Efficient geometric algorithms for preference top-k queries, stochastic line arrangements, and proximity problems

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Yuan Li

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Advisor: Prof. Ravi Janardan

June, 2017

Acknowledgements

There are numerous people I would like to express my great gratitude for their contributions over these years.

First, my deepest gratitude goes to my advisor Professor Ravi Janardan. This dissertation would not be possible without his continuous support and guidance. As an amazing mentor, Professor Janardan always enlightened and encouraged me with patience. I greatly appreciate the countless time and effort he spent on me, discussing problems, brainstorming new ideas, and revising manuscripts. I have also learned a lot from his extreme self-discipline, which I believe would be tremendously beneficial for my future career.

I would also like to thank Professor John Gunnar Carlsson, Professor Volkan Isler, Professor George Karypis, Professor Mohamed F. Mokbel, and Professor Victor Reiner for serving as my committee members and for providing me valuable feedback and advice.

My sincere thanks go to my lab mates for their friendship, support, and continuous brainstorming and discussions. In particular, I am grateful to Akash Agrawal, Rahul Saladi, and Jie Xue. I also appreciate the chance to collaborate with Professor Ahmed Eldawy. In addition, I want to express my thanks and wishes for all the friends I met at the University of Minnesota in my entire Ph.D. life. Their help and support are much appreciated.

Finally, I would like to thank the Department of Computer Science and Engineering at the University of Minnesota for the generous funding support over the years.

Dedication

To my parents, Weixiang Li and Jianhe Zhang.

Abstract

Problems arising in diverse real-world applications can often be modeled by geometric objects such as points, lines, and polygons. The goal of this dissertation research is to design efficient algorithms for such geometric problems and provide guarantees on their performance via rigorous theoretical analysis. Three related problems are discussed in this thesis.

The first problem revisits the well-known problem of answering preference top-k queries, which arise in a wide range of applications in databases and computational geometry. Given a set of n points, each with d real-valued attributes, the goal is to organize the points into a suitable data structure so that user preference queries can be answered efficiently. A query consists of a d-dimensional vector w, representing a user's preference for each attribute, and an integer k, representing the number of data points to be retrieved. The answer to a query is the k highest-scoring points relative to w, where the score of a point, p, is designed to reflect how well it captures, in aggregate, the user's preferences for the different attributes. This thesis contributes efficient exact solutions in low dimensions (2D and 3D), and a new sampling-based approximation algorithm in higher dimensions.

The second problem extends the fundamental geometric concept of a line arrangement to stochastic data. A line arrangement in the plane is a partition of the plane into vertices, edges, and faces. Surprisingly, diverse problems, including the preference top-k query and k-order Voronoi Diagram, essentially boil down to answering questions about the set of k-topmost lines at some abscissa. This thesis considers line arrangements in a new setting, where each line has an associated existence probability representing uncertainty that is inherent in real-world data. An upper-bound is derived on the expected number of changes in the set of k-topmost lines, taken over the entire x-axis, and a worst-case upper bound is given for $k = 1$. Also, given is an efficient algorithm to compute the most likely k-topmost lines in the arrangement. Applications of this problem including the most likely Voronoi Diagram in $\mathbb{R}^1$ and stochastic preference top-k query are discussed.

The third problem discussed is geometric proximity search in both the stochastic setting and the query-retrieval setting. Under the stochastic setting, the thesis considers two fundamental problems, namely, the stochastic closest pair problem and the k most likely nearest neighbor search. In both problems, the data points are assumed to lie on a tree embedded in $\mathbb{R}^2$ and distances are measured along the tree (a so-called tree space). For the former, efficient solutions are given to compute the probability that the closest pair distance of a realization of the input is at least ℓ and to compute the expected closest pair distance. For the latter, the thesis generalizes the concept of most likely Voronoi Diagram from $\mathbb{R}^1$ to tree space and bounds its combinatorial complexity. A data structure for the diagram and an algorithm to construct it are also given.

For the query-retrieval version which is considered in $\mathbb{R}^2$, the goal is to retrieve the closest pair within a user-specified query range. The contributions here include efficient data structures and algorithms that have fast query time while using linear or near-linear space for a variety of query shapes. In addition, a generic framework is presented, which returns a closest pair that is no farther apart than the closest pair in a suitably shrunken version of the query range.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Consider the following questions that are typical of many modern-day applications: How might a tourist in a large city use her smartphone to identify hotels that meet her preferences in terms of cost, convenience, ratings, and safety? How should a climate scientist interpret data gathered from a collection of sensors if there is uncertainty in their precise locations and/or in their level of activity due to ambient conditions? How might a traffic control center keep monitoring real-time vehicle positions to detect or predict potential traffic collisions in hot-spot areas of a city, where hot-spots are changing dynamically on an hourly basis?

While diverse in nature, these questions can be unified under the umbrella of geometric computing. For example, hotels can be modeled as points and user preferences as query vectors. Sensors can be modeled as polygons with an associated probability density function to model potential locations; or, if the sensor locations are known precisely, then as points with associated existence probabilities. Vehicles can be modeled as points in the road map, and the minimum Euclidean distance between any pair of points can be a reasonable measure to the traffic density. Also, certain areas on the map, hot-spots for instance, are often specified by the user as query regions, and the goal is to compute quantities of interest (e.g., closest pair distance) relative to the data objects lying in the query region.

Given the huge volume and sheer diversity of the datasets generated by modern applications, it is imperative to develop algorithms that can model and process the underlying geometric representation efficiently. This dissertation aims to develop efficient

algorithms for fundamental geometric problems that are motivated by practical applications. The goal is to obtain algorithms that are efficient with respect to both run time and the amount of storage used, as demonstrated by formal theoretical analysis.

1.1 Problem motivation and statement

In this thesis we investigate three problems. They are the *preference top-k query*, the *stochastic line arrangement*, and *geometric proximity search*. In the remainder of this section, we discuss these problems in more details.

1.1.1 Preference top-k query

An important requirement of a database query engine is that it allows users to perform queries that retrieve a small amount of relevant information from a potentially large search space, where the information is tailored to the individual preferences of each user.

For example, consider a real-estate database that contains information about thousands of houses for sale in a major metropolitan area, such as (say) Chicago. The information might include attributes such as asking price, age, number of bedrooms, distance to nearest school, etc. Prospective buyers are interested in extracting from this dataset only a small subset of (say) ten houses meeting their criteria that can be further researched, rather than search through all of the information in the database. Furthermore, different buyers often have different levels of preferences (i.e., weights) for the associated attributes and each buyer will want to retrieve only the ten houses that score highest on a (linear) combination of the attributes based on his/her preferences. (The ability to perform such queries is all the more crucial if buyers are interrogating the database on mobile devices, as these tend to have small screen sizes and bandwidth limitations.) Other examples of where such queries arise include students wishing to rank colleges based on tuition, graduation rate, enrollment, etc.; shoppers using a recommender system at an online retailer to buy a product (e.g., choosing a laptop based on price, CPU speed, memory, and weight); ecologists grading nature preserves based on amount of water, elevation, diversity of flora and fauna; and so on.

Based on the above observations, the underlying database should have the ability to

answer a so-called *preference top-k query*, i.e., given a set of multidimensional objects (where the dimension is the number of attributes), retrieve the k best objects with respect to the preferences given by the user. See Figure 1.1.1 for a quick example of the preference top-1 query for a visitor to select her desired hotel near the campus with respect to her preference on the attributes of price and distance.

Furthermore, a user may also wish to restrict the query to a subset of the dataset, by specifying a range for each attribute, and retrieve the top-k objects in the restricted subset. For example, a user may be interested in houses in a certain neighborhood or in laptops in a certain price and weight range.

As another extension, sometimes it may be difficult for a user to specify preferences exactly. It is more reasonable to specify preferences "fuzzily", as a set of several (possibly infinitely-many) candidate preferences. For instance, a fuzzy preference in 2D can be given as an interval of angles, and all the preferences that are inside this interval are candidates. Based on this fuzzy preference, the score of each object is redefined as the best (minimum or maximum) linear combination with respect to any preference in the given region, and the database should output the top-k objects according to this score.

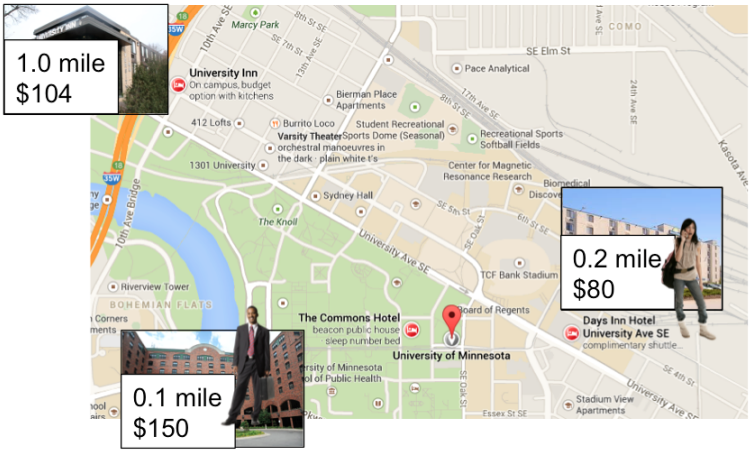


Figure 1.1: Illustrating the preference top-1 query. The points are hotels near the UMN campus, and the attributes for each hotel are the distance to the campus and the room rate. A visitor might prefer being closer to campus for convenience or may prefer a lower price for financial reasons, which results in different query results. (Figures in the thesis are best viewed in color.)

1.1.2 Stochastic line arrangement in $\mathbb{R}^2$

A problem defined on points in the xy-plane can be mapped, via a certain geometric transformation, to an equivalent dual problem defined on a set of lines in the plane. These lines induce a partition of the plane into vertices, edges, and faces; this partition is called an *arrangement*. The dual problem is sometimes easier to handle because, intuitively, "non-local" properties in the primal problem become "local" in the dual. For example, for the preference top-k problem, points (e.g., hotels) with two attributes map to lines in the plane. The user's preference vector w can be shown to map to a vertical line. Furthermore, the top-k points relative to w happen to be exactly the topmost k lines that are intersected by this vertical line. As another example, Voronoi Diagrams are used widely in operations research to solve proximity problems, e.g., finding the facility closest to a query location. Remarkably, it turns out that the Voronoi-based framework can be mapped (using a different transformation) to the problem of finding the topmost line in a certain arrangement that is intersected by a vertical line.

Many such diverse problems essentially boil down to answering the following questions efficiently: "In a given line arrangement, what are the topmost k lines intersected by some vertical line with x-coordinate q? Moreover, as the line sweeps over the arrangement from left to right, how many times does the set of topmost k lines change?" Owing to their many applications, these questions have been investigated extensively and have been well-settled.

We investigate this problem in a new setting, where the lines are stochastic. That is, the lines do not exist with certainty but instead each line, f_i, has an associated existence probability p_i that is inherited from the underlying primal point. Such stochasticity arises naturally, due to noise or imprecision, when data is gathered in the real world using GPS, sensors or other probabilistic systems or measurements.

For a given vertical line with x-coordinate q, the likelihood associated with the topmost k lines that exist at q can be extremely small and is thus not very meaningful. Instead, it is more relevant to compute, for a given q, the set of k lines that have the greatest likelihood of being the topmost, i.e., the *most-likely* k-topmost lines. However, data uncertainty often invalidates many of the known results for conventional (i.e., non-stochastic) arrangements and makes the corresponding problems far more difficult. Indeed, it turns out that any k lines (even those at the very bottom) could be most-likely

k-topmost lines. (See Figure 1.2.) Also, unlike the conventional case, as the vertical line sweeps from left to right over the entire x-axis, the most likely k-topmost lines w.r.t. q can change arbitrarily with no apparent pattern, which complicates the situation. Therefore, we study the underlying combinatorial complexity of the line arrangement under uncertainty in both the worst case and the average case. Efficient algorithms for computing the most-likely k-topmost lines over the entire line arrangement are also useful to solve related problems in the stochastic setting (via duality).

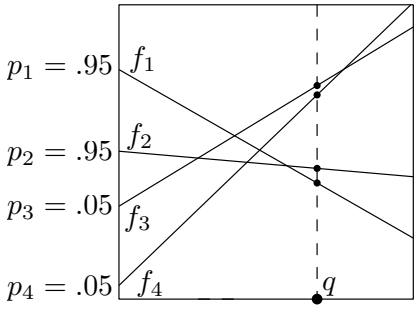


Figure 1.2: Illustrating the line arrangement of four stochastic lines in the plane. The conventional 2-topmost lines w.r.t. x-coordinate q are clearly f_3 and f_4, but they only have $.05^2 = .0025 \approx 0$ probability to be present. However, the probability for lines f_2 and f_1 to be the true 2-topmost lines at q is $.95^2 \times .95^2 \approx .815$, i.e., f_2 and f_1 must be present, and f_3 and f_4 must not be present. Obviously, the latter likelihood is significantly larger, even though at q f_2 and f_1 are below other lines.

As mentioned earlier, there is a close connection between line arrangements and the preference top-k problem. We now elaborate on this briefly by showing a link between stochastic line arrangements and a stochastic version of the preference top-k problem. In the latter problem, we are given a set of data points in the plane, where each input data point h_i (say a hotel) has fixed attributes and a probability p_i related to (say) its user rating. (A hotel with a low rating has low probability to be visited by a tourist.) The goal is to retrieve the most-likely top-k hotels corresponding to a user's preference vector. This problem can be modeled, via the aforementioned duality transform, as the stochastic line arrangement problem and can hence be solved by retrieving the most-likely 2-topmost lines at the position corresponding to the user's preference in the dual space.

On the other hand, as we shall see later, this result can also be used to solve the

so-called stochastic Voronoi Diagram problem in $\mathbb{R}^1$ and hence the most likely nearest neighbor search in $\mathbb{R}^1$. However, further generalization of this idea to higher dimensions (even to $\mathbb{R}^2$), while still preserving good theoretical bounds, appears to be very challenging. Thus, we consider a special, but natural, version of this problem where the input points are constrained to be in a so-called tree space. We elaborate more below.

