)\log(n|\bar{Q}|/\delta))$ is sufficient, so all that remains is to bound $|\bar{Q}|$.

To choose \bar{Q} , we show that each continuous distribution $h(v)$ can be approximated with a discrete distribution $\bar{h}(v)$ of size $O((n^2/\varepsilon^2)\log(n/\delta))$, and then reduce the problem to the discrete case.

For parameters $\alpha > 0$ and $\delta' \in (0, 1)$, set

$$\xi(\alpha) = \frac{c}{\alpha^2} \log \frac{1}{\delta'},$$

where c is a constant. For each $v \in V$, we choose a random sample $\bar{h}(v)$ of size $\xi(\alpha)$, according to the continuous distribution $h(v)$. We regard $\bar{h}(v)$ as a uniform discrete distribution. Let $1(p, q, \Sigma)$ be an indicator function that is 1 when p, q lies on an edge of \mathbb{T}_h of the terrain instance Σ , and 0 otherwise.

We first derive the following lemma.

Lemma 6.3.7. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of one vertex v is described by a continuous distribution while the heights of all other vertices are fixed. Then there are at most two disjoint intervals $I, I' \subseteq \mathbb{R}$ such that if $h(v) \in I \cup I'$, $1(p, q, \Sigma) = 1$ for the resulting terrain Σ .*

Proof. If $1(p, q, \Sigma) = 0$ holds no matter what value $h(v)$ takes, then there is nothing to reason about. The interesting case is when $1(p, q, \Sigma) = 1$ for any terrain Σ with $h(v)$ taking values from an interval $I \subseteq \mathbb{R}$, we argue that there is at most one more interval $I' \subseteq \mathbb{R}$ and $I' \cap I = \emptyset$ such that if $h(v) \in I'$, then $1(p, q, \Sigma) = 1$ for the resulting terrain Σ . Suppose when $h(v) \in I$, p, q lies on an edge defined by $u, t \in V$. There are two cases:

- (i) v has no chance lying in between p and q on \mathbb{T}_h . In this case, p, q will always lie on the edge of u, t , and $I = \mathbb{R}$.
- (ii) v has a chance lying in between p and q on \mathbb{T}_h . In this case, as the height value of v changes from $-\infty$ to ∞ , p, q first lies on the edge of u and t , then p (resp. q) lies on the edge of u (resp. t) and v , then again p, q lies on the edge of u and t . Hence there are two such intervals I, I' that if $h(v) \in I \cup I'$, $1(p, q, \Sigma) = 1$ for the resulting terrain Σ .

This concludes the proof. □

Now assume the same assumption as in Lemma 6.3.7. Let $X \subseteq \mathbb{R}$ be the set of height values of vertex v for which $1(p, q, \Sigma) = 1$. By Lemma 6.3.7, X can be empty, consists of one interval, or consists of two disjoint intervals. A well-known result in the theory of random sampling implies that

$$\left| \int_X h(v) - \int_X \bar{h}(v) \right| \leq \alpha, \tag{6.2}$$

with probability at least $1 - \delta'$, provided that the constant c in $\xi(\alpha)$ is chosen sufficiently large.

Let $\bar{\pi}(p, q)$ denote the probability of p, q lying on an edge on the contour tree $T_{\bar{H}}$, where \bar{H} denotes the random height function of the random terrain generated using the sampled distributions $\bar{h}(v)$. We prove the following:

Lemma 6.3.8. *For any $p, q \in \mathbb{R}^2$,*

$$|\pi(p, q) - \bar{\pi}(p, q)| \leq \alpha n,$$

with probability at least $1 - \delta'$.

Proof. Let the continuous (resp. sampled) distribution of the height function of v_i be denoted $h_i(v_i)$ (resp. $\bar{h}_i(v_i)$), for all $v_i \in V = \{v_1, \dots, v_n\}$. We can now rewrite $\pi(p, q)$ as

$$\pi(p, q) = \int_{v_1} h_1(v_1) \dots \int_{v_n} h_n(v_n) 1(p, q, \Sigma) dv_n \dots dv_1.$$

Consider the innermost integral

$$I_n = \int_{v_n} h_n(v_n) 1(p, q, \Sigma) dv_n$$

in which case $h_1(v_1), \dots, h_{n-1}(v_{n-1})$ are fixed. If $1(p, q, \Sigma)$ is always 0, then there is nothing to reason about, since replacing $h_n(v_n)$ with $\bar{h}_n(v_n)$ does not generate any error. Furthermore, by Lemma 6.3.7, there are at most two intervals $I_1, I_2 \subseteq \mathbb{R}$ such that if $h(v) \in I_1 \cup I_2$, $1(p, q, \Sigma) = 1$ for the resulting terrain Σ , for which $\bar{h}_n(v_n)$ approximates within α . Hence

$$\left| I_n - \int_{v_n} \bar{h}_n(v_n) 1(p, q, \Sigma) dp_n \right| \leq \alpha.$$

Consider the case where $\bar{h}_n(v_n)$ undercounts this probability, change the order of integration, and move the α error outside to obtain

$$\pi(p, q) \leq \int_{v_n} \bar{h}_n(v_n) \int_{v_1} h_1(v_1) \dots \int_{v_{n-1}} h_{n-1}(v_{n-1}) 1(p, q, \Sigma) dv_{n-1} \dots dv_1 dv_n + \alpha.$$

We now repeat this argument $n - 1$ times on the current innermost term until the full integration of the right hand side is exactly $\bar{\pi}(p, q)$ and we obtain $\pi(p, q) \leq \bar{\pi}(p, q) + \alpha n$. Bounding the overestimate of $\bar{\pi}(p, q)$ is symmetric. \square

Thus by setting $\alpha = \varepsilon/2n$, a random sample $\bar{h}(v)$ of size $O((n^2/\varepsilon^2) \log(n/\delta))$ from each continuous distribution $h(v)$ ensures that

$$|\pi(p, q) - \bar{\pi}(p, q)| \leq \varepsilon/2 \tag{6.3}$$

for any pair of points $p, q \in \mathbb{R}^2$. By choosing $\delta' = \delta/2$, Eq.(6.3) holds with probability at least $1 - \delta/2$.

We consider $\widehat{M}_{\bar{h}}$, the overlay of all possible height level maps using the sampling distributions $\bar{h}(v)$. We choose one point from each region of $\widehat{M}_{\bar{h}}$, and set \bar{P} to be the resulting set of points. Let $\bar{Q} = \bar{P} \times \bar{P}$ denote the set of all pairs of points in \bar{P} . For two points $p, q \in \mathbb{R}^2$, let $(\bar{p}, \bar{q}) \in \bar{Q}$ be the pair of representative points such that \bar{p} (resp. \bar{q}) lies in the same region of $\widehat{M}_{\bar{h}}$ as p (resp. q). Then $|\pi(p, q) - \bar{\pi}(\bar{p}, \bar{q})| < \varepsilon/2$ for any two points $p, q \in \mathbb{R}^2$, with probability at least $1 - \delta/2$.

Now applying the analysis for the discrete case on the sampled distributions $\bar{h}(v)$, if we choose

$$s = O\left(\frac{1}{\varepsilon^2} \log \frac{n|\bar{Q}|}{\delta}\right),$$

then $|\bar{\pi}(p, q) - \widehat{\pi}(p, q)| < \varepsilon/2$ for any points $p, q \in \mathbb{R}^2$ with probability at least $1 - \delta/2$.

Since the description complexity \bar{k} of each sampled distribution $\bar{h}(v)$ is

$$\bar{k} = |\bar{h}(v)| = O\left(\frac{n^2}{\varepsilon^2} \log \frac{n}{\delta}\right),$$

by Corollary 6.3.5,

$$|\bar{Q}| = O\left(n^6 \bar{k}^{16}\right) = O(\text{poly}(n/(\varepsilon\delta))).$$

Putting everything together, we obtain the following.

Theorem 6.3.9. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of each vertex v is described by a continuous distribution such that a random instantiation can be performed in constant time, and let $\varepsilon, \delta \in (0, 1)$ be two parameters. It can be preprocessed, in $O((n/\varepsilon^2) \log(n/(\varepsilon\delta)) \log n)$ time, into a data structure of size $O((n/\varepsilon^2) \log(n/(\varepsilon\delta)))$, which computes, for any two points $p, q \in \mathbb{R}^2$, in $O((1/\varepsilon^2) \log(n/(\varepsilon\delta)) \log n)$ time, a value $\hat{\pi}(p, q)$ such that $|\hat{\pi}(p, q) - \pi(p, q)| \leq \varepsilon$ with probability at least $1 - \delta$.*

6.4. The Distance Statistics of Two Points

In this section, we study statistics about the distance of any two points p, q in \mathbb{R}^2 on the contour tree. Here the distance is defined as in [24]: given a terrain Σ , the distance of p, q on its contour tree \mathbb{T}_h , denoted as $d_\Sigma(p, q)$, is

$$d_\Sigma(p, q) = \max_{x \in \chi(p, q)} h_\Sigma(x) - \min_{x \in \chi(p, q)} h_\Sigma(x),$$

where $\chi(p, q)$ denotes the unique path from p to q on \mathbb{T}_h , and $h_\Sigma(x)$ denotes the height value of point x on Σ .

We are interested in the following statistics: (i) the expected distance of p, q , denoted by $\text{Ed}(p, q)$, which can be written as

$$\text{Ed}(p, q) = \sum_{\text{an outcome } \Sigma} \gamma(\Sigma) \cdot d_\Sigma(p, q),$$

and (ii) the probability that the distance of p, q is at least ℓ , i.e.,

$$\Pr\{d_\Sigma(p, q) \geq \ell\} = \sum_{d_\Sigma(p, q) \geq \ell} \gamma(\Sigma).$$

6.4.1. *The expected distance of two points*

Here we assume that for any given vertex v of \mathbb{M} , the range of its k possible height values is bounded, more precisely, we assume that

$$\max_{1 \leq j \leq k} h_v^j - \min_{1 \leq j \leq k} h_v^j \leq \Delta, \quad (6.4)$$

for some bounded parameter $\Delta > 0$.

6.4.1.1. *A deterministic algorithm with additive error Δ*

In this section, we derive a very simple deterministic algorithm to estimate $\text{Ed}(p, q)$, and show that the resulting additive error is bounded by Δ . It also serves as a warm-up for later sections.

The deterministic algorithm. For each vertex v of \mathbb{M} , we define

$$\bar{h}_v = \left(\max_{1 \leq j \leq k} h_v^j + \min_{1 \leq j \leq k} h_v^j \right) / 2.$$

By assigning the height value \bar{h}_v to v for each vertex v of \mathbb{M} , we obtain a terrain $\bar{\Sigma}$, for which we compute its contour tree $\mathbb{T}_{\bar{h}}$. We estimate $\text{Ed}(p, q)$ by using the distance of p and q with respect to the contour tree $\mathbb{T}_{\bar{h}}$, denoted by $d_{\bar{\Sigma}}(p, q)$.

The error analysis: special case. First, we start with a special case, where for any two vertices u and v of \mathbb{M} , if $\bar{h}_u > \bar{h}_v$, then $h_u^i > h_v^j$, for each $i, j \in \{1, \dots, k\}$. In this case, the contour tree \mathbb{T}_h of each instance Σ “resembles” the contour tree $\mathbb{T}_{\bar{h}}$, that is, \mathbb{T}_h and $\mathbb{T}_{\bar{h}}$ are the same in terms of nodes, edges and relative height value orders, but not the exact height values. Notably,

$$\arg \max_{x \in \chi(p, q)} h_{\Sigma}(x) = \arg \max_{x \in \bar{\chi}(p, q)} h_{\bar{\Sigma}}(x) = u,$$

and

$$\arg \min_{x \in \chi(p,q)} h_{\Sigma}(x) = \arg \min_{x \in \bar{\chi}(p,q)} h_{\bar{\Sigma}}(x) = v,$$

for some vertices u, v of \mathbb{M} , where $\chi(p, q), \bar{\chi}(p, q)$ denote the paths of p and q on $\mathbb{T}_h, \mathbb{T}_{\bar{h}}$, respectively; and $h_{\Sigma}(x), h_{\bar{\Sigma}}(x)$ denote the heights of location x in the terrains $\Sigma, \bar{\Sigma}$, respectively. Note that $|h_{\Sigma}(u) - h_{\bar{\Sigma}}(u)| \leq \Delta/2$, and $|h_{\Sigma}(v) - h_{\bar{\Sigma}}(v)| \leq \Delta/2$. Therefore,

$$\begin{aligned} |d_{\Sigma}(p, q) - d_{\bar{\Sigma}}(p, q)| &= |h_{\Sigma}(u) - h_{\Sigma}(v) - (h_{\bar{\Sigma}}(u) - h_{\bar{\Sigma}}(v))| \\ &\leq |h_{\Sigma}(u) - h_{\bar{\Sigma}}(u)| + |h_{\Sigma}(v) - h_{\bar{\Sigma}}(v)| \\ &\leq \Delta/2 + \Delta/2 = \Delta, \end{aligned}$$

where $d_{\Sigma}(p, q)$ and $d_{\bar{\Sigma}}(p, q)$ denote the distance of p and q with respect to \mathbb{T}_h and $\mathbb{T}_{\bar{h}}$, respectively.

Summing over all possible instances Σ , we obtain

$$|\mathbb{E}d(p, q) - d_{\bar{\Sigma}}(p, q)| = \left| \sum_{\text{an outcome } \Sigma} \gamma(\Sigma) \cdot d_{\Sigma}(p, q) - d_{\bar{\Sigma}}(p, q) \right| \leq \Delta,$$

where $\gamma(\Sigma)$ denotes the probability that the terrain instance Σ exists.

The error analysis: general case. Next, we show that the additive error is no more than Δ even without the above assumption for the special case. Fix a terrain instance Σ . We again want to show that

$$|d_{\Sigma}(p, q) - d_{\bar{\Sigma}}(p, q)| \leq \Delta. \tag{6.5}$$

Let $\chi(p, q), \bar{\chi}(p, q)$ denote the paths of p and q on $\mathbb{T}_h, \mathbb{T}_{\bar{h}}$, respectively; Define $u = \arg \max_{x \in \chi(p,q)} h_{\Sigma}(x)$, $v = \arg \min_{x \in \chi(p,q)} h_{\Sigma}(x)$, $u' = \arg \max_{x \in \bar{\chi}(p,q)} h_{\bar{\Sigma}}(x)$, $v' = \arg \min_{x \in \bar{\chi}(p,q)} h_{\bar{\Sigma}}(x)$. We show that $|h_{\Sigma}(u) - h_{\bar{\Sigma}}(u')| \leq \Delta/2$ and $|h_{\Sigma}(v) - h_{\bar{\Sigma}}(v')| \leq \Delta/2$; next we only show the former, since the latter can be obtained in a similar

manner. Without loss of generality, we assume that $h_\Sigma(u) > h_{\bar{\Sigma}}(u')$. We distinguish two cases:

1. $u \in \bar{\chi}(p, q)$.

Note that $h_{\bar{\Sigma}}(u') > h_{\bar{\Sigma}}(u)$ and $h_\Sigma(u) - h_{\bar{\Sigma}}(u) \leq \Delta/2$. Therefore

$$h_\Sigma(u) - h_{\bar{\Sigma}}(u') < h_\Sigma(u) - h_{\bar{\Sigma}}(u) \leq \Delta/2.$$

2. $u \notin \bar{\chi}(p, q)$.

Since the path $\chi(p, q)$ of p and q passes through u on the contour tree \mathbb{T}_h , it implies that on the height level map \mathbb{M}_h , the dual of \mathbb{T}_h , there exists a contour C_u of u such that p and q are separated by C_u , namely, one of p and q lies in the region bounded by C_u , while the other lies outside. Note that on the height level map $\mathbb{M}_{\bar{h}}$, C_u is not a valid contour any more. (Otherwise it has to be the contour of u since it passes through u , and then u would lie on the path of p and q , since any path would have to pass through u ; this contradicts with the assumption that $u \notin \bar{\chi}(p, q)$.) Nevertheless, for any point $x \in C_u$, $|h_{\bar{\Sigma}}(x) - h_\Sigma(u)| < \Delta/2$. Furthermore, $h_{\bar{\Sigma}}(u') \geq h_{\bar{\Sigma}}(x)$, for any $x \in C_u$. Therefore

$$h_\Sigma(u) - h_{\bar{\Sigma}}(u') < h_\Sigma(u) - h_{\bar{\Sigma}}(x) \leq \Delta/2.$$

The remaining argument follows from the analysis for the special case. Hence,

$$|\text{Ed}(p, q) - d_{\bar{\Sigma}}(p, q)| \leq \Delta.$$

Remarks. For simplicity of discussions, we did not consider the cases when p or q realizes the maximum height or the minimum height on the path $\chi(p, q)$ in our error analysis. One can easily incorporate those cases, considering that $|h_\Sigma(p) - h_{\bar{\Sigma}}(p)| \leq \Delta/2$ and $|h_\Sigma(q) - h_{\bar{\Sigma}}(q)| \leq \Delta/2$.