1.1.3 Stochastic closest-pair problem and most-likely nearest-neighbor search in tree space

The closest-pair problem and nearest-neighbor search are two interrelated fundamental problems, which have numerous applications. The uncertain versions of both the problems have also been studied recently in [6, 41, 44, 46, 58] and have generated significant interest in the database and data structures communities.

Let S be a set of n stochastic points in some metric space $\mathcal{X}$. For the closest pair problem, a basic question one may ask is how to compute elementary statistics about the stochastic closest-pair (SCP) of S, e.g., the probability that the closest-pair distance of a realization of S is at least ℓ, or the expected closest-pair distance, etc. Unfortunately, most problems of this kind have been shown to be NP-hard or #P-hard for general metrics, and some of them remain #P-hard even when $\mathcal{X} = \mathbb{R}^d$ for $d \geq 2$ [41, 44]. For nearest-neighbor search, an important problem is to find the most-likely nearest-neighbor (LNN) [58], i.e., the data point in S with the greatest probability of being the nearest-neighbor of a query point q. The LNN search introduces the concept of most-likely Voronoi diagram (LVD), which decomposes $\mathcal{X}$ into connected cells such that the query points in the same cell have the same LNN. However, as in [46, 58], the size of LVD in $\mathbb{R}^d$ is high even on average. Due to the difficulties of both problems in general and Euclidean space, it is then natural to ask whether these problems are relatively easier in other metric spaces such as a *tree space*. Informally, a tree space consists of an edge-weighted tree embedded in the plane. Each point of S is constrained to lie somewhere in the tree and distances are measured along the tree. (See Figure 1.3 for a simple example of a tree space. Formal definitions will be given later.) Indeed, further exploring these problems in tree space will be helpful and interesting since any finite metric (say a road network in practice) can be embedded in a tree space under some reasonable distortions [34]. With the above motivations, we study the stochastic

closest-pair (SCP) problem and k most-likely nearest-neighbor (k-LNN) search in tree space, where use the same uncertainty model as before, that is, each stochastic input point has a fixed location (in the tree space) with an associated (independent) existence probability.

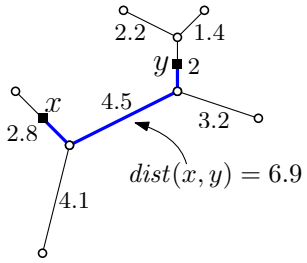


Figure 1.3: An example of a tree space with two stochastic points x and y in it. Since here x and y are at the midpoints of edges, the distance between them is $1.4+4.5+1 = 6.9$.

1.1.4 Range closest pair search in $\mathbb{R}^2$

The conventional closest pair search problem has many applications in collision detection, similarity search, traffic control, and so on; see the survey [57] for a collection of related topics. However, in many cases, it is too expensive and also unnecessary to compute the closest pair with respect to the entire data set. Instead, it is more useful to zoom into a smaller region of interest and compute the closest pair only for the points in that region. For instance, consider a scenario where one wishes to monitor traffic patterns and potential collisions/near-misses in a large city. Instead of computing the closest pair information for all vehicles in the entire city in real-time, it is better to run the algorithm on the hot-spot areas only. As another example mentioned in [55], VLSI designers often need to zoom in to a sub-screen of the VLSI layout editor and check whether certain features violate the separation rule (i.e., they are too close). This again reduces to the problem of finding the closest pair in a certain range if we treat the features of interest as points in the plane. Both examples motivate the so-called *range closest pair search* problem, which has drawn a lot of attention recently.

In this thesis, we revisit this topic (in $\mathbb{R}^2$) and study the exact and approximate solutions for a variety of query shapes. For each type of query, our goal is to design

efficient data structures and algorithms that have fast query time while using linear or near-linear space. (We pay more attention to the storage and the query time as the former is permanent and the latter must be real-time, whereas the preprocessing time is of less concern as it is a one-time cost.)

We remark that while both sets of problems in this subsection pertain to geometric proximity, they are investigated in different settings. The first set addresses proximity questions for stochastic points in the tree space whereas the second set considers proximity problem in query-retrieval mode in $\mathbb{R}^2$. The latter sets the stage for future work on stochastic query-retrieval problems.

1.2 Related work

The preference top-k query problem is studied in a multimedia context [31, 32]. Subsequently, the so-called *Threshold Algorithm* (TA) is given in [33] which works for not only linear preferences but also other monotone preference functions (i.e., the preferences are represented as monotone functions on the attributes rather than as weight vectors). TA maintains a threshold value to help prune a lot of data points with low scores and terminate the algorithm early, which allows it to work much more efficiently than the naïve (brute-force) algorithm in most cases. Another work known as *Onion Index* [18] is based on the notion of convex layers. Besides the Onion Index, other indexing methods have also been studied recently, which include *Robust Index* [61] and *Cube Index* [22]. Recently, an efficient algorithm for the 3D half-space range reporting problem has been established in [4]. This algorithm can also be used to solve, within the same bounds, the preference top-k problem in 3D by transforming the problem to its dual version. For a good survey of the top-k problem, please see [42].

As for the approximation algorithms, in the last several decades, many general approximate query processing techniques such as [15, 16, 19] have been proposed. Some of them can also be used to do approximate preference top-k reporting. Furthermore, some exact top-k algorithms we mentioned previously, such as TA, also have approximate variants. A recent approach to sampling-based approximate preference top-k algorithms is a coreset algorithm proposed in [63]. For any accuracy requirement parameter given by the user, this algorithm can sample a corresponding small subset (called *coreset*) of

data points from the original dataset satisfying that accuracy requirement. Theoretical bounds are also given in [63] to guarantee the size of the sampling set in various dimensions.

Next, we first briefly review some of the existing work on the arrangement of lines and line levels then give a broad sampling on data uncertainty and stochastic closest pair related problems. The conventional arrangement of lines as well as the related concept of k-level/$\leq k$-level are fundamental structures that have a rich and long history. Readers can refer to [12] for a good survey about arrangements and their applications; see [5, 8, 12] for more information about k-level/$\leq k$-level and its applications. In terms of the combinatorial complexity of k-level, both the lower and upper bound are still open even in 2D; see [28, 30, 48, 60] for a chain of improvements in the past. Also, see [17] for a more detailed summary and for the bounds beyond 2D. The maximum complexity of the $\leq k$-level is precisely $\Theta(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ in $\mathbb{R}^d$, by using a probabilistic argument [56].

The topic of uncertain data has received significant attention in various areas such as computational geometry, algorithms, databases, etc. Many classical problems have been studied in stochastic settings, including convex hull [10, 47, 59], minimum spanning tree [43], range search [7, 11], linear separability [35, 62], top-k queries [23, 38], etc. See also [14] and [26] for two survey papers.

More relevantly, the stochastic versions of the closest-pair problem and nearest-neighbor search have also been investigated in [6, 9, 41, 44, 58]. Kamousi et al. [44] show that computing the ℓ-threshold probability of the closest-pair distance and some variants of the problem are #P-hard under existential uncertainty even in $\mathbb{R}^2$. The nearest-neighbor search is also considered in [44] under existential uncertainty, but the problem considered is that of finding the point minimizing the expected distance to the query point instead of the LNN. Huang et al. [41] give hardness results and randomized approximation algorithms for some stochastic closest-pair related problems under general metric. It is shown in [41] that computing the expected closest-pair distance under existential uncertainty is #P-hard in a general metric space. Agarwal et al. [6, 9] study the uncertain nearest-neighbor search, but their main focus is locational uncertainty (where each point can exist at any of several possible locations according to some probability distribution) and the problems studied are quite different from the

LNN search. Suri et al. [58] investigate the LNN search and give upper bounds for the complexity of the LVD as well as the way to construct the LVD. However, only the case of 1-LNN search in $\mathbb{R}^1$ is studied in [58]. The problem in general Euclidean space and non-Euclidean metric spaces is quite open, as is the k-LNN search.

Finally, we list several prior work on the range closest pair query in $\mathbb{R}^2$. The rectangle query was first considered by Shan et al. in [54], where they gave an R-tree based solution that performs well in practice; there is no theoretical analysis for this approach, though. The same problem with theoretical guarantee is mainly studied by [40, 55]. In [55], Sharathkumar and Gupta propose a solution with $O(\log^3 n)$ query time and $O(n \log^3 n)$ space. Gupta et al. improve the query time to $O(\log^2 n)$ in [40], but the space occupied is $O(n \log^5 n)$. They also give variety results for other query shapes under different assumptions. For halfplane queries, Abam et al. gave in [2] two solutions based on Semi-Separated Pair Decomposition. One has $O(n^{0.5+\varepsilon})$ query time using $O(n \log n)$ space and can be built in $O(n^2 \log^2 n)$ time. The other has $O(n^{3/4+\varepsilon})$ query time using $O(n \log^2 n)$ space with $O(n^{1+\varepsilon})$ preprocessing time.

1.3 Summary of contributions

Preference top-k query: We present a series of progressively more efficient exact algorithms in 2D, culminating in an optimal algorithm that uses $O(n)$ space and has query time $O(\log n + k)$. We also propose two useful extensions of the basic problem to enable the user to restrict the search (in different ways) to a user-specified subset of the dataset. In the first extension, the user's focus is on a subset of the data points (as might be the case if the user wishes to "zoom into" a small geographic location in a spatial dataset) and the goal is to identify the top-k points in this subset based on the user's preferences. In the second extension, the user's preference vector may be known only roughly (which is often the case) and the goal is to report the top-k data points under this set of fuzzy preferences. We show how to use our basic algorithm in conjunction with suitable range trees [27] and priority search trees [49] to solve each of these problems efficiently.

We consider the preference top-k query in 3D and demonstrate how to generalize our 2D algorithm, based on convex layers, efficiently to 3D via gnomonic projection [25],

planar point location [27], and an appropriate grouping of points in each layer to help speed up certain steps.

In addition, we propose a new sampling-based algorithm, in which the idea used for sampling data points is rather different from the coreset method. Our algorithm first reduces the top-k sampling task into k iterations of top-1 sampling, in which the so-called "critical detection vectors" are introduced to help judiciously sample the dataset while upper-bounding the error. Specifically, for any maximum allowable error parameter α (which is given by the user), a sampling set with a top-1 error smaller than α can be always constructed. Theoretical analysis of our sampling algorithm is given to prove that the size of the final sampling set obtained by our algorithm is well-bounded by $O(k\alpha^{-0.5})$ in 2D. Although the bounds of our algorithm are difficult to analyze in higher dimensions, experimental results on different datasets are presented to show that our algorithm works very well in dimensions 2, 3, and 4.

Stochastic line arrangement in $\mathbb{R}^2$: Given a collection of stochastic lines in $\mathbb{R}^2$, we give a formal definition of the most likely k-topmost lines at a certain x-coordinate q. We also study the combinatorial behavior of how these most likely k-topmost lines change when q moves continuously from $-\infty$ to ∞. Specifically, we show, by a concrete example, that in the worst case such a structure can change quadratic times even when $k = 1$. On the other hand, we also give a detailed combinatorial analysis proving that the expected number of changes is $O(nk)$ if the probabilities of all lines are independently drawn from any fixed probability distribution. An efficient algorithm is also designed to compute the most likely k-topmost lines of n stochastic lines over the entire x-axis in $O(n^2 \log n + nk^2 \log k)$ time, which can be directly leveraged to answer the so-called stochastic preference top-k query. Finally, as another application we also apply our results to bound the combinatorial complexity of the stochastic Voronoi Diagram in $\mathbb{R}^1$.

Stochastic closest-pair problem and most-likely nearest-neighbor search in tree space: Let $\mathcal{T}$ be a tree space represented by a t-vertex weighted tree T, and S be the given set of n stochastic points in $\mathcal{T}$ each of which is associated with an existence probability. A *realization* of S refers to a random sample of S in which each point is sampled with its existence probability.

For the SCP problem, define $\kappa(S)$ as a random variable indicating the closest-pair distance of a realization of S. We first show that the ℓ-threshold probability of $\kappa(S)$ (i.e., the probability that $\kappa(S)$ is at least ℓ) can be computed in $O(t+n\log n+\min\{tn,n^2\})$ time for any given positive threshold ℓ. Based on this, we immediately obtain an $O(t+\min\{tn^3,n^4\})$-time algorithm for computing the expected closest-pair distance, i.e., the expectation of $\kappa(S)$. We then further show that one can approximate the expected closest-pair distance within a factor of $(1+\varepsilon)$ in $O(t+\varepsilon^{-1}\min\{tn^2,n^3\})$ time, by arguing that the expected closest-pair distance can be approximated via $O(\varepsilon^{-1}n)$ threshold probability queries.

For the LNN search, we first study the size of the the k-LVD $\Psi_{\mathcal{T}}^S$ of S on $\mathcal{T}$. A matching $O(n^2)$ upper bound for the worst-case size of $\Psi_{\mathcal{T}}^S$ is given. More interestingly, we show that (1) the worst-case size of $\Psi_{\mathcal{T}}^S$ is $O(kn)$, if the existence probabilities of the points in S are constant-far from 0; and (2) the average-case size of $\Psi_{\mathcal{T}}^S$ is $O(kn)$, if the existence probabilities are i.i.d. random variables drawn from a fixed distribution. These results further imply the existence of an LVD data structure which answers k-LNN queries in $O(\log n+k)$ time using average-case $O(t+k^2n)$ space, and worst-case $O(t+k^2n)$ space if the existence probabilities of the points are constant-far from 0. Finally, we give an $O(t+n^2\log n+n^2k)$-time algorithm to construct such a data structure.

Range closest pair search: Given a set S of n points in $\mathbb{R}^2$, we show how to design efficient data structures and algorithms such that, given a query range Q, the closest pair in $S\cap Q$ can be reported quickly.