Note that the above applies to the continuous case if the height range of each vertex is bounded by Δ as well. Furthermore, one can compute $\mathbb{T}_{\bar{h}}$ in $O(n \log n)$ time [33], and preprocess it for answering each distance query in $O(\log n)$ time. We conclude the following.

Theorem 6.4.1. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of each vertex v is described by a probability distribution with bounded range Δ . A representative contour tree $\mathbb{T}_{\bar{h}}$ can be constructed in $O(n \log n)$ time such that for any two point p, q in \mathbb{R}^2 , $\text{Ed}(p, q)$ can be approximated using $\mathbb{T}_{\bar{h}}$ in $O(\log n)$ time within additive error Δ .*

6.4.1.2. A Monte-Carlo method with additive error $\varepsilon\Delta$

In this section, we present a simple Monte-Carlo method, by adapting [79, Lemma 3.1]. Given two parameters $\varepsilon, \delta \in (0, 1)$, it outputs a value $\widehat{\text{Ed}}(p, q)$, such that with probability at least $1 - \delta$,

$$|\widehat{\text{Ed}}(p, q) - \text{Ed}(p, q)| \leq \varepsilon\Delta.$$

for any two points p and q satisfying the condition $\text{Ed}(p, q) \geq \varepsilon\Delta + m_{p,q}$, where $m_{p,q}$ is the minimum possible distance $d_{\Sigma}(p, q)$ among all $O(k^n)$ instances Σ . This method admits much smaller additive error compared with the one presented in Section 6.4.1.1, while being slightly less efficient and non-deterministic.

The algorithm. We draw N samples according to the distribution of the height of each vertex v , where N is a parameter to be determined later. Given any two points p and q in \mathbb{R}^2 , we estimate $\text{Ed}(p, q)$ by using $\widehat{\text{Ed}}(p, q)$, the average distance of p and q among these N samples.

The analysis. We start by restating [79, Lemma 3.1].

Lemma 6.4.2. (*Chernoff Bound.*) Let X_1, \dots, X_N be i.i.d. random variables over a bounded domain $[0, U]$ with expectation $\mathbb{E}[X_i] = \mu$. Let $\bar{X} = \frac{1}{N} \sum_{i=1}^N X_i$. Then for $0 < \varepsilon < 2e - 1$, we have

$$\Pr[(1 - \varepsilon)\mu \leq \bar{X} \leq (1 + \varepsilon)\mu] > 1 - 2e^{-N(\mu/U)\varepsilon^2/4}.$$

Hence, to ensure correctness with probability at least $1 - \delta$, one needs to set the number of samples to be $N = O((\lambda/\varepsilon^2) \log(1/\delta))$, where λ is the ratio between U , the maximum possible value of X_i , and μ , the expected value of X_i .

For fixed p and q , let

$$M_{p,q} = \max_{\Sigma} d_{\Sigma}(p, q) \text{ and } m_{p,q} = \min_{\Sigma} d_{\Sigma}(p, q) \quad (6.6)$$

denote the maximum and minimum possible distance of p, q among all $O(k^n)$ instances, respectively. Let

$$W_{p,q} = M_{p,q} - m_{p,q}. \quad (6.7)$$

denote the span (range) of $d_{\Sigma}(p, q)$ among all $O(k^n)$ instances Σ . Note that $W_{p,q} \leq 2\Delta$ by the analysis in Section 6.4.1.1. Set

$$\lambda_{p,q} = W_{p,q}/(\text{Ed}(p, q) - m_{p,q}). \quad (6.8)$$

If we set the number of samples to be $N = O((\lambda_{p,q}/\varepsilon^2) \log(1/\delta))$ and use $\varepsilon/2$ instead of ε , then by Lemma 6.4.2 and the fact that $\text{Ed}(p, q) - m_{p,q} \leq W_{p,q} \leq 2\Delta$, we have that, with probability at least $1 - \delta$,

$$\begin{aligned} & |\widehat{\text{Ed}}(p, q) - \text{Ed}(p, q)| \\ &= |(\widehat{\text{Ed}}(p, q) - m_{p,q}) - (\text{Ed}(p, q) - m_{p,q})| \\ &\leq (\varepsilon/2) \cdot (\text{Ed}(p, q) - m_{p,q}) \\ &\leq \varepsilon\Delta. \end{aligned}$$

To have a reasonable running time, we have to control the value $\lambda_{p,q}$, hence the number of samples. Note that if

$$\text{Ed}(p, q) \geq \varepsilon\Delta + m_{p,q}, \quad (6.9)$$

then $\lambda_{p,q} = O(1/\varepsilon)$, and $N = O((1/\varepsilon^3) \log(1/\delta))$. Therefore, this method is efficient when Eq. (6.9) holds for fixed p, q .

Next, we show that how to extend this to estimate $\text{Ed}(p, q)$, for *any* two points p and q satisfying Eq. (6.9).

We again turn to $\widehat{\mathbb{M}}_h$, the overlay of all the height level maps. For our purpose here, we further overlay $\widehat{\mathbb{M}}_h$ with the triangulation \mathbb{M} , and let $\overline{\mathbb{M}}_h$ denote the resulting subdivision. Note that the size of $\overline{\mathbb{M}}_h$ remains $O(n^3 k^8)$, and furthermore, the height of each point in a region is determined by three vertices of a triangle (this is why we overlay \mathbb{M} with $\widehat{\mathbb{M}}_h$). For each region R of $\overline{\mathbb{M}}_h$, we pick a small (finite) set $R_0 \subset R$ of representative points such that for any point $p \in R$, there exists a representative point $z \in R_0$ with

$$|h_\Sigma(z) - h_\Sigma(p)| \leq \eta/4, \quad (6.10)$$

for any instance Σ .

Note that each region R is fully contained inside a triangle $t = \Delta(v_1, v_2, v_3)$ of \mathbb{M} . Define

$$h(t) = \max\{h(v_1), h(v_2), h(v_3)\} - \min\{h(v_1), h(v_2), h(v_3)\}. \quad (6.11)$$

$$H = \max_{t \text{ in } \mathbb{M}} h(t). \quad (6.12)$$

Note that H is bounded by the difference between the maximum possible height and the minimum possible height in the $O(k^n)$ terrains.

To ensure additive error $\eta/4$ in Eq. (6.10), the number of representative points we need to choose in one region is $O(H^3 k^3 / \eta^3)$. Since the number of regions is

$O(n^3k^8)$, the total number of representative points is $O(n^3k^{11}H^3/\eta^3)$. For any two representative points z, z' , we define $\lambda_{z,z'}$ similarly as in Eq. (6.8) for $\lambda_{p,q}$. Let

$$\lambda = \max_{z \neq z'} \lambda_{z,z'}. \quad (6.13)$$

By decreasing δ by a factor of $O(n^3k^{11}W^3/\eta^3)$ and hence setting

$$N = O((\lambda/\varepsilon^2) \log((nkH)/(\delta\eta))),$$

we obtain that for any pair of such representative points z, z' , with probability at least $1 - \delta$,

$$|\widehat{\text{Ed}}(z, z') - \text{Ed}(z, z')| \leq \varepsilon\Delta. \quad (6.14)$$

Now for any two points p and q , suppose z, z' are two such representative points that

$$|h_\Sigma(z) - h_\Sigma(p)| \leq \eta/4 \text{ and } |h_\Sigma(z') - h_\Sigma(q)| \leq \eta/4,$$

for any instance Σ . Then it is easy to see that

$$|\text{Ed}(p, q) - \text{Ed}(z, z')| \leq \eta/2 \text{ and } |\widehat{\text{Ed}}(p, q) - \widehat{\text{Ed}}(z, z')| \leq \eta/2.$$

Therefore, with probability at least $1 - \delta$,

$$|\widehat{\text{Ed}}(p, q) - \text{Ed}(p, q)| \leq \varepsilon\Delta + \eta, \quad (6.15)$$

for any two points p and q in \mathbb{R}^2 .

Note that if

$$\text{Ed}(p, q) \geq \varepsilon\Delta + m_{p,q} + \eta/2, \quad (6.16)$$

then $\text{Ed}(z, z') \geq \varepsilon\Delta + m_{z,z'}$, and $\lambda_{z,z'} = O(1/\varepsilon)$. In other words, if we set $\lambda = O(1/\varepsilon)$ globally, then for any two points p, q satisfying Eq. (6.16), Eq. (6.15) holds. By decreasing ε to $\varepsilon/2$ and setting $\eta = \varepsilon\Delta$, Eq. (6.16) becomes Eq. (6.9), and $N = O((1/\varepsilon^3) \log((nkH)/(\varepsilon\Delta\delta)))$. Therefore, we obtain the following.