For Q being a p-sided rectangle query, compared to the existing results in [40, 55] we improve the space for quadrant and strip queries by a $\log n$ factor and, for $p\geq 2$, improve both the space and query time by $\log n$ factors if the input points satisfy a certain flatness property defined later. For Q being a halfplane, the existing result in [2] has $O(n^{0.5+\varepsilon})$ (resp. $O(n^{3/4+\varepsilon})$) query time using $O(n\log n)$ (resp. $O(n\log^2 n)$) space with $O(n^2\log^2 n)$ (resp. $O(n^{1+\varepsilon})$) preprocessing time. We improve significantly these bounds. Specifically, we show that there is an optimal solution that can answer each query in $O(\log n)$ time, using only $O(n)$ space. For Q being a radius-fixed disc, we present a solution that answers each query in $O(\log^2 n)$ time and uses $O(n\log n)$ space.

To our best of knowledge, there is no existing work with good theoretical guarantee for disc (of any radius) query.

Finally, we propose a general approximation framework for the range closest pair query. Given a query range type, the algorithm returns a closest pair that is no farther apart than the closest pair in a suitably shrunken version of the query range, where the shrinkage is controlled by a user-specified positive real ε. The framework uses two fundamental structures in computational geometry, namely, a range reporting structure and a range minimum query structure. By plugging in suitable black boxes, we can handle a variety of query shapes.

We summarize our results on range closest pair search in Table 1.1.

Type of query	Space	Query time
Quadrant	$O(n)$	$O(\log n)$
Strip	$O(n \log n)$	$O(\log n)$
Strip with $O(1)$-flat input	$O(n)$	$O(\log n)$
3-sided rectangle with $O(1)$-flat input	$O(n \log n)$	$O(\log^2 n)$
4-sided rectangle with $O(1)$-flat input	$O(n \log^3 n)$	$O(\log^2 n)$
Halfplane	$O(n)$	$O(\log n)$
Disc with fixed radius	$O(n \log n)$	$O(\log^2 n)$
Disc	$O(n^{1+\zeta})$	$O(\log^2 n + (1/\varepsilon^2) \log(1/\varepsilon))$
Fat 4-sided rectangle	$O(n \log n)$	$O(\log n + (1/\varepsilon^2) \log(1/\varepsilon))$
Fat convex shape of $O(1)$ complexity	$O(n \log^2 n)$	$O(\log^2 n + (1/\varepsilon^2) \log(1/\varepsilon))$

Table 1.1: Summary of our results. Here ζ and ε are positive reals. The first seven results correspond to exact closest pairs and the last three to approximate closest pairs.

1.4 Organization

The remainder of this thesis is organized as follows. We formulate and solve the preference top-k query in Chapter 2. Thereafter, we present our approximation solution to the same problem in Chapter 3. The stochastic line arrangement and its related applications are discussed in Chapter 4. The following two chapters are related to extensions on the proximity search. Specifically, we study in Chapter 5 the stochastic closest-pair problem and most-likely nearest-neighbor search in tree space. In Chapter 6, we revisit

the range closest pair problem. Finally, we conclude in Chapter 7 with a summary of our contributions and a discussion of directions for possible future work.

We note that for the sake of clarity in the exposition and in order to not impede the flow of the discussion, most of the proofs in Chapters 3-5 are deferred to the ends of these chapters. Also, the figures in the thesis are best viewed in color.

Chapter 2

Preference top-k query

In this chapter, we present our solution to the preference top-k problem in $\mathbb{R}^2$ and $\mathbb{R}^3$. We also discuss solutions to some extensions of the problem in $\mathbb{R}^2$.

2.1 Problem formulation

An object with d real-valued attributes can be written as a point $p = (x_1, x_2, \cdots, x_d)$ in $(\mathbb{R}^+)^d$. A weighting vector is represented as a unit vector $w = (w_1, w_2, \cdots, w_d)$, where $w_i \geq 0$ and $||w|| = 1$. We can treat w as a point on the $d-1$ dimensional unit sphere $\mathbb{S}^{d-1}$. Define the *score* of a point p with respect to a weighting (i.e., preference) vector w to be $\sum_{i=1}^{d} x_i w_i$, i.e., $p \cdot w$ if we treat p as a vector as well. In other words, the score of p is length of the projection of p on the line through w. Figure 2.1 shows a 2D case, where line l passes through p, and is normal to w. The score of p with respect to w is $||Op'||$. It is easy to see that in general, for any weighing vector w, all the points on the line l have the same score. Now assume that we have n different objects (points) $P = \{p_1, p_2, \cdots, p_n\} \subset (\mathbb{R}^+)^d$. A preference *top-k* query on P specifies a weight vector w and an integer k ($1 \leq k \leq n$) and the goal of the query is to report the k objects of P that have the highest score w.r.t. w.

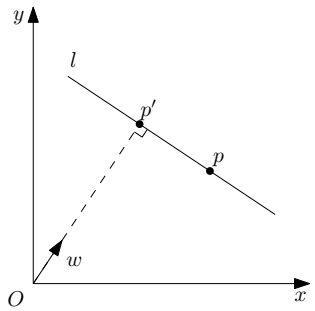


Figure 2.1: The score of point p w.r.t. weight vector w is $||Op'||$.

2.2 Algorithm in 2D

In this section we propose a preliminary algorithm in 2D, based on convex layers [20], which uses $O(n)$ space and has query time $O(k \log n)$. Then we improve the query time to $O(\log n + k \log k)$ by propagating information about extreme points on each layer. Finally we use the special selection technique given in [37] to get an optimal query time of $O(\log n + k)$.

The *convex layers* of P are defined as follows: The first layer is the convex hull of P. The second layer is the convex hull of the set obtained by deleting from P the points in the first layer. And so on until there are no more points left.

2.2.1 Preliminary algorithm

Consider a unit weight vector w in the 2D plane. If we sweep a line l perpendicular to w over P, from $+\infty$ to $-\infty$, the first k points that we encounter correspond to the top k objects in decreasing order of score. Based on this, the top-1 point must lie on the convex hull of P, and that point, called *extreme point*, is just at the position where l is the tangent of the hull (see Figure 2.2a). To move one step further, the candidates for the point with second largest score are p's two neighbors on the convex hull and one point inside the convex hull (see Figure 2.2b). Formally, let p be the point with rank i, based on score, and let *Cand* be the set containing all the possible candidate points with rank $i + 1$. We maintain the following invariant.

1. If p is the extreme point in the current layer, $Cand \leftarrow Cand \cup \{q, p', p''\}$, where q is the extreme point in the next inner layer, and p', p'' are p's two neighbors on its

layer (if they exist).

2. Else, $Cand \leftarrow Cand \cup \{p'\}$, where p' is p's left or right neighbor on its layer, as appropriate (if it exists).

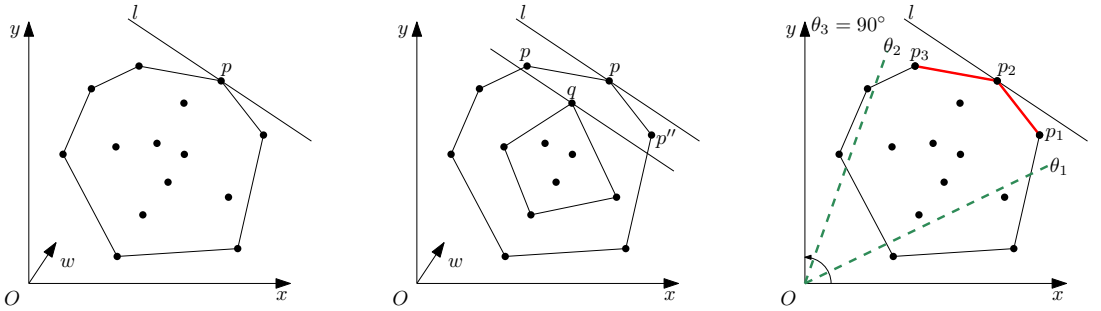
Based on this invariant, we use a binary heap to maintain all the possible candidates, and report the top-k objects one by one in non-increasing ordering of score. In case 1, we delete one element from the heap, and insert at most three elements; in case 2, we delete one element and insert one more. Hence, the size of the heap is $O(k)$, and $O(k)$ heap operations done (insertion and deletion) take $O(k \log k)$ time.

Since w lies in the first quadrant, the extreme point on each convex layer must lie between the topmost and rightmost vertex on the layer (vertices p_1, p_2, p_3 for the first layer in Figure 2.2c).

If we shoot rays which are perpendicular to each segment of this part, we partition the range of angles of all the vectors ($0° - 90°$) into several parts. For a given weighting vector, the interval in which it lies yields the corresponding extreme point. For instance, in Figure 2.2c, if the weighting vector lies in $[0, \theta_1]$, $(\theta_1, \theta_2]$, or $(\theta_2, \theta_3 = 90°]$, the corresponding extreme point will be p_1, p_2, or p_3, respectively. If the angle of weighting vector is equal to some θ_i, we can arbitrarily choose either p_i or p_{i+1} to break the tie. Hence finding the extreme point on a layer requires only one binary search, if the angles are stored in sorted order in an array, and takes $O(\log n)$ time. We only need to find at most k such points, hence the total query time is $O(k \log n + k \log k) = O(k \log n)$. The total space used is $O(n)$.

2.2.2 Applying fractional cascading

We note that finding the extreme points in different layers requires a sequence of binary searches, whose total cost is $O(k \log n)$. This cost can be reduced to $O(\log n + k \log k)$, without affecting the $O(n)$ space bound, by using the *fractional cascading* technique [21]. This technique stores appropriate pointers (called *bridges*) between consecutive arrays of angles. With this approach, we need to perform one $O(\log n)$ time binary search to find the first extreme point; subsequent extreme points are found in $O(1)$ time each by following the stored pointers. Hence, the total time improves to $O(k + \log n)$.



(a) Line l is normal to unit vector w, and l is the tangent of the convex hull. Point p on the hull ranks as the top-1 point in this situation.

(b) The candidates for the points with the second largest score can be p', p'', and the extreme point q in the inner convex hull.

(c) The extreme point can only lie in the red region. The green lines, which are perpendicular to the red segments, divide the range $[0°, 90°]$ into three intervals that can be searched for the extreme point as discussed in the text.

Figure 2.2: Illustrating the search for the top-k points for weight vector w.

Here are details. Initially we have m angle arrays, denoted by $angle[1][.]$, $angle[2][.]$, $\cdots$, $angle[m][.]$, as well as information about the corresponding nodes (extreme points) as arrays $node[1][.]$, $node[2][.]$, $\cdots$, $node[m][.]$, where m is the number of convex layers. (Layers are numbered from 1 (outermost) to m (innermost).) We will construct the fractional cascading data structure in two steps. Figure 2.3a shows an example of three convex layers, and we will use it to illustrate the construction.

Step 1: Extending arrays. First, we take every other element from $angle[m][.]$, i.e., $angle[m][1]$, $angle[m][3]$, $angle[m][5]$, $\cdots$, and merge them into $angle[m-1][.]$ in linear time. While merging, we can calculate the *node* information for each added entry very easily. Then we take every other element from the extended $angle[m-1][.]$, and merge them into angle$[m-2][.]$. We repeat the above process $m-1$ times from bottom to top to extend the array information of the first $m-1$ layers (see Figure 2.3b). Note that it is unnecessary to have identical numbers in an array, so if there exist two identical numbers after an extension, we will just keep only one of them (and the corresponding node).

Step 2: Building pointers. We start at layer 1 and proceed to layer m, as follows. For each element in current layer, we set a pointer to the smallest element (with odd

index) in the next layer which is larger than or equal to it (if such element does not exist, then we take the largest element in the next layer instead). Formally, for the j-th element in i-th layer ($1 \leq i < m$), i.e., $angle[i][j]$, we point it to $angle[i+1][j']$, where (1) j' is odd, or j' is the last element of $angle[i+1][.]$; (2) $angle[i+1][j'] \geq angle[i][j]$; and (3) $j' = 1$ or $angle[i+1][j'-2] < angle[i][j]$ (see Figure 2.3c).

We can analyze the total space used via the accounting method of amortized analysis [24]. We assign each element of each original $angle[.][.]$ array 1 credit. Throughout, we maintain the invariant that each element in the array where elements are currently being propagated has 1 credit available. This is true for $angle[m][.]$ by the above assignment. Let $angle[i][.]$ be the current array ($1 < i \leq m$) and assume that the invariant holds. Each propagated element from $angle[i][.]$ pays for itself using its stored credit and carries with it to $angle[i-1][.]$ the credit from the unpropagated neighbor to its right. Thus, the invariant holds for $angle[i-1][.]$. Hence the total number of elements propagated is upper-bounded by the number of credits assigned initially, which is $\sum s_i = O(n)$ where s_i is the original size of $angle[i][.]$. Thus the total space is $\sum s_i$ (for the original elements) $+ \sum s_i$ (for the propagated elements), which is $O(n)$. (Note that there can be at most one element in $angle[i][.]$ that does not have a neighbor on the right to borrow a credit from. There are $O(n)$ such elements in total, so this does not affect the space bound.)

Moreover, instead of performing binary search on each layer, we perform binary search only on the first layer to find the first extreme point, and then follow the pointers to the other ones. Specifically, assuming the j-th element of layer i represents the extreme point under some weighting vector w, and it points to the j'th element of layer $i+1$, then we claim that the extreme point of layer $i+1$ must be either the j'th element or the $(j'-1)$th element. For instance, assume the angle of the given vector w is $38°$. The binary search on the first layer will tell us that the element p_3, corresponding to the angle interval $(30, 45]$, is the extreme point. Then we follow its pointer to the element q_3 on the second layer, corresponding to $(30, 45]$. The angle on its left is $30° < 38°$, so q_3 is the extreme point of the second layer. This in turn points to the element r_5 on the third layer, corresponding to $(40, 45]$. However the angle to its left is $40° > 38°$, so r_4 is the extreme point on the third layer, corresponding to $(30, 40]$.