Theorem 6.4.3. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of each vertex v is described by a discrete distribution of description complexity k with bounded range Δ . Let H be as defined in Eq. (6.12). It can be preprocessed, in*

$$O(nk + (n/\varepsilon^3) \log(nk) \log((nkH)/(\varepsilon\Delta\delta)))$$

time, into a data structure of size $O((n/\varepsilon^3) \log((nkH)/(\varepsilon\Delta\delta)))$, which computes, for any two points $p, q \in \mathbb{R}^2$ satisfying the condition $\text{Ed}(p, q) \geq \varepsilon\Delta + m_{p,q}$, in $O((1/\varepsilon^3) \log((nkH)/(\varepsilon\Delta\delta)) \log n)$ time, a value $\widehat{\text{Ed}}(p, q)$ such that $|\widehat{\text{Ed}}(p, q) - \text{Ed}(p, q)| \leq \varepsilon\Delta$ with probability at least $1 - \delta$.

Remarks. If we bound the variance (as a function of ε), besides the range, of the k height values at v , then the condition $\text{Ed}(p, q) \geq \varepsilon\Delta + m_{p,q}$ may hold for any p, q , i.e., this method may be efficient for all pair of points p, q in \mathbb{R}^2 with the additional assumption.

6.4.1.3. A Monte-Carlo method with relative error $\varepsilon\text{Ed}(p, q)$ and additive error $3\varepsilon\Delta$

In this section, we present another simple Monte-Carlo method, which guarantees $|\widehat{\text{Ed}}(p, q) - \text{Ed}(p, q)| \leq \varepsilon\text{Ed}(p, q) + 3\varepsilon\Delta$ with probability at least $1 - \delta$, for two parameters $\varepsilon, \delta \in (0, 1)$ and any two points $p, q \in \mathbb{R}^2$. This method gets rid of the condition $\text{Ed}(p, q) \geq \varepsilon\Delta + m_{p,q}$ required for the one presented in Section 6.4.1.2, while adds a factor of $O(1/\varepsilon)$ in terms of efficiency and additional relative error.

To start, we fix $p, q \in \mathbb{R}^2$ and $I \subseteq \mathbb{R}$, and we want to use Monte Carlo to estimate $\Pr\{d_\Sigma(p, q) \in I\}$, the probability that $d_\Sigma(p, q) \in I$, within ε additive error with high probability. This is the most basic usage of Monte Carlo, and it is easy to derive the following.

Lemma 6.4.4. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of each vertex v is described by a discrete distribution of description complexity k , and let $\varepsilon, \delta \in (0, 1)$ be two parameters. Given **fixed** $p, q \in \mathbb{R}^2$ and $I \subseteq \mathbb{R}$, using space $O((n/\varepsilon^2) \log(1/\delta))$, we can compute, in $O(nk + (n/\varepsilon^2) \log(nk) \log(1/\delta))$ time, a value $\widehat{\Pr}\{d_\Sigma(p, q) \in I\}$ such that $|\widehat{\Pr}\{d_\Sigma(p, q) \in I\} - \Pr\{d_\Sigma(p, q) \in I\}| \leq \varepsilon$ with probability at least $1 - \delta$.*

Next, we show how to extend Lemma 6.4.4 to estimate $\text{Ed}(p, q)$ for fixed points $p, q \in \mathbb{R}^2$. Let H be the difference between the maximum possible height and the minimum possible height among all instances. We divide the interval $[0, H]$ into a set \mathcal{J} of $O(H/\eta)$ pairwise-disjoint intervals: $I_0 = [0, \eta), I_1 = [\eta, 2\eta), \dots, I_s = [s\eta, H]$, where $s = \lfloor H/\eta \rfloor$. Let $d_I^{\text{mid}} = (i + 0.5) \cdot \eta$ and $d_I^{\text{min}} = i \cdot \eta$. After decreasing δ by a factor of $O(H/\eta)$, we apply Lemma 6.4.4 to estimate $\Pr\{d(p, q) \in I\}$ for each $I \in \mathcal{J}$. Let $M_{p,q}, m_{p,q}, W_{p,q}$ be defined as in Eq. (6.6) and Eq. (6.7), respectively. Let $I_{p,q} = [m_{p,q}, M_{p,q}]$, and $\mathcal{J}_{p,q} = \bigcup_{I \in \mathcal{J}, I \cap I_{p,q} \neq \emptyset} I$. Define

$$\widehat{\text{Ed}}_I(p, q) = \widehat{\Pr}\{d(p, q) \in I\} \cdot d_I^{\text{mid}}, \text{ and } \widehat{\text{Ed}}(p, q) = \sum_{I \in \mathcal{J}_{p,q}} \widehat{\text{Ed}}_I(p, q).$$

Lemma 6.4.5. *Given fixed p, q , with probability $1 - \delta$,*

$$|\widehat{\text{Ed}}(p, q) - \text{Ed}(p, q)| \leq \varepsilon \sum_{I \in \mathcal{J}_{p,q}} d_I^{\text{mid}} + \eta/2.$$

The rest follows from Lemma 6.4.4 by decreasing δ with a factor of $O(H/\eta)$.

Proof. Fix an interval $I \in \mathcal{J}_{p,q}$. Define

$$\text{Ed}_I(p, q) = \sum_{\ell \in I, \Pr\{d(p, q) = \ell\} > 0} \Pr\{d(p, q) = \ell\} \cdot \ell.$$

Then $\text{Ed}(p, q) = \sum_{I \in \mathcal{J}_{p,q}} \text{Ed}_I(p, q)$. We first show that

$$|\widehat{\text{Ed}}_I(p, q) - \text{Ed}_I(p, q)| \leq \Pr\{d(p, q) \in I\} \cdot \eta/2 + \varepsilon \cdot d_I^{\text{mid}}.$$

When $0 \leq \widehat{\Pr}\{d(p, q) \in I\} - \Pr\{d(p, q) \in I\} \leq \varepsilon$,

$$\begin{aligned}
& |\widehat{\mathbf{E}d}_I(p, q) - \mathbf{E}d_I(p, q)| \\
&= |\widehat{\Pr}\{d(p, q) \in I\} \cdot d_I^{\text{mid}} - \sum_{\ell \in I, \Pr\{d(p, q) = \ell\} > 0} \Pr\{d(p, q) = \ell\} \cdot \ell| \\
&\leq |\varepsilon \cdot d_I^{\text{mid}} - \sum_{\ell \in I, \Pr\{d(p, q) = \ell\} > 0} \Pr\{d(p, q) = \ell\} \cdot (\ell - d_I^{\text{mid}})| \\
&\leq \varepsilon \cdot d_I^{\text{mid}} + \Pr\{d(p, q) \in I\} \cdot \eta/2.
\end{aligned}$$

Similarly, when $0 \leq \Pr\{d(p, q) \in I\} - \widehat{\Pr}\{d(p, q) \in I\} \leq \varepsilon$,

$$\begin{aligned}
& |\widehat{\mathbf{E}d}_I(p, q) - \mathbf{E}d_I(p, q)| \\
&= |\widehat{\Pr}\{d(p, q) \in I\} \cdot d_I^{\text{mid}} - \sum_{\ell \in I, \Pr\{d(p, q) = \ell\} > 0} \Pr\{d(p, q) = \ell\} \cdot \ell| \\
&\leq \widehat{\Pr}\{d(p, q) \in I\} \cdot \eta/2 + \varepsilon \cdot (d_I^{\text{min}} + \eta/2). \\
&\leq \varepsilon \cdot d_I^{\text{mid}} + \Pr\{d(p, q) \in I\} \cdot \eta/2.
\end{aligned}$$

The lemma follows from summing over all intervals $I \in \mathcal{J}_{p, q}$. \square

Note that $W_{p, q} \leq 2\Delta$, hence $|\mathcal{J}_{p, q}| \leq \lceil 2\Delta/\eta \rceil + 1$. If we choose $\eta = \varepsilon\Delta$, then $|\mathcal{J}_{p, q}| \leq 2/\varepsilon + 2 \leq 4/\varepsilon$. Applying Lemma 6.4.5 with $\varepsilon^2/4$ instead of ε , we have, with probability $1 - \delta$,

$$\begin{aligned}
|\widehat{\mathbf{E}d}(p, q) - \mathbf{E}d(p, q)| &\leq (\varepsilon^2/4) \sum_{I \in \mathcal{J}_{p, q}} d_I^{\text{mid}} + \eta/2 \\
&\leq (\varepsilon^2/4) |\mathcal{J}_{p, q}| (\mathbf{E}d(p, q) + 2\Delta) + \varepsilon\Delta/2 \\
&\leq \varepsilon \mathbf{E}d(p, q) + 5\varepsilon\Delta/2.
\end{aligned}$$

Next, we show that how to extend this to estimate $\mathbf{E}d(p, q)$ for any two points p, q in \mathbb{R}^2 . The idea is the same as for Section 6.4.1.2. We choose a small finite set of representative points so that for any two points p, q , there exist two representative

points z, z' such that

$$|\text{Ed}(p, q) - \text{Ed}(z, z')| \leq \varepsilon\Delta/4, \text{ and } |\widehat{\text{Ed}}(p, q) - \widehat{\text{Ed}}(z, z')| \leq \varepsilon\Delta/4.$$

It can be shown that if $|\widehat{\text{Ed}}(z, z') - \text{Ed}(z, z')| \leq \varepsilon\text{Ed}(z, z') + 5\varepsilon\Delta/2$, then

$$|\widehat{\text{Ed}}(p, q) - \text{Ed}(p, q)| \leq \varepsilon\text{Ed}(p, q) + 3\varepsilon\Delta.$$

Therefore, we conclude the following.

Theorem 6.4.6. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of each vertex v is described by a discrete distribution of description complexity k with bounded range Δ , and let $\varepsilon, \delta \in (0, 1)$ be two parameters. Furthermore, let H be the difference between the maximum possible height and the minimum possible height among all instances. It can be preprocessed, in*

$$O(nk + (n/\varepsilon^4) \log(nk) \log((nkH)/(\varepsilon\Delta\delta)))$$

time, into a data structure of size $O((n/\varepsilon^4) \log((nkH)/(\varepsilon\Delta\delta)))$, which computes, for any two points $p, q \in \mathbb{R}^2$, in $O((1/\varepsilon^4) \log((nkH)/(\varepsilon\Delta\delta)) \log n)$ time, a value $\widehat{\text{Ed}}(p, q)$ such that $|\widehat{\text{Ed}}(p, q) - \text{Ed}(p, q)| \leq \varepsilon\text{Ed}(p, q) + 3\varepsilon\Delta$ with probability at least $1 - \delta$.

Remarks. When $\Delta \leq \text{Ed}(p, q)/3$, by decreasing ε by a factor of 2, the error becomes $\varepsilon\text{Ed}(p, q)$. That is, we can get rid of the additive error $3\varepsilon\Delta$, and only have the relative error.

6.4.2. Probability that the distance of two points is at least ℓ

In this section, we describe how to estimate the other distance statistics we are interested in: $\Pr\{d_{\Sigma}(p, q) \geq \ell\}$, the probability that the distance of p, q is at least ℓ .

We first show how to estimate $\Pr\{d_{\Sigma}(p, q) \geq \ell\}$ for fixed $p, q \in \mathbb{R}^2$ and any $\ell \in \mathbb{R}$, then extend it to any $p, q \in \mathbb{R}^2$ and any $\ell \in \mathbb{R}$.

Fixed p, q . For a fixed pair of points $p, q \in \mathbb{R}^2$ and a fixed value $\ell \in \mathbb{R}$, it is straightforward to estimate $\Pr\{d_{\Sigma}(p, q) \geq \ell\}$ using Monte-Carlo. Namely, draw N samples, and if $d_{\Sigma}(p, q) \geq \ell$ for s samples Σ , we set the estimate $\widehat{\Pr}\{d_{\Sigma}(p, q) \geq \ell\} = s/N$. The number of instances we need is $N = O(1/\varepsilon^2 \log(1/\delta))$ such that $|\widehat{\Pr}\{d_{\Sigma}(p, q) \geq \ell\} - \Pr\{d_{\Sigma}(p, q) \geq \ell\}| \leq \varepsilon$ with probability at least $1 - \delta$.

Now we extend it to any value $\ell \in \mathbb{R}$ while we fix p, q . The main observation is that though there are infinite number of values of ℓ , the number of distinct values of $\Pr\{d_{\Sigma}(p, q) \geq \ell\}$ is only polynomial. Recall that given a terrain Σ ,

$$d_{\Sigma}(q, p) = \max_{x \in \chi(p, q)} h_{\Sigma}(x) - \min_{x \in \chi(p, q)} h_{\Sigma}(x),$$

where $\chi(p, q)$ denotes the unique path from p to q on \mathbb{T}_h , and $h_{\Sigma}(x)$ denotes the height value of point x on Σ . It is easy to see that

$$\max_{x \in \chi(p, q)} h_{\Sigma}(x), \min_{x \in \chi(p, q)} h_{\Sigma}(x) \in \{h(p), h(q), h(v) \mid \forall v \in V\}.$$

That is, the maximum (resp. minimum) height on $\chi(p, q)$ has only $O(nk + k^3)$ possible choices, implying that $d_{\Sigma}(p, q)$ has only $O(nk + k^3)^2$ distinct values. Therefore, it is sufficient to choose $O(nk + k^3)^2$ number of representative values ℓ' such that given a value ℓ , there exists a representative value ℓ' satisfying

$$\Pr\{d_{\Sigma}(p, q) \geq \ell\} = \Pr\{d_{\Sigma}(p, q) \geq \ell'\}. \quad (6.17)$$

By decreasing δ by a factor of $O(nk + k^3)^2$, we conclude the following.

Theorem 6.4.7. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of each vertex v is described by a discrete distribution of description complexity k , and let $\varepsilon, \delta \in (0, 1)$ be two parameters. It can be preprocessed, in*

$$O(nk + (n/\varepsilon^2) \log(nk) \log(nk/\delta))$$

time, into a data structure of size $O((n/\varepsilon^2) \log(nk/\delta))$, which computes, for **fixed** points $p, q \in \mathbb{R}^2$ and any value $\ell \in \mathbb{R}$, in $O((1/\varepsilon^2) \log(nk/\delta) \log n)$ time, a value $\widehat{\Pr}\{d_{\Sigma}(p, q) \geq \ell\}$ such that $|\widehat{\Pr}\{d_{\Sigma}(p, q) \geq \ell\} - \Pr\{d_{\Sigma}(p, q) \geq \ell\}| \leq \varepsilon$ with probability at least $1 - \delta$.

Any p, q . We extend the above to any pair of points $p, q \in \mathbb{R}^2$. The challenge lies in that there are infinite number of values $h(p), h(q)$. To resolve this, we again adapt the idea from Section 6.4.1.2. For a parameter $\eta > 0$, we choose a small finite set of representative points so that for any two points p, q , there exist two representative points z, z' such that for any terrain instance Σ ,

$$|d_{\Sigma}(p, q) - d_{\Sigma}(z, z')| \leq \eta/2. \quad (6.18)$$

We can show that if for any ℓ , $|\widehat{\Pr}\{d_{\Sigma}(z, z') \geq \ell\} - \Pr\{d_{\Sigma}(z, z') \geq \ell\}| \leq \varepsilon$, then

$$\Pr\{d_{\Sigma}(p, q) \geq \ell + \eta\} - \varepsilon \leq \widehat{\Pr}\{d_{\Sigma}(p, q) \geq \ell\} \leq \Pr\{d_{\Sigma}(p, q) \geq \ell - \eta\} + \varepsilon.$$

As in Section 6.4.1.2, the total number of representative points we need to choose is $O(n^3 k^{11} H^3 / \eta^3)$, resulting in $O(n^6 k^{22} H^6 / \eta^6)$ pair of representative points z, z' . For each pair of representative points z, z' , we choose $O(nk + k^3)^2$ number of representative values of ℓ' . Now for any $p, q \in \mathbb{R}^2$ and $\ell \in \mathbb{R}$, we can find representative points z, z' and representative value ℓ' such that Eq. (6.17) and Eq. (6.18) hold.

By further decreasing δ by a factor of $O(n^6 k^{22} H^6 / \eta^6)$ in Theorem 6.4.7, we conclude the following.