In summary, finding all the extreme points now costs only $O(\log n + k)$, thus the time for the top-k query improves to $O(\log n + k \log k)$.

layer 1	angle	10	30	45	60	70	90
	node	p_1	p_2	p_3	p_4	p_5	p_6

layer 2	angle	5	25	75	90
	node	q_1	q_2	q_3	q_4

layer 3	angle	15	20	30	40	45	80	90
	node	r_1	r_2	r_3	r_4	r_5	r_6	r_7

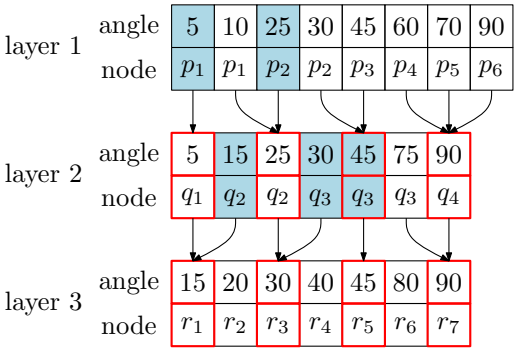
(a) Original angle and node arrays. For example, in layer 2, the numbers 5, 25, 75 and 90 represent intervals $[0, 5]$, $(5, 25]$, $(25, 75]$, $(75, 90]$ respectively, and q_1, q_2, q_3, q_4 are their corresponding extreme points.

layer 1	angle	5	10	25	30	45	60	70	90
	node	p_1	p_1	p_2	p_2	p_3	p_4	p_5	p_6

layer 2	angle	5	15	25	30	45	75	90
	node	q_1	q_2	q_2	q_3	q_3	q_3	q_4

layer 3	angle	15	20	30	40	45	80	90
	node	r_1	r_2	r_3	r_4	r_5	r_6	r_7

(b) Extending the arrays. The elements with red border are selected to be merged above, and the shaded cells are the final positions of these selected elements from the array one level below.



(c) Each element (except the ones on the last level) points to the proper selected element one level below.

Figure 2.3: Illustrating the fractional cascading technique.

2.2.3 An optimal algorithm

Note that the k objects reported are actually in non-increasing order of score, and that is why the $k \log k$ factor appears in the bound. If we want the results to be sorted, then the previous algorithm is suitable; otherwise, we can do better. We now propose an optimal algorithm which will output the top-k objects in arbitrary order within a query of $O(\log n + k)$. Our approach relies on the following result from [37].

Lemma 2.1. *([37]) Given m sorted arrays of reals, it is possible to find the ck-th smallest/largest real in $O(m)$ time, where $c \geq 1$ is a constant.*

In [37], the operation associated with Lemma 2.1 is called a "CUT". Assume that the points on each layer are given in (say) clockwise order in an array. Then for a given weighting vector w, the corresponding extreme points on the layer partition the points of the layer into two sub-arrays that are sorted by score w.r.t. w. (See, for instance, the points colored red and green in Figure 2.4.) From our earlier discussion, we know that the top-k points w.r.t. w must belong to the first k or fewer layers. Using the approach in Section 2.2.2, we find the extreme points in the first k layers in $O(k + \log n)$ time and use these to create $m = 2k$ sorted arrays of points from these layers. We then apply the CUT operation (Lemma 2.1) to identify the ck-th largest point (by score) in $O(k)$ time. Next we scan each of the $2k$ sorted arrays by non-increasing score and identify a set, S, of points whose score is greater than or equal to the ck-th largest score; this takes $O(k)$ time. Note that since $c \geq 1$, S is a superset of the set of top-k points desired, and the size of S is just $ck = O(k)$. We then run a standard selection algorithm [24] on S to find the point with the k-th largest score in $O(k)$ time. Finally, we use this point to extract from S the desired top-k points in additional $O(k)$ time. Thus, the overall time to answer the top-k query is $O(\log n + k)$, which improves upon the result in Section 2.2.2. (However, note that the points are now no longer reported in non-increasing order of score.)

2.3 Extensions

In this section, we introduce two extensions of the standard preference top-k query in 2D, i.e., preference top-k query with range restriction on data points (Section 2.3.1) and

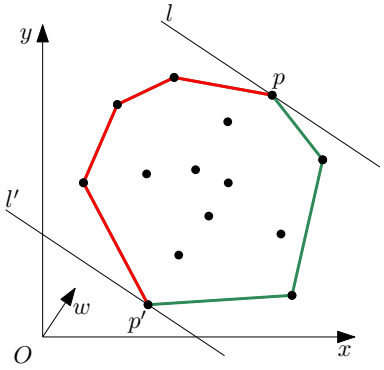


Figure 2.4: The maximal point p and minimal point p' in layer-1 with respect to weighting vector w; point p (resp. p') has the maximum (resp. minimum) score on layer-1.

preference top-k query with a fuzzy weighting vector (Section 2.3.2).

2.3.1 Preference top-k query with range restriction on data points

Usually not all the data points are interesting to a user. Instead, the user may want to "zoom into" a region of interest and want only the top-k points among the data points in this region. We now give a formal definition for this type of top-k query.

Definition 2.1. *Given a set of n data points $P = \{p_1, p_2, \cdots, p_n\}$. A user query consists of a rectangle $R = [x_1, x_2] \times [y_1, y_2]$, and a weighting vector w. Define $P' = \{p \mid p \in P \cap R\}$. The top-$k$ query with range restriction on R reports the k objects in P' with highest score with respect to w. We call this query a range top-k query.*

Range trees [27] are often used to answer various types of range queries. In this section, we show how to answer a range top-k query.

We build a 2D range tree, T, on the set P, and in each internal node we store the convex layers of the points in the node's subtree, along with the corresponding fractional cascading structure. Then our range top-k query can be answered as shown in Algorithm 1. The pseudocode of Algorithm 1 refers to so-called *canonical nodes*. These are a subset of nodes identified in the second level of the range tree, T, when searching with the query range R. The canonical nodes have the following nice property: The subset of P that is contained in R is the disjoint union of the sets of points stored

in the subtrees rooted at the different canonical nodes. Moreover, the number, C, of such canonical nodes is $O(\log^2 n)$. (See [27] for more details.)

Algorithm 1 Range-top-k-query

1: **Input:** Input point set P, 2D range tree T, query range R, weighting vector w.
2: **Output:** Range top-k points.
3: Search in T with R to find the set of canonical nodes. Let C be the number of canonical nodes.
4: For each canonical node, find the extreme point w.r.t. w in the first convex layer stored with the node.
5: Build a max-heap H on the C extreme points found in step 2, using the score w.r.t. w as the key.
6: $Ans \leftarrow \emptyset$
7: **for** $i \leftarrow 1$ **to** k **do**
8: $\quad p \leftarrow H.DeleteMax$
9: $\quad Ans \leftarrow Ans \cup \{p\}$
10: $\quad$ Assume that p is stored at canonical node x.
11: $\quad$ **if** p is one of extreme points of some convex layer stored at x **then**
12: $\quad\quad$ Insert p's two neighbors and the extreme point of the next inner layer into H.
13: $\quad$ **else**
14: $\quad\quad$ Insert the appropriate neighbor of p into H.
15: $\quad$ **end if**
16: **end for**
17: **return** Ans

In Algorithm 1, Lines 3 and 5 take $O(\log^2 n)$ time. Line 4 takes $O(C \cdot \log n) = O(\log^3 n)$ time because a binary search is needed at each canonical node. Since the size of H at any time is $O(C + k)$, the time for all executions of the **for**-loop on Line 7 is $O(k \log(C+k)) = O(k \log(Ck)) = O(k \log\log n + k \log k)$. Therefore the overall running time is $O(\log^3 n + k \log\log n + k \log k)$, and the reported objects are in non-increasing order of score.

The bottleneck of this algorithm is Line 4, which costs $O(\log^3 n)$ time. To reduce the running time, we build a fractional cascading structure on the second level of the range tree. (The idea is similar to the one used in Section 2.2.2 but there are several key differences as discussed below.) Specifically, for each node c in the first level of T, we wish to jump to its corresponding subtree rooted at c' in the second level of T via a stored pointer (see Figure 2.5 for an example). Towards this end, we extract the arrays

corresponding to the first layer only in each node in the second level of T, and build a fractional cascading structure for these extracted arrays bottom-up; note that the structure is slightly different compared to the one in Section 2.2.2, i.e, we have pointers from layer-i to layer-$(i+1)$ only in the previous case, but now, for each entry of the array, we have one pointer to the entry in the left subtree, and one pointer to the entry in the right subtree. For instance, in Figure 2.5, the array pointed to by each node in the subtree rooted at c' corresponds to the first layer information of the fractional cascading structure stored inside that node. Every entry (except the one in the last level) in these arrays has two pointers, one each for the arrays in the left and right subtrees. Based on this newly built structure, when we perform a search to find the canonical nodes in the subtree rooted at c' (e.g. the red nodes in Figure 2.5), we can also retrieve the extreme points of their first layers by doing one binary search at the root of c', in $O(\log n)$ time, to find the extreme point at that node and then following the stored pointers to the appropriate left or right child in $O(1)$ time to retrieve the other extreme points. For the example in Figure 2.5, we need to do just one binary search (at c') and follow pointers at the three red nodes, whereas previously we did three binary searches (at the three red nodes).

Recall the property of a 2D range tree. Given any range, $O(\log n)$ nodes will be involved in the first level so that $O(\log n)$ binary searches are done for a total of $O(\log^2 n)$ time. Moreover, we can charge the cost of following by pointers and finding the extreme points (in the cascading structure) to the C canonical nodes, which clearly takes $O(C) = O(\log^2 n)$ time. Therefore, the run time of Step 2 improves to $O(\log^2 n)$, and the overall run time for the algorithm improves to $O(\log^2 n + k \log\log n + k \log k)$. The overall space usage is $O(n \log n)$.

2.3.2 Preference top-k query with a fuzzy weighting vector

In the standard version of the top-k query considered so far, it is assumed that the user can specify the unit weighting vector accurately. However this may not always be the case because the preference of a user is may not be known exactly. Therefore it is more reasonable to let a user specify a fuzzy unit weighting vector. In 2D, a unit vector lies on the unit circle, and can be represented by the angle it makes with the positive x-axis. Hence a range of weighting vectors can be defined as an interval of angles, say $[\theta_1, \theta_2]$.

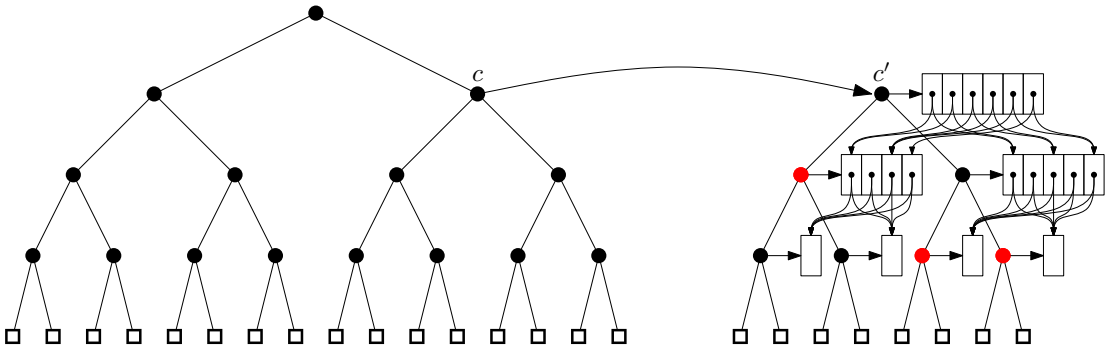


Figure 2.5: Example to show the fractional cascading structure in the 2D range tree.

We now give the formal definition of the top-k query in this setting.

Definition 2.2. *Given a set of n data points $P = \{p_1, p_2, \cdots, p_n\}$. A user query consists of a range of angles $R = [\theta_1, \theta_2]$ illustrating the range of the desired unit weighting vector w. Redefine the score of any point p to be $\max(\sum p \cdot w)$, taken over all $w \in R$. A top-k query on P with fuzzy weighting vector $w \in R$ will report the k objects in P with highest redefined score.*

Our strategy is to identify efficiently a superset of the desired top-k points, of size at most $3k$, and then identify the top-k points from this. Let w_1 and w_2 be the unit weighting vectors corresponding to θ_1 and θ_2, respectively. As shown in Figure 2.6, w_1 and w_2 partition P into three sets P_1, P_2, and P_3. We make the following observations.

1. For any $p \in P_1$ and $w \in R$, the redefined score is maximized for $w = w_1$ (since then the angle between w and $\overrightarrow{Op}$ is minimized). Suppose that p is reported when a top-k query is performed on P with a fuzzy weighting vector $w \in R$. Thus, $p \cdot w_1$ is among the k highest scores found in P. Since $P_1 \subseteq P$, a standard top-k query on P_1 with weighting vector w_1 (see Section 2.2.2) will also report p. Hence, all points of P_1 that are part of the answer to the fuzzy top-k query on P will be included in the output of the standard top-k query on P_1 (along with $O(k)$ spurious points, i.e., points that are not part of the answer to the fuzzy top-k query on P).

   A similar observation also applies to P_2 w.r.t. weighting vector w_2.

2. For any $p \in P_3$, the redefined score is maximized when the weighting vector $w = \overrightarrow{Op}$; the score is the L_2-distance of p from O, i.e., $|Op|$. Thus, reasoning as in Observation 1, if p is reported when a top-k query is performed on P with a fuzzy weighting vector $w \in R$, then p is among the set of k points of P_3 farthest from O. (Again, this set can have $O(k)$ spurious points.)

For a query R, we can find the k points of P_3 farthest from O as follows: In preprocessing, we sort the points of P by non-decreasing angle from the positive x-axis and map each point p to a weighted 1D point p', where the angle of p becomes the coordinate of p', and the distance from O to p is the weight. We then build a priority search tree T on these points [27, 49].

Given R, we traverse T and identify the set of C canonical nodes, where $C = O(\log n)$. The desired set of k farthest points from O is contained in the disjoint union of all the heap-ordered trees rooted at these canonical nodes. We can find these by initializing a max-heap H with the root node of each canonical node. Then we repeatedly delete and report the maximum from H and insert the children of each maximum (in terms of the priority search tree T) into H. We do this k times (or until H is empty). Since $|H| \leq C+k$ this takes $O(k \log(C+k)) = O(k \log\log n + k \log k)$ time. Including the time to identify the set of canonical nodes, the total query time is $O(\log n + k \log\log n + k \log k)$. The space is $O(n)$.

Alternatively, as noted in [3], one can solve the problem on P_3 in $O(\log n + k)$ time and $O(n)$ space by using a priority search tree combined with Frederickson's $O(k)$-time algorithm for finding the k smallest/largest elements in a binary heap-ordered tree [36].

At this point we have a set P' of at most $3k$ points, resulting from the output of the queries on P_1, P_2 and P_3, which contains the output set for the fuzzy top-k query on P. Among the points of P', we find the one with the k-th largest score, using a standard selection algorithm [24] and then scan P' with this point to identify the top-k points for the fuzzy top-k query on P. This step takes $O(k)$ time. Hence, the total query time on P_1–P_3 is $O(\log n + k \log k + k \log\log n)$ and the space is $O(n)$. Alternatively, by incorporating the heap-selection algorithm from [36] in the query of P_3 and the CUT operation from [37] in the standard top-k query of P_1 and P_2, the query time can be reduced to $O(\log n + k)$, while still using $O(n)$ space.

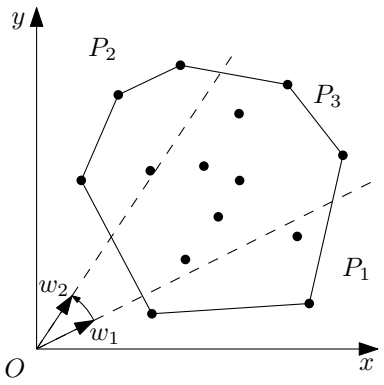


Figure 2.6: Illustrating the approach for answering a preference top-k query with a fuzzy weighting vector lying anywhere between w_1 and w_2. The sets P_1-P_3 are defined by weighting vectors w_1 and w_2 that make angles θ_1 and θ_2, respectively, with the positive x-axis.

2.4 Algorithm in 3D

In this section, we extend to 3D our exact algorithm in Section 2.2.1, which was based on convex layers.

The size of the convex hull in 3D is linear in the number, n, of input points. Hence the convex layers occupy only $O(n)$ space. The sketch of the algorithm in Section 2.2 still holds, but we need to address two issues that can impact the query time: 1) how to find the extreme points efficiently? and 2) what is the degree of each vertex (i.e., its number of neighbors) on the hull?

1. To find an extreme point on a convex layer, we can use planar point location [27], as follows.

   Given any 3D convex hull, we first create the unit-normal of each facet. Then we translate all the normals so that their starting points coincide with the origin; thus their ending points will be all on the unit sphere. For each vertex p_i on the hull, we list all its associated facets, $f_{i_1}, f_{i_2}, \cdots, f_{i_t}$, in clockwise/counter-clockwise order. Next we connect f_{i_1} and f_{i_2}, f_{i_2} and f_{i_3}, $\cdots$, $f_{i_{t-1}}$ and f_{i_t}, and f_{i_t} and f_{i_1} respectively via arcs on their corresponding great circles. These arcs will form a closed cycle on the surface of the sphere, and we associate the interior of the cycle with vertex p_i. For convenience, we name that interior c_i. Figure 2.7a

shows an example where we assume the convex hull is a tetrahedron with vertices p_1, p_2, p_3, p_4. f_i and n_i indicate the i^{th} facet and its unit normal respectively. Figure 2.7b shows the positions of these normals after translation. Moreover, $c_1 = n_1 n_2 n_3 n_1$ is the cell associated with p_1, cell $c_2 = n_1 n_2 n_4 n_1$ is associated to p_2, etc.

Now given any weighting vector w, vertex p_i is the extreme point with respect to w if and only if the ending point of w is inside the interior of the cell c_i. Note that every arc on the sphere is part of the great circle, and thus it will become a line segment if gnomonic projection [25] is applied. Therefore, after gnomonic projection, the result will be a planar graph in which each cell c_i will uniquely correspond to a facet, $\hat{c}_i$, in that graph, and the weighting vector w will become a point $\hat{w}$. It is clearly that w lies in c_i if and only if $\hat{w}$ is inside facet $\hat{c}_i$. Hence finding an extreme point has been reduced to the planar point location problem. By using a persistent search tree, planar point location can be done in $O(\log n)$ time and $O(n)$ space [52]. Therefore, finding an extreme point in 3D can be done in $O(\log n)$ time without increasing the asymptotic space complexity.

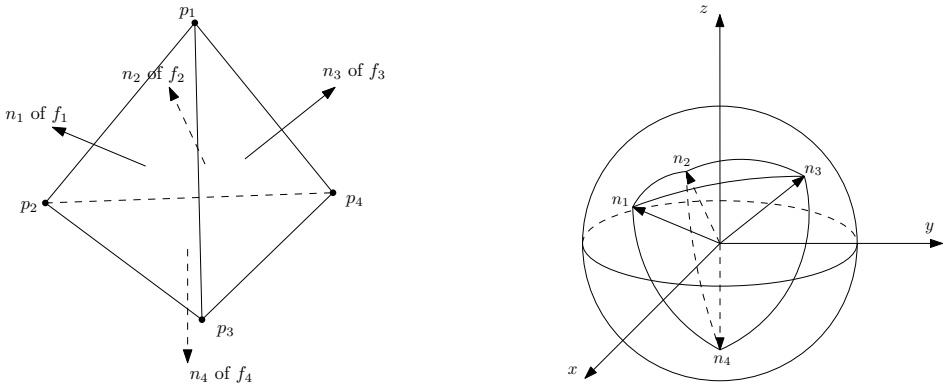
Note that, unlike the 2D case, each 3D weighting vector is uniquely determined by two parameters, and the fractional cascading technique discussed in Section 2.2.2 is no longer supported here.

2. Unfortunately, the degree of each vertex is no longer a constant in 3D; and it can be any number from 3 to $n-1$, so that after some point p is deleted from the heap, we have to check all its neighbors, which would be costly in the worst case.

   We can soften the worst case a little by dividing the $m \leq n$ points on the hull into $\sqrt{m}$ groups of roughly $\sqrt{m}$ points each. We then build the convex hull for each group. It is clear that the total space remains $O(m)$, but the maximum degree in each sub-hull is less than $\sqrt{m}$. To find the extreme point from all the original m points w.r.t. some preference vector w, we can find the extreme point of each sub-hulls using point location and maintain these $O(\sqrt{m})$ points (by score) in a max-heap, which takes $O(\sqrt{m}\log\sqrt{m}) = O(\sqrt{m}\log m)$ time. Retrieving the next largest point involves checking its neighbors in its sub-hull and inserting them into the heap and finally deleting the maximum point. Note that a point will be

inserted and deleted at most once, hence all the heap operations invoked in each group take $O(\sqrt{m}\log m)$ time.

Assume the 3D onion structure of all the n points consists of t layers, and the i-th layer contains n_i points, i.e., $n_1 + n_2 + \cdots + n_t = n$. We apply the strategy above, i.e., for the hull in layer i, we partition the n_i points into $\sqrt{n_i}$ groups and maintain a max-heap on them. We also build one extra max-heap to keep track of the largest point in the first k layers as our previous algorithm does. To sum up, at most k extreme points will be accessed, which takes at most $O(\sqrt{n_1}\log n_1 + \cdots + \sqrt{n_k}\log n_k) = O(k\sqrt{n}\log n)$. Reporting the top-$k$ objects in sorted order also takes $O(k\sqrt{n}\log n)$. Therefore, the worst case running time is bounded by $O(k\sqrt{n}\log n)$.



(a) An example of 3D convex hull

(b) Unit normals partition the unit sphere into several cells.

Figure 2.7: Illustrating how to reduce finding extreme points in 3D to planar point location.

Remark. The basic idea of our algorithm extends to higher dimension as well, but it is not very attractive due to two reasons: 1) the degree of a vertex of the convex hull can be as large as n, and, more importantly, 2) the size of the convex hull grows dramatically to $O(n^{\lfloor \frac{d}{2} \rfloor})$ in d-dimensions. In other words, storing the entire convex hull structure is too costly when the dimension is high. Therefore, in next chapter develop an efficient approximation algorithm for the preference top-k query in higher dimensions that avoids storing the entire hull.

Chapter 3

Approximate preference top-k query

In this chapter, we present our sampling-based approximation scheme for the preference top-k problem in higher dimensions.

3.1 Problem formulation

Recall the preference top-k problem we defined in Section 2.1. We are given a set of n d-dim data points $D = \{a_1, a_2, \ldots, a_n\} \subset (\mathbb{R}^+)^d$ and an integer $k \leq n$. For any given d-dim query vector (i.e., preference) $q = (q_1, q_2, \ldots, q_d)$ satisfying each $q_i \geq 0$ and $q \neq \mathbf{0}$, the goal of the preference top-k problem is to find k points from D, which have the largest inner products with q, and report them in order. In other words, the reported points $a_{\pi_1}, a_{\pi_2}, \ldots, a_{\pi_k}$ have to satisfy

$$q \cdot a_{\pi_1} \geq q \cdot a_{\pi_2} \geq \cdots \geq q \cdot a_{\pi_k} \geq q \cdot a_i,$$

for any $a_i \in D - \{a_{\pi_1}, a_{\pi_2}, \ldots, a_{\pi_k}\}$.

An effective approximation approach for dealing with preference top-k queries is sampling. The high-level idea of a sampling-based approximation is to sample a subset of the original dataset D (called sampling set or sampling subset), which can well-represent the top-k features of the entire set but has a much smaller size. When a query preference vector q is given, the algorithm focuses only on the points in the sampling

set, i.e., it identifies the top-k points of the sampling set under q, and uses the data points so identified as an approximation to the true top-k result. In this way, each query can be answered much more efficiently. Clearly, how to get a small sampling set with high quality is the crucial part of sampling-based approximation. Ideal sampling sets should be representative and small-sized so that both the quality of the top-k answer and query efficiency can be guaranteed.

3.2 Our sampling algorithm

In this section, we first introduce an important conclusion which enables us to reduce top-k sampling to top-1 case. Then, we introduce the concept of critical detection vectors, which play a key role in our sampling algorithm. Finally, we present the overall framework of our algorithm.

3.2.1 Reducing top-k to top-1

We denote by $\phi_q(D)$ the maximum of the inner products of the query vector q and the points in the dataset D, i.e.,

$$\phi_q(D) = \max_{a_i \in D} q \cdot a_i,$$

and the corresponding a_i is called the *top-1 point* of D under q. Let $S \subseteq D$ be a (top-1) sampling set, we now define the *overall top-1 error* of S, denoted by E_S, as

$$E_S = \sup_q \{Err(\phi_q(D), \phi_q(S))\}.$$

where

$$Err(\phi_1, \phi_2) = \frac{\phi_1 - \phi_2}{\phi_1}.$$

Also, we call $Err(\phi_q(D), \phi_q(S))$ the *top-1 error of S under q*.

Assume we have a sampling algorithm A only for top-1 (i.e., the case of $k = 1$), which can guarantee the overall top-1 error of the sampling set to be less than α for any given dataset D. We construct a corresponding top-k sampling algorithm $\bar{A}$ as indicated in Algorithm 2.

Theorem 3.1 shows that the quality of the top-k sampling set S^* obtained by $\bar{A}$ is well-guaranteed.

Algorithm 2 $\bar{A}$

procedure $\bar{A}(D, k)$ ▷ Return the top-k sampling set S^* of the dataset D.
    Initial $i = 0$ and $S^* = \varnothing$
    while $i < k$ **do**
        $i \leftarrow i + 1$
        $S \leftarrow A(D)$
        $S^* \leftarrow S^* \cup S$
        $D \leftarrow D \backslash S$
    end while
    return S^*
end procedure

Theorem 3.1. *For any query vector q, let $a_{\pi_1}, a_{\pi_2}, \ldots, a_{\pi_k}$ be the true sorted top-k data points in D with respect to q and let $b_{\pi'_1}, b_{\pi'_2}, \ldots, b_{\pi'_k}$ be the sorted top-k points in the sampling set S^* obtained by $\bar{A}$. Define*

$$L = \{i : Err(q \cdot a_{\pi_i}, q \cdot b_{\pi'_i}) > 0\}.$$

Assume that $|L| = k'$ and $L = \{l_1, l_2, \ldots, l_{k'}\}$ where $l_1 < l_2 < \cdots < l_{k'}$. If the algorithm A can guarantee the overall top-1 error of the sampling set to be less than α, then for any $i \in \{1, 2, \ldots, k'\}$, we have

$$Err(q \cdot a_{\pi_{l_1}}, q \cdot b_{\pi'_{l_i}}) \leq \alpha.$$

(See Section 3.5.1 for a proof.)

Since $q \cdot a_{\pi_{l_1}} \geq q \cdot a_{\pi_{l_i}}$, the conclusion of Theorem 3.1 implies that $Err(q \cdot a_{\pi_{l_i}}, q \cdot b_{\pi'_{l_i}}) \leq \alpha$ and is even better than this. With this conclusion, the top-k sampling task can be naturally reduced to k iterations of top-1 sampling. In other words, in order to do sampling for top-k, it suffices to propose a good top-1 sampling algorithm A and then develop it into $\bar{A}$, as specified in Algorithm 2.

3.2.2 Critical detection vectors

To do sampling for the top-1 case, the basic strategy of our algorithm is to use different query vectors on the original dataset D to get different top-1 data points and collect them as the sampling set. We call the used query vectors *detection vectors*. Since the

number of possible query vectors is infinite, we can only use a small subset of these vectors for detection. Thus, the selection of the detection vectors largely determines the quality of the obtained sampling set.

If the detection vectors are blindly selected, i.e., the selection made is independent of the dataset D, the quality of the sampling set in general cannot be well-guaranteed. As we see below, even in the 2-D case, the overall top-1 error of the sampling set can almost reach 0.5 in the worst case, no matter how many detection vectors we select and what they are.