Theorem 6.4.8. *Given a terrain in \mathbb{R}^3 , induced by a triangulation $\mathbb{M} = (V, E, F)$ in \mathbb{R}^2 , where $n = |V|$ and the height $h(v)$ of each vertex v is described by a discrete distribution of description complexity k , and let $\varepsilon, \delta \in (0, 1), \eta > 0$ be three parameters. It can be preprocessed, in*

$$O(nk + (n/\varepsilon^2) \log(nk) \log(nkH/(\delta\eta)))$$

time, into a data structure of size $O((n/\varepsilon^2) \log(nkH/(\delta\eta)))$, which computes, for any points $p, q \in \mathbb{R}^2$ and any value $\ell \in \mathbb{R}$, in $O((1/\varepsilon^2) \log(nkH/(\delta\eta)) \log n)$ time, a value $\widehat{\Pr}\{d_{\Sigma}(p, q) \geq \ell\}$ such that

$$\Pr\{d_{\Sigma}(p, q) \geq \ell + \eta\} - \varepsilon \leq \widehat{\Pr}\{d_{\Sigma}(p, q) \geq \ell\} \leq \Pr\{d_{\Sigma}(p, q) \geq \ell - \eta\} + \varepsilon.$$

with probability at least $1 - \delta$.

6.5. Conclusion

In this chapter we studied contour trees of terrains in a probabilistic setting. We first showed that the probability of a vertex being a critical point, and the expected number of nodes (resp. edges) of the contour tree, can be computed exactly efficiently. Then we presented efficient sampling-based methods for estimating, with high probability, (i) the probability that two points lie on an edge of the contour tree, within additive error; (ii) the expected distance of two points p, q and the probability that the distance of p, q is at least ℓ on the contour tree, within additive error and/or relative error. We conclude this chapter with a few open problems:

1. How hard is it to compute the probability of two points lying on an edge of the contour tree *exactly*? What about the distance statistics of two points?
2. Given a point p in \mathbb{R}^2 , let $\Gamma(p) \subseteq \mathbb{R}^2$ denote the set of points q such that p and q always lie on an edge of T_H . How quickly can we compute $\Gamma(p)$?
3. What is a robust and useful contour tree representation of a terrain in the presence of data uncertainty?

Bibliography

- [1] P. Afshani, P. Agarwal, L. Arge, K. Larsen, and J. Phillips, (Approximate) uncertain skylines, *Proc. 14th IEEE Int. Conf. Data Theory*, 2011, pp. 186–196.
- [2] P. Afshani and T. M. Chan, Optimal halfspace range reporting in three dimensions, *Proc. 20th Annu. ACM-SIAM Sympos. Discrete Algs.*, 2009, pp. 180–186.
- [3] P. K. Agarwal, Range searching, in: *Handbook of Discrete and Computational Geometry* (J. E. Goodman and J. O’Rourke, eds.), CRC Press LLC, 2nd edition, 2004, pp. 809–838.
- [4] P. K. Agarwal, L. Arge, T. Mølhave, M. Revsbæk, and J. Yang, Maintaining contour trees of dynamic terrains, *CoRR*, abs/1406.4005 (2014).
- [5] P. K. Agarwal, L. Arge, and K. Yi, I/O-efficient batched union-find and its applications to terrain analysis, *ACM Trans. Algs.*, 7 (2010), 11:1–11:21.
- [6] P. K. Agarwal, B. Aronov, S. Har-Peled, J. M. Phillips, K. Yi, and W. Zhang, Nearest neighbor searching under uncertainty II, *Proc. 32nd ACM Sympos. Principles Database Syst.*, 2013, pp. 115–126.
- [7] P. K. Agarwal, S. Cheng, Y. Tao, and K. Yi, Indexing uncertain data, *Proc. 28th ACM Sympos. Principles Database Syst.*, 2009, pp. 137–146.
- [8] P. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf, Constructing levels in arrangements and higher order Voronoi diagrams, *SIAM J. Comput.*, 27 (1998), 654–667.
- [9] P. K. Agarwal, A. Efrat, S. Sankararaman, and W. Zhang, Nearest-neighbor searching under uncertainty, *Proc. 31st ACM Sympos. Principles Database Syst.*, 2012, pp. 225–236.
- [10] P. K. Agarwal, A. Efrat, and M. Sharir, Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications, *SIAM J. Comput.*, 29 (1999), 912–953.

- [11] P. K. Agarwal, S. Har-Peled, M. Sharir, and Y. Wang, Hausdorff distance under translation for points and balls, *ACM Trans. Algs.*, 6 (2010), 71:1–71:26.
- [12] P. K. Agarwal, S. Har-Peled, S. Suri, H. Yıldız, and W. Zhang, Convex hulls under uncertainty, *Proc. 22nd Annu. European Sympos. Algs.*, 2014, pp. 37–48.
- [13] P. K. Agarwal and J. Matousek, Ray shooting and parametric search, *SIAM J. Comput.*, 22 (1993), 794–806.
- [14] P. K. Agarwal and M. Sharir, Efficient algorithms for geometric optimization, *ACM Comput. Surv.*, 30 (1998), 412–458.
- [15] P. K. Agarwal and M. Sharir, Arrangements and their applications, in: *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, eds.), Elsevier, 2000, pp. 49–119.
- [16] P. K. Agarwal, M. Sharir, and E. Welzl, Algorithms for center and Tverberg points, *ACM Trans. Algs.*, 5 (2008), 5:1–5:20.
- [17] C. C. Aggarwal, *Managing and Mining Uncertain Data*, Springer-Verlag, 2009.
- [18] H.-K. Ahn, S. Bae, and W. Son, Group nearest neighbor queries in the L_1 plane, *Proc. 10th Annu. Conf. Theory Appl. Models of Comput.*, 2013, pp. 52–61.
- [19] A. Andoni and P. Indyk, Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions, *Commun. ACM*, 51 (2008), 117–122.
- [20] S. Arya, T. Malamatos, and D. M. Mount, Space-time tradeoffs for approximate nearest neighbor searching, *J. ACM*, 57 (2009), 1:1–1:54.
- [21] P. F. Ash and E. D. Bolker, Generalized Dirichlet tessellations, *Geometriae Dedicata*, 20 (1986), 209–243.
- [22] M. Atallah and Y. Qi, Computing all skyline probabilities for uncertain data, *Proc. 28th ACM Sympos. Principles Database Syst.*, 2009, pp. 279–287.
- [23] F. Aurenhammer and R. Klein, Voronoi diagrams, in: *Handbook of Computational Geometry* (J. E. Goodman and J. O’Rourke, eds.), Elsevier Science Publishers, Amsterdam, 2000, pp. 201–290.
- [24] U. Bauer, X. Ge, and Y. Wang, Measuring distance between reeb graphs, *Proc. 30th Annu. ACM Sympos. Comput. Geom.*, 2014, pp. 464–473.

- [25] M. Ben-Or, Lower bounds for algebraic computation trees, *Proc. 15th Annu. ACM Sympos. Theory Comput.*, 1983, pp. 80–86.
- [26] M. d. Berg and M. J. v. Kreveld, Trekking in the alps without freezing or getting tired, *Proc. 1st Annu. European Sympos. Algs.*, 1993, pp. 121–132.
- [27] T. Bernecker, T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Zuefle, A novel probabilistic pruning approach to speed up similarity queries in uncertain databases, *Proc. 27th IEEE Int. Conf. Data Eng.*, 2011, pp. 339–350.
- [28] G. Beskales, M. A. Soliman, and I. F. Ilyas, Efficient search for the top-k probable nearest neighbors in uncertain databases, *Proc. 34th Int. Conf. Very Large Databases*, (2008), 326–339.
- [29] J. Blomer, Computing sums of radicals in polynomial time, *Proc. 32nd Annu. IEEE Sympos. Foundat. Comp. Sci.*, 1991, pp. 670–677.
- [30] S. Cabello, Approximation algorithms for spreading points, *J. Algorithms*, 62 (2007), 49–73.
- [31] S. Cabello and M. J. van Kreveld, Approximation algorithms for aligning points, *Proc. 19th Annu. ACM Sympos. Comput. Geom.*, 2003, pp. 20–28.
- [32] L. Cai and J. M. Keil, Computing visibility information in an inaccurate simple polygon, *Int. J. Comput. Geometry Appl.*, 7 (1997), 515–538.
- [33] H. Carr, J. Snoeyink, and U. Axen, Computing contour trees in all dimensions, *Proc. 11th Annu. ACM-SIAM Sympos. Discrete Algs.*, 2000, pp. 918–926.
- [34] T. M. Chan, Optimal output-sensitive convex hull algorithms in two and three dimensions, *Discrete Comput. Geom.*, 16 (1996), 361–368.
- [35] T. M. Chan, Low-dimensional linear programming with violations, *SIAM J. Comput.*, 34 (2005), 879–893.
- [36] B. Chazelle, On the convex layers of a planar set, *IEEE Trans. Inform. Theory*, 31 (1985), 509–517.
- [37] B. Chazelle, An algorithm for segment-dragging and its implementation, *Algorithmica*, 3 (1988), 205–221.
- [38] B. Chazelle, Cutting hyperplanes for divide-and-conquer, *Discrete Comput. Geom.*, 9 (1993), 145–158.