Consider the selection of detection vectors in the 2-D case. Assume we have selected the detection vectors independent of the dataset. Since the number of detection vectors we select is finite, we can always find a number $\theta \in (0, \pi/2)$ so that there is no detection vector selected whose angle from the y-axis belongs to $(0, \theta)$. Then we simply construct a dataset D which only contains 3 data points:

$$D = \{p_1, p_2, p_3\},$$

$$p_1 = (m, 1),\ p_2 = (M, 1 - m),\ p_3 = (M + \cot\theta, m),$$

where $M \to +\infty$ and $m \to 0^+$. Let S be the sampling set obtained by using these detection vectors. Because the angles of the detection vectors (from the y-axis) are not in $(0, \theta)$, the data point p_2 can never be detected, i.e., never be included in S. Then we consider the query vector $q = (1, M + \cot\theta)$. Since $p_2 \notin S$, even if p_1 and p_3 are both in S, we have

$$Err(\phi_q(D), \phi_q(S)) = \frac{M - 2m(M + \cot\theta)}{M + (1 - m)(M + \cot\theta)} \to 0.5,$$

which means the overall top-1 error of S can almost reach 0.5. This unsatisfactory result motivates us to select the detection vectors adaptively, based on the dataset.

Our algorithm performs adaptive selection of detection vectors by alternately selecting new detection vectors based on the current sampling points and then expanding the sampling set. Suppose now that we have a sampling set S and the dataset D is unknown. We investigate the detection vectors which are most helpful for improving the quality of S. Intuitively, good detection vectors should reveal large top-1 errors of S when used as query vectors, thereby yielding meaningful top-1 points as new sampling points and helping remedy significant defects of S. One important observation we have

is that, although the number of the possible query vectors which may lead to top-1 error of S is infinite, there exists a small set of vectors under which the top-1 error of S always "dominates" those under other vectors. Moreover, this set of vectors is independent of the dataset D. We formalize this intuition via the following definition and via Theorem 3.2 and Theorem 3.3.

Definition 3.1. *Let S be a sampling set. We say a vector v is critical to S if and only if*

1) *all the components of v are nonnegative;*

2) *for some c, there exists $s_{\pi_1}, \ldots, s_{\pi_c} \in S$ and $z_1, \ldots, z_{d-c} \in \{1, \ldots, d\}$ such that*

- *$s_{\pi_1}, \ldots, s_{\pi_c}$ are the top-1 points of S under v;*
- *all the z_i-th components of v are 0;*
- *$s_{\pi_1}, \ldots, s_{\pi_c}$ and $z_1, \ldots, z_{d-c}$ satisfy*

$$rank \begin{pmatrix} & s_{\pi_1} & \\ & \vdots & \\ & s_{\pi_c} & \\ & e_{z_1} & \\ & \vdots & \\ & e_{z_{d-c}} & \end{pmatrix}_{d \times d} = d. \tag{3.1}$$

Also, define the critical vector set C of S as the set of all unit vectors critical to S.

In the previous example, if $S = \{p_1, p_3\}$, there are only three vectors critical to S: $(1, 0)$, $(0, 1)$ and $(1 - m, M + \cot\theta - m)$. So the critical vector set of S is $V = \{(1, 0), (0, 1), (1 - m, M + \cot\theta - m)\}$. (Strictly, the third vector should be normalized but we have omitted this to avoid clutter.)

Theorem 3.2. *Given $s_{\pi_1}, \ldots, s_{\pi_c} \in S$ and $z_1, \ldots, z_{d-c} \in \{1, \ldots, d\}$ satisfying Equation 3.1, there exists at most one unit vector v such that*

- *all the components of v are nonnegative and all the z_i-th components are 0;*
- *$s_{\pi_1}, \ldots, s_{\pi_c}$ are the top-1 points of S under v.* (See Section 3.5.2 for a proof.)

Clearly, Theorem 3.2 implies the finiteness of the number of the vectors critical to S. More precisely, if the dimension d is constant, the size of the critical vector set of S

is always bounded by $O(|\mathcal{CH}(S)|)$, where $|\mathcal{CH}(S)|$ is the complexity of the convex hull of S (note that $s_{\pi_1}, \ldots, s_{\pi_c}$ are the top-1 points under v only if they form a $(c-1)$-D edge of the convex hull of S).

Theorem 3.3. *Let S be a sampling set and V be its critical vector set. For any $D \supseteq S$ and any query vector q, we have*

$$Err(\phi_q(D), \phi_q(S)) \leq \max_{v \in V}\{Err(\phi_v(D), \phi_v(S))\}.$$

As a result,

$$E_S = \sup_q\{Err(\phi_q(D), \phi_q(S))\} = \max_{v \in V}\{Err(\phi_v(D), \phi_v(S))\}.$$

(See Section 3.5.3 for a proof.)

Roughly speaking, Theorem 3.3 tells us that S has the worst performance (i.e., maximum top-1 error) under the vectors critical to it. In this sense, when we want to further expand S by detecting new top-1 data points, these critical vectors will be the best choices as detection vectors. By adding the top-1 points detected by them, the robustness of S can be most improved.

3.2.3 Overall framework

Based on the observation in the previous subsection, the basic idea of our top-1 sampling algorithm emerges: we begin from a very small S, find the vectors critical to S and use them as detection vectors to expand S to a larger set, then find the new critical detection vectors of the current S and further expand it to an even larger set, until the current S satisfies some termination conditions. Since the d basis vectors

$$e_1 = (1, 0, 0, \ldots, 0),$$

$$e_2 = (0, 1, 0, \ldots, 0),$$

$$\ldots,$$

$$e_d = (0, 0, 0, \ldots, 1)$$

are critical to any non-empty S, we also regard them as critical detection vectors of the empty set, which allows us to initialize our algorithm with $S = \varnothing$.

In order to compute the critical detection vectors for the current sampling set, what we do is to maintain a convex hull for the expanding S. Then for each edge $\overline{s_{\pi_1} \dots s_{\pi_c}}$ of the convex hull, if at least one of its vertices is new (i.e., added to S in the last round), we consider each tuple $(z_1, \dots, z_{d-c})$ satisfying Equation 3.1. We compute the unique vector v^* such that

$$v^* \cdot s_{\pi_1} = \cdots = v^* \cdot s_{\pi_c}$$

and all the z_i-th components of v^* are 0. We say v^* is a *candidate* determined by $s_{\pi_1}, \dots, s_{\pi_c}$ and $z_1, \dots, z_{d-c}$. If all the components of v^* are nonnegative and $s_{\pi_1}, \dots, s_{\pi_c}$ are the top-1 points of S under v^*, then v^* is indeed a vector critical to S and not used in the previous rounds as detection vectors. So we add v^* to the set of the critical detection vectors we want to use in the current round. To set the termination condition for expanding, we need a threshold α to indicate the maximum allowable error of S, which is an input parameter given by the user. While expanding, only if the top-1 error of S under a critical detection vector reaches or exceeds α, we add the corresponding new top-1 data point to the sampling set. Once the top-1 error of S under each critical detection vector is less than α (equivalent to $E_S < \alpha$), we terminate our expanding and output the current S as our result.

The working of our top-1 sampling algorithm in 2D can be seen in the next section. By combining our top-1 sampling algorithm with Algorithm 2, we obtain the overall framework of our top-k sampling algorithm, which is shown as Algorithm 3.

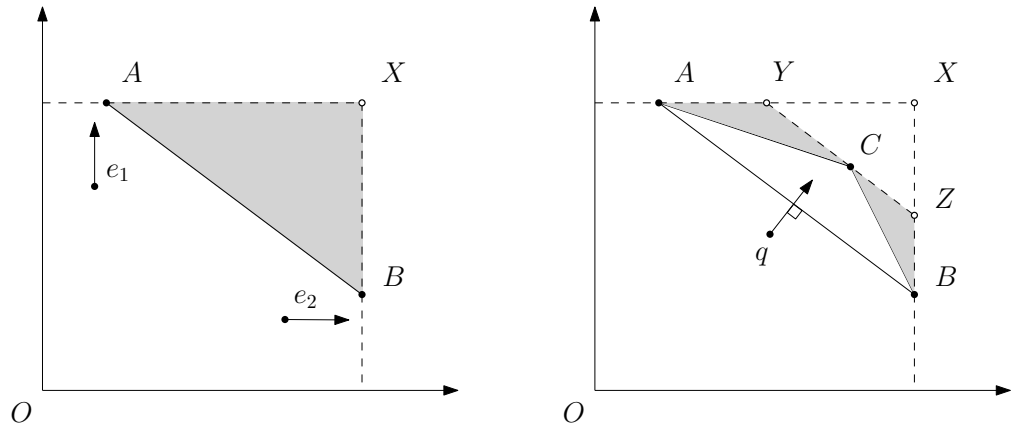
3.3 Theoretical analysis in 2D

Since our algorithm for top-k sampling is realized by k iterations of top-1 sampling, it suffices to analyze the process of top-1 sampling. We begin from the empty set. In the case of $d = 2$, this process becomes relatively simple. In a general case, we use the two critical vectors $e_1 = (0, 1)$ and $e_2 = (1, 0)$ for detection and obtain the topmost data point A and the rightmost point B (Figure 3.1a) in our first round of expanding. Then, for the expanded S, we have a new critical detection vector q which is perpendicular to the line $\overline{AB}$. By this vector, we detect a new data point C in our second round (Figure 3.1b). Accordingly, we further obtain two new critical detection vectors, one is perpendicular to $\overline{AC}$ and the other is perpendicular to $\overline{CB}$. By these

Algorithm 3 Our Top-k Sampling Algorithm

procedure TOP-k-SAMPLING(D, k, α) ▷ Return the top-k sampling set S^* of dataset D with maximum allowable error α.
  $S^* \leftarrow \varnothing$
  $i \leftarrow 0$
  while $i < k$ **do**
    $i \leftarrow i + 1$
    $S_i = \varnothing$
    $C = \{e_1, e_2, \ldots, e_d\}$
    while $T \neq \varnothing$ **do**
      $T = \varnothing$;
      for every $v \in C$ **do**
        if $Err(\phi_v(D), \phi_v(S_i)) > \alpha$ **then**
          $T \leftarrow T \cup \{a \mid a \in D, v \cdot a = \phi_v(D)\}$
        end if
      end for
      $S_i \leftarrow S_i \cup T$
      Update $\mathcal{CH}(S_i)$
      $C \leftarrow$FIND-NEW-CRITICAL-VECTORS(S_i,T)
    end while
    $S^* \leftarrow S^* \cup S_i$
    $D \leftarrow D \backslash S_i$
  end while
  return S^*;
end procedure
procedure FIND-NEW-CRITICAL-VECTORS(U,V)
  $C \leftarrow \varnothing$
  for every edge $\overline{s_{\pi_1} \ldots s_{\pi_c}}$ of $\mathcal{CH}(U)$ **do**
    if at least one of $s_{\pi_1}, \ldots, s_{\pi_c}$ is contained in V **then**
      for any $z_1, \ldots, z_{d-c} \in \{1, \ldots, d\}$ such that Equation 3.1 holds **do**
        Compute the candidate v^* determined by $s_{\pi_1}, \ldots, s_{\pi_c}$ and $z_1, \ldots, z_{d-c}$
        if all the components of v^* are nonnegative and $s_{\pi_1}, \ldots, s_{\pi_c}$ are the top-1 points of U under v^* **then**
          $C \leftarrow C \cup \{v^*\}$
        end if
      end for
    end if
  end for
  return C
end procedure

two detection vectors, we may add even more data points to keep expanding S. This procedure repeats until the top-1 error of S under every new critical detection vector is less than α.



(a) The first round of expanding

(b) Finding new detection vector q and further expanding S

Figure 3.1: Sampling algorithm in 2D

In this process, we have some observations:

(1) At any time, if we sort the data points in S by their x-coordinates, we then obtain a sequence $s_1, s_2, \ldots, s_m$ ($m = |S|$) in which $s_1.x < s_2.x < \cdots < s_m.x$ and $s_1.y > s_2.y > \cdots > s_m.y$.

(2) If S currently contains m points, then there are totally $m + 1$ vectors critical to S. Two of them are $(0, 1)$ and $(1, 0)$. The remaining $m - 1$ ones are the vectors perpendicular to $\overline{s_1 s_2}$, $\overline{s_2 s_3}$, $\ldots$, $\overline{s_{m-1} s_m}$. Some of these critical vectors may be used as detection vectors in previous rounds while the others are new.

(3) Since each $s_i \in S$ is the top-1 point under a previous detection vector q_i, all of the data points in D should be on the same side of l_i as the origin point O, where l_i is the line passing through s_i and perpendicular to q_i. Furthermore, all of the meaningful points in $D - S$ (i.e., the undetected points which can improve the quality of S if added to S) are clearly in the $m - 1$ triangles $\triangle s_1 s_2 t_1$, $\triangle s_2 s_3 t_2$, $\ldots$, $\triangle s_{m-1} s_m t_{m-1}$, where t_i is the intersection point of l_i and l_{i+1}. We call such triangles *blind triangles* and their union the *blind area*. For instance, the blind area in Figure 3.1a is the gray triangle $\triangle ABX$ and in Figure 3.1b is the two gray triangles $\triangle ACY$ and $\triangle CBZ$.

(4) The data points in one triangle of the blind area, $\triangle s_i s_{i+1} t_i$, are only meaningful for the critical vector perpendicular to $\overline{s_i s_{i+1}}$, among all of the $m+1$ critical vectors. In other words, under the other m critical vectors, the points in $\triangle s_i s_{i+1} t_i$ never lead to the top-1 error of S. Furthermore, under the vector perpendicular to $\overline{s_i s_{i+1}}$, the point t_i is the one that leads to the largest top-1 error of S, among all possible points in $\triangle s_i s_{i+1} t_i$. We call this maximum the *causing error* of the blind triangle $\triangle s_i s_{i+1} t_i$.

(5) Once we use the critical vector perpendicular to $\overline{s_i s_{i+1}}$ to detect a new point s' and add it to S, we get two new critical vectors which are perpendicular to $\overline{s_i s'}$ and $\overline{s' s_{i+1}}$ for the next round of expanding. And in the next round, the blind triangle $\triangle s_i s_{i+1} t_i$ will be replaced by two new ones, $\triangle s_i s' u$ and $\triangle s' s_{i+1} v$ (Figure 3.2). We say these two new triangles are generated by the original one $\triangle s_i s_{i+1} t_i$. Note that if we fail to detect a new point (e.g., there is no point in the blind triangle) or the point detected leads a top-1 error less than our threshold α (so that we do not add it to S), then the corresponding blind triangle will always exist but be meaningless in the subsequent rounds. We say it's *invalid* in the subsequent rounds. Obviously, in a particular round, the valid blind triangles are just the ones newly generated in the last round while the others are all invalid.

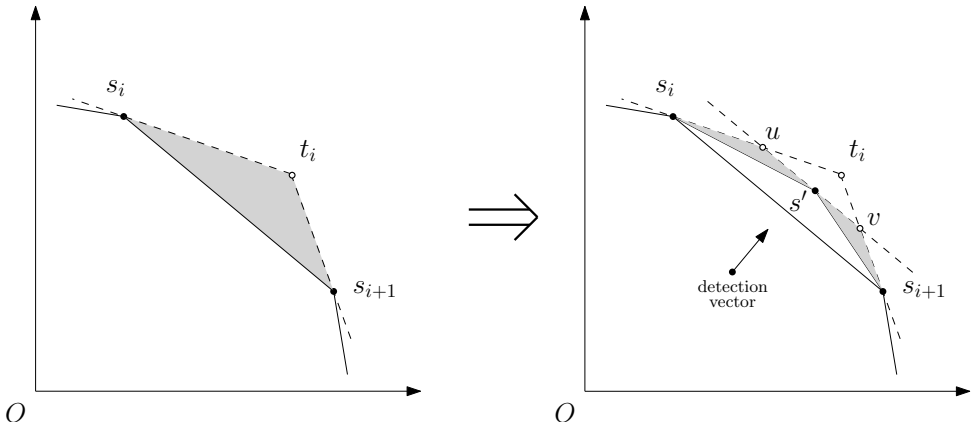


Figure 3.2: Generating two new blind triangles from an old one

Based on these observations, we have a conclusion about the relationship between the causing error of a blind triangle and that of the two blind triangles it generates in the next round. Consider the situation in Figure 3.2. We have the blind triangle $\triangle s_i s_{i+1} t_i$ in some round and $\triangle s_i s' u$, $\triangle s' s_{i+1} v$ are the two new blind triangles it generates in the

next round. Then our conclusion is shown in Theorem 3.4.

Theorem 3.4. *Suppose E, E_1, E_2 are the causing errors of $\triangle s_i s_{i+1} t_i$, $\triangle s_i s' u$, $\triangle s' s_{i+1} v$, respectively. Then we have*

$$\frac{E_1}{1-E_1} + \frac{E_2}{1-E_2} \leq \frac{E}{1-E}, \text{ and } \max\{E_1, E_2\} \leq \frac{E}{E + 2\sqrt{E(1-E)} + 1}.$$

(See Section 3.5.4 for a proof.)

From Theorem 3.4, we readily have the following corollary.

Corollary 3.1. *The number of the rounds for expanding S is bounded by $O(\alpha^{-0.5})$. More precisely,*

$$r \leq \left\lceil \sqrt{\frac{1}{\alpha} - 1} \right\rceil,$$

where r is the number of the rounds. (See Section 3.5.5 for a proof.)

Based on the conclusions of Theorem 3.4 and Corollary 3.1, we can upper-bound the size of the final sampling set we obtain as well as the number of detections during our algorithm, as stated in Theorem 3.5.

Theorem 3.5. *In Algorithm 3, the final sampling set obtained contains at most $O(k\alpha^{-1.5})$ data points and the total number of detections is also bounded by $O(k\alpha^{-1.5})$.* (See Section 3.5.6 for a proof.)

In higher dimensions ($d > 2$), it is difficult to analyze theoretically the bounds of our sampling algorithm. Thus, we will use the experiments to demonstrate its effectiveness in Section 3.4. According to our experimental results, the algorithm can still work very well in higher dimensions. Furthermore, we will also see that, even in 2D, the theoretical bounds we get in Theorem 3.5 is not that tight, i.e., it overestimates the size of the sampling set as well as the total number of detections.

3.4 Experimental results in 2D and higher dimensions

In this section, we present the experimental results of our sampling algorithm in 2D, 3D, and 4D. Although we have shown theoretical analysis for 2D in Section 3.3, the

bounds we get are in fact not very tight because in each step of our proofs we always consider the worst cases (see the proofs of Theorem 3.4 and 3.5 in Section 3.5). This is verified in the later 2D experiments.

Since our top-k sampling algorithm is just a repeat of k iterations of top-1 sampling by using the framework of Algorithm 2, for simplicity, we do experiments only for the case of $k = 1$, i.e., top-1 sampling. The datasets used for experiments have 5 different sizes: n = 10K, 20K, 50K, 100K, 200K. For each of 2D, 3D, 4D and each size, we generate 10 different datasets and record the average sizes of the sampling sets generated by our algorithm on them. All of the datasets we use are randomly generated, where the data points distribute uniformly inside an open unit-ball (i.e., the space $Z = \{x \in (\mathbb{R}^+)^d : \|x\|_2 < 1\}$). The maximum allowable error parameters we use are α = 0.1, 0.05, 0.02, 0.01, 0.005, 0.001.

Table 3.4 shows our experimental results (the average sizes of the sampling sets) in 2D for different datasets and α-parameters. As we see, although the original datasets are large-sized, the sizes of the sampling sets obtained by our algorithm are in general very small. The sizes naturally increase when α becomes smaller. But they are not sensitive to the sizes of the original datasets, n. This coincides our analysis in Section 3.3. However, if we compare the results with $\alpha^{-1.5}$, we can find that the size of the sampling set grows much slower than $O(\alpha^{-1.5})$ when α decreases. That means our algorithm performs much better than our theoretical expectation in 2D. In other words, the bound $O(k\alpha^{-1.5})$ obtained in Section 3.3 (Theorem 3.5) somehow overestimates the sizes of the sampling sets. According to the table, the growth of the size with respect to α is approximately $O(\alpha^{-0.5})$ in 2D. Consequently, if we do k iterations of such top-1 sampling to obtain the top-k sampling set, its size can be approximately estimated as $O(k\alpha^{-0.5})$.

2D	n =10K	n =20K	n =50K	n =100K	n =200K
$\alpha = 0.1$	3.0	3.0	3.0	3.0	3.0
$\alpha = 0.05$	5.0	5.0	5.0	5.0	5.0
$\alpha = 0.01$	9.0	9.0	9.0	9.0	9.0
$\alpha = 0.005$	11.6	11.0	11.5	11.8	11.7
$\alpha = 0.001$	21.1	22.9	25.9	26.8	27.7

Table 3.1: Experimental results in 2D: the average sizes of the sampling sets

Table 3.4 shows our experimental results in 3D. As we see, the size of the sampling set in 3D is in general much larger than the counterpart in 2D for the same n and α. But compared with the sizes of the original datasets, they are still significantly small. Most times, the change of n does not have great impact on the sizes of the sampling sets. In the last two rows ($\alpha = 0.005$ and $\alpha = 0.001$), the results seem to be sensitive to the growth of n. One possible explanation for this phenomenon is that the datasets are not large enough for such high accuracy requirement so that the sampling set almost contains all possible "top-1 points" in that dataset, the number of which is influenced by n. According to the statistics, the growth of the size of the sampling set with respect to α is still slower than $O(\alpha^{-1})$.

3D	n =10K	n =20K	n =50K	n =100K	n =200K
$\alpha = 0.1$	8.8	9.3	9.6	9.8	9.9
$\alpha = 0.05$	15.6	15.9	16.5	16.6	17.0
$\alpha = 0.01$	54.8	59.4	62.6	63.0	64.6
$\alpha = 0.005$	84.5	94.7	107.3	113.5	116.0
$\alpha = 0.001$	138.7	181.4	255.6	314.8	377.0

Table 3.2: Experimental results in 3D: the average sizes of the sampling sets

Table 3.4 shows our experimental results in 4D. As we see from the table, the results are generally similar to those in 3D while the sizes of the sampling sets get even larger. Likewise, we find that the sizes of the sampling sets are not sensitive to n when the datasets are large enough in terms of the accuracy requirement. Also, compared with the original datasets, the sampling sets are indeed small-sized, e.g., we only need to sample about 2000 points from a 200K dataset in order to restrict the maximum error to 0.001. And according to the statistics, the growth of the size of the sampling set with respect to α is between $O(\alpha^{-1})$ and $O(\alpha^{-1.5})$.

4D	n =10K	n =20K	n =50K	n =100K	n =200K
$\alpha = 0.1$	19.9	19.6	19.5	19.9	19.4
$\alpha = 0.05$	43.2	43.8	47.1	48.5	48.4
$\alpha = 0.01$	218.6	262.0	311.3	339.4	360.3
$\alpha = 0.005$	314.9	413.0	563.7	686.9	791.0
$\alpha = 0.001$	434.3	633.5	1041.2	1483.3	2087.5

Table 3.3: Experimental results in 4D: the average sizes of the sampling sets

3.5 Proofs

3.5.1 Proof of Theorem 3.1

Algorithm $\bar{A}$ repeatedly uses the top-1 sampling algorithm A on D and excludes the sampling points from D, and does so k times. Let S_i be the sampling set we find by the i-th call of A. Obviously, $S_1 \cup S_2 \cup \cdots \cup S_k = S^*$ and $S_i \cap S_j = \varnothing$ for any $i \neq j$. According to the definition, l_1 is the smallest index at which the true result is superior than the one derived from S^*. That means, among $\{a_{\pi_1}, a_{\pi_2}, \ldots, a_{\pi_{l_1}}\}$, there must exist one point which is not in S^*, which we denote by a^*. On the other hand, since the sampling set obtained by A is guaranteed to have an overall top-1 error less than α, we have

$$Err(\phi_q(D - \bigcup_{j=1}^{i-1} S_j), \phi_q(S_i)) \leq \alpha, \tag{3.2}$$

for any $i \in \{1, 2, \ldots, k\}$. From another fact $a^* \in D - \bigcup_{j=1}^{i-1} S_j$ (because $a^* \in D$ and $a^* \notin S^*$), we have

$$q \cdot a^* \leq \phi_q(D - \bigcup_{j=1}^{i-1} S_j). \tag{3.3}$$

Combining Equation 3.2 and 3.3, we conclude that either $q \cdot a^* < \phi_q(S_i)$ or

$$Err(q \cdot a^*, \phi_q(S_i)) \leq \alpha,$$

which indicates we have at least k data points $c_1, c_2, \ldots, c_k$ in S^* such that for any c_i either $q \cdot a^* < q \cdot c_i$ or

$$Err(q \cdot a^*, q \cdot c_i) \leq \alpha.$$

Thus, we conclude $Err(q \cdot a^*, q \cdot b_{\pi'_k}) \leq \alpha$. And since $q \cdot a^* \geq q \cdot a_{\pi_{l_1}}$ ($a^* \in \{a_{\pi_1}, a_{\pi_2}, \ldots, a_{\pi_{l_1}}\}$), we can replace a^* by $a_{\pi_{l_1}}$ to get $Err(q \cdot a_{\pi_{l_1}}, q \cdot b_{\pi'_k}) \leq \alpha$, which implies our final conclusion

$$Err(q \cdot a_{\pi_{l_1}}, q \cdot b_{\pi'_{l_i}}) \leq \alpha,$$

for $i \in \{1, 2, \ldots, k'\}$. □

3.5.2 Proof of Theorem 3.2

Clearly, v satisfies the two conditions only when

$$\begin{cases} v_{z_1} = 0, \\ \ldots, \\ v_{z_{d-c}} = 0, \\ v \cdot (s_{\pi_1} - s_{\pi_2}) = 0, \\ \ldots, \\ v \cdot (s_{\pi_1} - s_{\pi_c}) = 0, \end{cases}$$

where v_i is the i-th component of v. According to Equation 3.1, the above $(d-1)$ linear equations are independent. Furthermore, the solutions of this system are invariant in terms of scaling. Thus, by restricting $\|v\|_2 = 1$, we have exactly two solutions. Among them, at most one solution has all the components nonnegative. Thus, there exists at most one unit vector v satisfying the two conditions. □

3.5.3 Proof of Theorem 3.3

Suppose $S = \{s_1, \ldots, s_m\}$. Let $D \supseteq S$ be any dataset and q be any query vector. Also, let a^* be the top-1 point of D under q. Without loss of generality, we assume s_1 is the top-1 point of S under q. For any vector v with nonnegative components, define

$$f(v) = \frac{v \cdot a^* - \phi_v(S)}{v \cdot a^*}.$$

Clearly, $f(v) \leq Err(\phi_v(D), \phi_v(S))$ for any v and $f(q) = Err(\phi_q(D), \phi_q(S))$. Thus, to complete the proof, it suffices to show that

$$f(q) \leq \max_{v \in V} f(v).$$

Consider the vector set

$$V' = \left\{ \frac{\lambda}{v \cdot a^*} v \;\middle|\; v \in V \right\},$$

where $\lambda = q \cdot a^*$. Since the function f is invariant in terms of vector scaling, we have

$$\max_{v \in V} f(v) = \max_{v' \in V'} f(v').$$

Note that for any $v' \in V'$, $f(v') = 1 - \phi_{v'}(S)/\lambda$. And $f(q) = 1 - \phi_q(S)/\lambda$. That means to show $f(q) \leq \max_{v' \in V'} f(v')$ is equivalent to proving

$$\phi_q(S) \geq \min_{v' \in V'} \phi_{v'}(S).$$

We formulate the following optimization problem

$$\begin{aligned} \min_w \quad & \phi_w(S) \\ \text{s.t.} \quad & w_1 \geq 0, \\ & \ldots, \\ & w_d \geq 0, \\ & w \cdot (b^* - s_1) \geq 0, \\ & \ldots, \\ & w \cdot (b^* - s_m) \geq 0, \\ & w \cdot a^* = \lambda. \end{aligned}$$

This is a typical linear programming problem. Let F be its feasible region. It is easy to see that

• F is bounded and $q \in F$;

• the vertices of F are critical to S and thus in V' (because of the last constraint). Therefore, there exists $v' \in V'$ such that $\phi_q(S) \geq \phi_{v'}(S)$. As a result, we can conclude that

$$Err(\phi_q(D), \phi_q(S)) = f(q) \leq \max_{v' \in V'} f(v') = \max_{v \in V} f(v) \leq \max_{v \in V} \{Err(\phi_v(D), \phi_v(S))\}$$

and

$$E_S = \sup_q \{Err(\phi_q(D), \phi_q(S))\} = \max_{v \in V} \{Err(\phi_v(D), \phi_v(S))\}.$$

□

3.5.4 Proof of Theorem 3.4

For convenience, when proving Theorem 3.4, we let $A = s_i$, $B = s_{i+1}$, $P = t_i$, $D = s'$, $A' = u$, $B' = v$. Figure 3.3 shows an example of the positions of these points with the new notations. Also, we define the following quantities for our proof ($S_\triangle$ denotes the area of the triangle):

(1) $e = S_{\triangle APB}/S_{\triangle AOB}$,

(2) $e_1 = S_{\triangle AA'D}/S_{\triangle AOD}$,

(3) $e_2 = S_{\triangle BB'D}/S_{\triangle BOD}$.