- [39] B. Chazelle, An optimal convex hull algorithm in any fixed dimension, *Discrete Comput. Geom.*, 10 (1993), 377–409.
- [40] B. Chazelle and L. J. Guibas, Fractional cascading: I. a data structuring technique, *Algorithmica*, 1 (1986), 133–162.
- [41] B. Chazelle, L. J. Guibas, and D. T. Lee, The power of geometric duality, *BIT*, 25 (1985), 76–90.
- [42] B. Chazelle and J. Matoušek, Derandomizing an output-sensitive convex hull algorithm in three dimensions, *Comput. Geom.*, 5 (1995), 27–32.
- [43] L. Chen and X. Lian, *Query Processing over Uncertain Databases*, Morgan & Claypool Publishers, 2012.
- [44] R. Cheng, J. Chen, M. Mokbel, and C. Chow, Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data, *Proc. 24th IEEE Int. Conf. Data Eng.*, 2008, pp. 973–982.
- [45] R. Cheng, L. Chen, J. Chen, and X. Xie, Evaluating probability threshold k -nearest-neighbor queries over uncertain data, *Proc. 12th Int. Conf. Ext. Database Tech.*, 2009, pp. 672–683.
- [46] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, Querying imprecise data in moving object environments, *IEEE Trans. Know. Data Eng.*, 16 (2004), 1112–1127.
- [47] R. Cheng, X. Xie, M. L. Yiu, J. Chen, and L. Sun, UV-diagram: A Voronoi diagram for uncertain data, *Proc. 26th IEEE Int. Conf. Data Eng.*, 2010, pp. 796–807.
- [48] K. L. Clarkson, Nearest-neighbor searching and metric space dimensions, *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, (2006), 15–59.
- [49] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, 3rd edition, 2009.
- [50] G. Cormode and A. McGregor, Approximation algorithms for clustering uncertain data, *Proc. 27th ACM Sympos. Principles Database Syst.*, 2008, pp. 191–200.

- [51] N. N. Dalvi, C. Ré, and D. Suciu, Probabilistic databases: Diamonds in the dirt, *Commun. ACM*, 52 (2009), 86–94.
- [52] M. de Berg, O. Cheong, M. van Kreveld, and M. H. Overmars, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 3rd edition, 2008.
- [53] T. K. Dey, Improved bounds for planar k -sets and related problems, *Discrete Comput. Geom.*, 19 (1998), 373–382.
- [54] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan, Making data structures persistent, *J. Comput. Syst. Sci.*, 38 (1989), 86–124.
- [55] R. Durier and C. Michelot, Geometrical properties of the Fermat-Weber problem, *Euro. J. Operational Research*, 20 (1985), 332–343.
- [56] J. Eckhoff, Helly, radon, and Carathéodory type theorems, in: *Handbook of Convex Geometry* (P. M. Gruber and J. M. Wills, eds.), North-Holland, 1993, pp. 389–448.
- [57] H. Edelsbrunner, J. O’Rourke, and R. Seidel, Constructing arrangements of lines and hyperplanes with applications, *Proc. 24th Annu. IEEE Sympos. Foundat. Comp. Sci.*, 1983, pp. 83–91.
- [58] E. Ezra and W. Mulzer, Convex hull of points lying on lines in $o(n \log n)$ time after preprocessing, *Comput. Geom.*, 46 (2013), 417–434.
- [59] G. Foody and P. Atkinson, eds., *Uncertainty in Remote Sensing and GIS*, Wiley, Chichester, 2002.
- [60] Y. Gao, L. Shou, K. Chen, and G. Chen, Aggregate farthest-neighbor queries over spatial data, *Proc. 16th Int. Conf. Database Syst. Adv. App.*, 2011, pp. 149–163.
- [61] R. L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, *Inform. Process. Lett.*, 1 (1972), 132–133.
- [62] C. Gray, *Shortest paths on uncertain terrains*, M.S. Thesis, Department of Computer Science, University of British Columbia, 2004.
- [63] C. Gray and W. S. Evans, Optimistic shortest paths on uncertain terrains, *Proc. 16th Canad. Conf. Comput. Geom.*, 2004, pp. 68–71.

- [64] S. Guha and K. Munagala, Exceeding expectations and clustering uncertain data, *Proc. 28th ACM Sympos. Principles Database Syst.*, 2009, pp. 269–278.
- [65] L. Guibas, J. Hershberger, and J. Snoeyink, Compact interval trees: a data structure for convex hulls, *Proc. 1st Annu. ACM-SIAM Sympos. Discrete Algs.*, 1990, pp. 169–178.
- [66] L. J. Guibas, D. Salesin, and J. Stolfi, Constructing strongly convex approximate hulls with inaccurate primitives, *Algorithmica*, 9 (1993), 534–560.
- [67] D. Günther, J. Salmon, and J. Tierny, Mandatory critical points of 2D uncertain scalar fields, *Computer Graphics Forum*, Vol. 33, 2014.
- [68] P. Gupta, R. Janardan, and M. Smid, Algorithms for generalized halfspace range searching and other intersection searching problems, *Comput. Geom. Theory Appl.*, 5 (1996), 321–340.
- [69] A. Guttman, R-trees: a dynamic index structure for spatial searching, *Proc. ACM SIGMOD Int. Conf. Management of Data*, 1984, pp. 47–57.
- [70] S. Har-Peled, *Geometric Approximation Algorithms*, Amer. Math. Soc., 2011.
- [71] S. Har-Peled and N. Kumar, Down the rabbit hole: robust proximity search and density estimation in sublinear space, *SIAM J. Comput.*, 43 (2014), 1486–1511.
- [72] D. Haussler and E. Welzl, epsilon-nets and simplex range queries, *Discrete Comput. Geom.*, 2 (1987), 127–151.
- [73] M. Hua, J. Pei, W. Zhang, and X. Lin, Ranking queries on uncertain data: a probabilistic threshold approach, *Proc. ACM SIGMOD Int. Conf. Management of Data*, 2008, pp. 673–686.
- [74] D. P. Huttenlocher, K. Kedem, and M. Sharir, The upper envelope of Voronoi surfaces and its applications, *Discrete Comput. Geom.*, 9 (1993), 267–291.
- [75] J. Jesters, G. Cormode, F. Li, and K. Yi, Semantics of ranking queries for probabilistic data, *IEEE Trans. Know. Data Eng.*, 23 (2011), 1903–1917.
- [76] M. Jooyandeh, A. Mohades, and M. Mirzakhah, Uncertain Voronoi diagram, *Inform. Process. Lett.*, 109 (2009), 709–712.
- [77] A. Jørgensen, M. Löffler, and J. Phillips, Geometric computations on indecisive points, *Proc. 12th Workshop Algs. Data Struct.*, 2011, pp. 536–547.

- [78] J. B. Kadane, *Principles of Uncertainty*, Chapman & Hall/CRC Texts in Statistical Science Series. CRC Press, 2011.
- [79] P. Kamousi, T. Chan, and S. Suri, Stochastic minimum spanning trees in euclidean spaces, *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, 2011, pp. 65–74.
- [80] P. Kamousi, T. M. Chan, and S. Suri, Closest pair and the post office problem for stochastic points, *Proc. 12th Workshop Algs. Data Struct.*, 2011, pp. 548–559.
- [81] M. J. Katz, 3-D vertical ray shooting and 2-D point enclosure, range searching, and arc shooting amidst convex fat objects, *Comput. Geom. Theory Appl.*, 8 (1997), 299–316.
- [82] Y. Kholondyrev and W. Evans, Optimistic and pessimistic shortest paths on uncertain terrains, *Proc. 19th Canad. Conf. Comput. Geom.*, 2007, pp. 197–200.
- [83] D. G. Kirkpatrick and R. Seidel, The ultimate planar convex hull algorithm?, *SIAM J. Comput.*, 15 (1986), 287–299.
- [84] M. Kraus, Visualization of uncertain contour trees, *Proc. Int. Conf. Imaging Theory Appl.*, 2010, pp. 132–139.
- [85] H.-P. Kriegel, P. Kunath, and M. Renz, Probabilistic nearest-neighbor query on uncertain objects, *Proc. 12th Int. Conf. Database Syst. Adv. App.*, 2007, pp. 337–348.
- [86] F. Li, B. Yao, and P. Kumar, Group enclosing queries, *IEEE Trans. Know. Data Eng.*, 23 (2011), 1526–1540.
- [87] H. Li, H. Lu, B. Huang, and Z. Huang, Two ellipse-based pruning methods for group nearest neighbor queries, *Proc. 13th Annu. ACM Int. Workshop Geo. Info. Syst.*, 2005, pp. 192–199.
- [88] Y. Li, F. Li, K. Yi, B. Yao, and M. Wang, Flexible aggregate similarity search, *Proc. ACM SIGMOD Int. Conf. Management of Data*, 2011, pp. 1009–1020.
- [89] Y. Li, P. M. Long, and A. Srinivasan, Improved bounds on the sample complexity of learning., *J. Comput. Syst. Sci.*, 62 (2001), 516–527.
- [90] X. Lian and L. Chen, Probabilistic group nearest neighbor queries in uncertain databases, *IEEE Trans. Know. Data Eng.*, 20 (2008), 809–824.

- [91] V. Ljosa and A. K. Singh, APLA: Indexing arbitrary probability distributions, *Proc. 23rd IEEE Int. Conf. Data Eng.*, 2007, pp. 946–955.
- [92] M. Löffler, *Data Imprecision in Computational Geometry*, Ph.D. Thesis, Dept. Computer Sci., 2009.
- [93] M. Löffler and M. J. van Kreveld, Largest bounding box, smallest diameter, and related problems on imprecise points, *Comput. Geom.*, 43 (2010), 419–433.
- [94] Y. Luo, H. Chen, K. Furuse, and N. Ohbo, Efficient methods in finding aggregate nearest neighbor by projection-based filtering, *Proc. 12th Int. Conf. Comput. Sci. and Its Appl.*, 2007, pp. 821–833.
- [95] J. Matoušek, Computing the center of planar point sets, in: *Computational Geometry: Papers from the DIMACS Special Year* (J. E. Goodman, R. Pollack, and W. Steiger, eds.), Amer. Math. Soc., 1991, pp. 221–230.
- [96] J. Matoušek, Reporting points in halfspaces, *Comput. Geom.*, 2 (1992), 169–186.
- [97] J. Matoušek and O. Schwarzkopf, Linear optimization queries, *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, 1992, pp. 16–25.
- [98] M. Mihai and R. Westermann, Visualizing the stability of critical points in uncertain scalar fields, *Computers & Graphics*, 41 (2014), 13 – 25.
- [99] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.
- [100] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, Group nearest neighbor queries, *Proc. 20th IEEE Int. Conf. Data Eng.*, 2004, pp. 301 – 312.
- [101] D. Papadias, Y. Tao, G. Fu, and B. Seeger, Progressive skyline computation in database systems, *ACM Trans. Database Syst.*, 30 (2005), 41–82.
- [102] D. Papadias, Y. Tao, K. Mouratidis, and C. Hui, Aggregate nearest neighbor queries in spatial databases, *ACM Trans. Database Syst.*, 30 (2005), 529–576.
- [103] A. N. Papadopoulos, *Nearest Neighbor Search: A Database Perspective*, Springer Verlag, 1st edition, 2010.
- [104] P. Pérez-Lantero, Area and perimeter of the convex hull of stochastic points, *CoRR*, abs/1412.5153 (2014).

- [105] J. Phillips, *Small and Stable Descriptors of Distributions for Geometric Statistical Problems*, Ph.D. Thesis, Dept. Computer Sci., 2009.
- [106] F. P. Preparata and S. J. Hong, Convex hulls of finite sets of points in two and three dimensions, *Commun. ACM*, 20 (1977), 87–93.
- [107] D. Salesin, J. Stolfi, and L. Guibas, Epsilon geometry: Building robust algorithms from imprecise computations, *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, 1989, pp. 208–217.
- [108] N. Sarnak and R. E. Tarjan, Planar point location using persistent search trees, *Commun. ACM*, 29 (1986), 669–679.
- [109] R. Seidel, Constructing higher-dimensional convex hulls at logarithmic cost per face, *Proc. 18th Annu. ACM Sympos. Theory Comput.*, 1986, pp. 404–413.
- [110] R. Seidel, Convex hull computations, in: *Handbook of Discrete and Computational Geometry* (J. E. Goodman and J. O’Rourke, eds.), CRC Press, 2004, pp. 495–512.
- [111] J. Sember and W. Evans, Guaranteed Voronoi diagrams of uncertain sites, *Proc. 20th Canad. Conf. Comput. Geom.*, 2008.
- [112] M. Sharifzadeh and C. Shahabi, Vor-tree: R-trees with Voronoi diagrams for efficient processing of spatial nearest neighbor queries, *Proc. 36th Int. Conf. Very Large Databases*, 3 (2010), 1231–1242.
- [113] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, 1995.
- [114] M. Soliman, I. Ilyas, and K.-C. Chang, Top-k query processing in uncertain databases, *Proc. 23rd IEEE Int. Conf. Data Eng.*, 2007, pp. 896–905.
- [115] S. Suri and K. Verbeek, On the most likely Voronoi diagram and nearest neighbor searching, *Proc. 25th Int. Sympos. Algs. and Comput.*, 2014, pp. 338–350.
- [116] S. Suri, K. Verbeek, and H. Yıldız, On the most likely convex hull of uncertain points, *Proc. 21st Annu. European Sympos. Algs.*, 2013, pp. 791–802.
- [117] S. P. Tarasov and M. N. Vyalyi, Construction of contour trees in 3D in $O(n \log n)$ steps, *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, 1998, pp. 68–75.

- [118] G. Trajcevski, R. Tamassia, H. Ding, P. Scheuermann, and I. F. Cruz, Continuous probabilistic nearest-neighbor queries for uncertain trajectories, *Proc. 12th Int. Conf. Ext. Database Tech.*, 2009, pp. 874–885.
- [119] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore, Contour trees and small seed sets for isosurface traversal, *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997, pp. 212–220.
- [120] M. J. van Kreveld, M. Löffler, and J. S. B. Mitchell, Preprocessing imprecise points and splitting triangulations, *SIAM J. Comput.*, 39 (2010), 2990–3000.
- [121] V. N. Vapnik and A. Y. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities, *Theory Probab. Appl.*, 16 (1971), 264–280.
- [122] M. Vazirgiannis, M. Halkidi, and D. Gunopulos, *Uncertainty Handling and Quality Assessment in Data Mining*, Springer Publishing Company, Incorporated, 2012.
- [123] H. Wang, Aggregate-Max nearest neighbor searching in the plane, *arXiv:1309.1807v1*, (2013). A preliminary version appeared in *Proc. of CCCG 2013*.
- [124] H. Wang and W. Zhang, The L1 nearest neighbor searching with uncertain queries, *CoRR*, abs/1211.5084 (2012).
- [125] H. Wang and W. Zhang, The τ -skyline for uncertain data, *Proc. 26th Canad. Conf. Comput. Geom.*, 2014, pp. 326–331.
- [126] M. Yiu, N. Mamoulis, and D. Papadias, Aggregate nearest neighbor queries in road networks, *IEEE Trans. Know. Data Eng.*, 17 (2005), 820 – 833.
- [127] S. M. Yuen, Y. Tao, X. Xiao, J. Pei, and D. Zhang, Superseding nearest neighbor search on uncertain spatial databases, *IEEE Trans. Know. Data Eng.*, 22 (2010), 1041–1055.
- [128] P. Zhang, R. Cheng, N. Mamoulis, M. Renz, A. Zufire, Y. Tang, and T. Emrich, Voronoi-based nearest neighbor search for multi-dimensional uncertain databases, *Proc. 29th IEEE Int. Conf. Data Eng.*, 2013, pp. 158–169.
- [129] Z. Zhao, D. Yan, and W. Ng, A probabilistic convex hull query tool, *Proc. 15th Int. Conf. Ext. Database Tech.*, 2012, pp. 570–573.

Biography

Wuzhou Zhang was born on August 3rd, 1988 in Zhoukou, Henan, China. He earned a Bachelor of Engineering in Computer Science from Wuhan University, China, in June 2010, a Master of Science in Computer Science en route to the Ph.D. from Duke University, United States, in May 2012, and a Doctor of Philosophy in Computer Science from Duke University, United States, in May 2015.

Wuzhou was placed 2nd for the data challenge task — Graph De-anonymization — at the Sixth ACM International Conference on Web Search and Data Mining (WSDM) in Rome, Italy, 2013. He spent a summer as a research intern at Microsoft Research Silicon Valley, Mountain View, California, 2013, and a summer as a software engineering intern at Facebook, Menlo Park, California, 2014. Wuzhou has an Erdős number of 2 through Boris Aronov, who has collaborated with Paul Erdős. Wuzhou has also coauthored a paper on τ -skyline [125] with Haitao Wang, which did not contribute to this thesis.