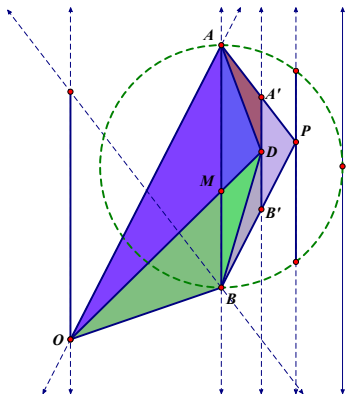


Figure 3.3: The first round of expanding S

First, we prove the first inequality. It is easy to see that

$$\begin{aligned} e &= \frac{E}{1-E}, \\ e_1 &= \frac{E_1}{1-E_1}, \\ e_2 &= \frac{E_2}{1-E_2}. \end{aligned}$$

Thus, it suffices to show $e_1 + e_2 \leq e$. Obviously, if we fix A, B, P, D, and move O on the line l_O (which is parallel to AB), the value of e is always fixed (while e_1 and e_2 may change). So we only need to find a position for O (on l_O) which can get e_1+e_2 maximum and show the maximum is less than or equal to e. According of the definitions of A, B,

P (indeed the definitions of s_i, s_{i+1}, t_i in Section 3.3), we know $\angle OAP + \angle APB \geq \pi$ and $\angle OBP + \angle APB \geq \pi$. That means the position of O is limited on the segment O_1O_2, where O_1 and O_2 are the points on l_O such that $O_1B \parallel AP$ and $O_2A \parallel BP$. We claim that, if A, B, P, D are fixed, $e_1 + e_2$ gets maximum when O coincides one of the two extreme points, O_1 and O_2. To prove this, we note that $S_{\triangle AA'D}$ and $S_{\triangle BB'D}$ will not change when moving O on l_O. Furthermore, $S_{\triangle AOD} + S_{\triangle BOD}$ is also fixed. Thus, according to the definitions of e_1 and e_2, we know $e_1 + e_2$ gets maximum when $S_{\triangle AOD}$ or $S_{\triangle BOD}$ gets extreme value, i.e., O coincides O_1 or O_2. Without loss of generality, we just assume O coincides O_2 (Figure 3.3). Then we have $OA \parallel BP$ so that $O = \frac{1}{e}(B - P) + A$. Assume

- $A' = tA + (1 - t)P$;
- $B' = tB + (1 - t)P$;
- $D = sA' + (1 - s)B'$;
- $M = xO + (1 - x)D = yA + (1 - y)B$.

We get that

$$\begin{aligned}
& yA + (1 - y)B \\
= \;& xO + (1 - x)D \\
= \;& x\left[\frac{1}{e}(B - P) + A\right] + (1 - x)[sA' + (1 - s)B'] \\
= \;& x\left[\frac{1}{e}(B - P) + A\right] + (1 - x)\{s[tA + (1 - t)P] + (1 - s)[tB + (1 - t)P]\} \\
= \;& [x + (1 - x)st]A + \left[\frac{x}{e} + (1 - x)(1 - s)t\right]B + \left[(1 - x)(1 - t) - \frac{x}{e}\right]P.
\end{aligned}$$

That means

$$\begin{aligned}
y &= x + (1 - x)st, \\
1 - y &= \frac{x}{e} + (1 - x)(1 - s)t, \\
0 &= (1 - x)(1 - t) - \frac{x}{e}.
\end{aligned}$$

And the corresponding solution is

$$\begin{aligned} x &= \frac{et-e}{et-e-1}, \\ y &= \frac{st-et+e}{-et+e+1}. \end{aligned}$$

Once we get the expression for y, we can represent e_1 and e_2 as

$$\begin{aligned} e_1 &= \frac{S_{\triangle AA'D}}{S_{\triangle AOD}} = \frac{(1-s)t(1-t)S_{\triangle APB}}{(1-y)(S_{\triangle AOB}+(1-t)S_{\triangle APB})} = \frac{(1-s)t(1-t)e}{(1-y)[1+(1-t)e]} \\ &= \frac{t(1-t)(1-s)e}{1-st}, \end{aligned}$$

$$e_2 = \frac{S_{\triangle BB'D}}{S_{\triangle BOD}} = \frac{st(1-t)e}{y[1+(1-t)e]} = \frac{t(1-t)se}{st-e(1-t)}, \text{ similarly.}$$

Thus, we have

$$\begin{aligned} \frac{e_1+e_2}{e} &= t(1-t)\left(\frac{1-s}{1-st}+\frac{s}{st+e(1-t)}\right) \\ &\leq t(1-t)\left(\frac{1-s}{1-st}+\frac{1}{t}\right) \\ &= t\frac{(1-t)(1-s)}{1-st}+(1-t) \\ &\leq t+(1-t) \\ &= 1, \end{aligned}$$

which implies $e_1+e_2 \leq e$, i.e.,

$$\frac{E_1}{1-E_1}+\frac{E_2}{1-E_2} \leq \frac{E}{1-E}.$$

Then we prove the second inequation. Obviously, we only need to prove the maximum of $\max\{E_1, E_2\}$ is less than or equal to $E/(E+2\sqrt{E(1-E)}+1)$. And it is easy to see $\max\{E_1, E_2\}$ gets maximum when D coincides A' or B'. Without loss of generality, we assume D coincides A'. In this situation, $\max\{E_1, E_2\} = E_2$. According to the

definition of e_2, we know E_2 gets maximum if and only if e_2 gets maximum. Thus, we study e_2 instead of E_2. It is easy to see

$$e_2 = \frac{S_{\triangle BB'D}}{S_{\triangle BOD}} = \frac{S_{\triangle BB'A'}}{S_{\triangle BOA'}} = \frac{t(1-t)S_{\triangle APB}}{tS_{\triangle OPB} + (1-t)S_{\triangle AOB}} = \frac{et(1-t)S_{\triangle AOB}}{tS_{\triangle OPB} + (1-t)S_{\triangle AOB}}.$$

As we see from the above equation, if t, e, $S_{\triangle AOB}$ are all fixed, e_2 gets maximum when $S_{\triangle OPB}$ gets minimum. Another fact is, while O moves on O_1O_2 (with other points fixed), the values of t, e, $S_{\triangle AOB}$ are always fixed and $S_{\triangle OPB}$ gets minimum when O coincides O_2. Thus, it suffices to study e_2 in the case of $O = O_2$. Since $AO \parallel PB$ in this case, we readily have $S_{\triangle OPB} = S_{\triangle APB}$ so that

$$e_2 = \frac{et(1-t)S_{\triangle AOB}}{etS_{\triangle AOB} + (1-t)S_{\triangle AOB}} = \frac{et(1-t)}{et + (1-t)}.$$

It is easy to see e_2 get maximum when $t = (1-\sqrt{e})/(1-e)$ and the corresponding maximum is

$$\frac{e}{e - 2\sqrt{e} + 1}.$$

This is already the overall maximum of e_2 for any case. Since $e_2 = E_2/(1-E_2)$ and $e = E/(1-E)$, we have

$$E_2 \leq \frac{E}{E + 2\sqrt{E(1-E)} + 1},$$

which readily implies

$$\max\{E_1, E_2\} \leq \frac{E}{E + 2\sqrt{E(1-E)} + 1},$$

completing the proof. □

3.5.5 Proof of Corollary 3.1

It is easy to see that after the first round (adding the rightmost and topmost points into S), the only blind triangle has a causing error at most 0.5. According to the second inequality of Theorem 3.4, if the maximum of the causing errors of all valid blind triangles in the ith round is bounded by u_i, then the maximum in the $(i+1)$st round is bounded by

$$u_{i+1} = \frac{u_i}{u_i + 2\sqrt{u_i(1-u_i)} + 1}.$$

And we know $u_1 = 0.5$, thus it can be verified that

$$u_i = \frac{1}{i^2 + 1}.$$

For any given α, we let

$$r = \left\lceil \sqrt{\frac{1}{\alpha} - 1} \right\rceil.$$

Since $u_r \leq \alpha$, after r rounds of expansion, we can get the causing errors of all valid blind triangles to be less than or equal to α, which means the sampling process stops after r rounds. Thus, we have this corollary. □

3.5.6 Proof of Theorem 3.5

We note that Algorithm 3 just uses the framework of Algorithm 2 to do k iterations of top-1 sampling with the same α. So we only need to show that the bound for each iteration of top-1 sampling is $O(\alpha^{-1.5})$. Since each detection can at most get one new point into the sampling set, the number of the sampling data points is dominated by the number of detections. Furthermore, each detection corresponds to a particular valid blind triangle (except the first two detections for the rightmost and topmost points) so that it suffices to show the total number of valid blind triangles (in all rounds) is $O(\alpha^{-1.5})$. We regard each valid blind triangle as a node of a binary tree in which one node x is a child of another node y if and only if the corresponding blind triangle of x is generated directly by the corresponding blind triangle of y. This is a full binary tree because if a valid blind triangle generates new triangles, it always generates two. It is easy to see that, in this binary tree, each level of nodes corresponds to the valid blind triangles in one round. We give each node x a weight

$$w(x) = \frac{E_x}{1 - E_x},$$

where E_x denotes the causing error of the corresponding blind triangle of x. For any non-leaf node x, it is easy to see

$$w(x) \geq \frac{\alpha}{1 - \alpha},$$

because only the blind triangles with the causing errors larger than or equal to α may generate new triangles (we will not find a useful point in a blind triangle with causing

error less than α). Now, we delete all of the leaves in this binary tree to get a new tree. We denote the original tree by T and the new one by T'. According to the property of a binary tree, we have

$$|T| < 2|T'| + 1.$$

Thus, it suffices to show $|T'|$ is bounded by $O(\alpha^{-1.5})$. According to Theorem 3.4, we know the weight of a node is always larger than or equal to the sum of the weights of its two children. The weight of the root of T' is at most 1 (after the first round of expansion, the only blind triangle has a causing error less than or equal to 0.5). And the weights of the leaf nodes of T' are at least $\alpha/(1-\alpha)$ since they are all non-leaf nodes in T. That means the number of the leaf nodes of T' is at most $(1-\alpha)/\alpha$, which is $O(\alpha^{-1})$. Furthermore, according to Corollary 3.1, we know the height of T' (also T) is bounded by $O(\alpha^{-0.5})$. Since the total number of the nodes of a tree is always less than the product of the height and the number of the leaf nodes, we can finally conclude $|T'|$ is bounded by $O(\alpha^{-1.5})$, thus completing the proof. □

Chapter 4

Stochastic line arrangement in $\mathbb{R}^2$

In this chapter, we extend the conventional line arrangement to the stochastic setting and study its underlying combinatorial complexity. We give an efficient algorithm to compute the most-likely k-topmost lines over the entire line arrangement, which also implies an efficient solution to the stochastic version of the preference top-k query we studied in Chapters 2 and 3. We also propose an application of our results to the stochastic Voronoi Diagram problem in $\mathbb{R}^1$.

4.1 Problem definition and main result

Let F be a set containing n lines in $\mathbb{R}^2$, i.e., $F = \{f_1, f_2, \ldots, f_n\}$, where $f_i(x) = k_i x + b_i$. Here, k_i and b_i can be any reals but cannot both be zero. For the convenience of our discussion, we make three assumptions about F:

1. $k_1 < k_2 < \cdots < k_n$, which immediately implies that any two lines have a unique intersection.

2. No three lines have a common intersection.

3. No two intersection points have the same x-coordinate.

For any x-coordinate q, we define the k-topmost lines of F at q as a k-element *ordered* sequence $(f_{l_1}, f_{l_2}, \ldots, f_{l_k})$, in which f_{l_i} has the i-th greatest function value (i.e., y-value) at q among all the lines in F (this implies $f_{l_1}(q) > f_{l_2}(q) > \cdots > f_{l_k}(q)$).

If F is stochastic, i.e., each line f_i has an existence probability of p_i, the true k-topmost lines at q is unknown beforehand. However, we can instead compute the *most likely k-topmost lines* at q as the k-element sequence that has the highest probability to be the true k-topmost lines, where the likelihood of each k-element sequence $S = (f_{l_1}, f_{l_2}, \ldots, f_{l_k})$ is defined by

$$L(S) = \prod_{\forall f_i \in S} p_i \times \prod_{\forall f_i \notin S, f_i(q) > f_{l_k}(q)} (1 - p_i).$$

For example, in Figure 4.1, f_1–f_4 are lines with increasing slope. Let us assume their existence probabilities are $p_1 = 0.9$, $p_2 = 0.5$, $p_3 = 0.4$, and $p_4 = 0.1$. If $k = 2$, then there are 6 candidates for the k-topmost lines at q, namely, $S_1 = (f_3, f_4)$, $S_2 = (f_3, f_2)$, $S_3 = (f_3, f_1)$, $S_4 = (f_4, f_2)$, $S_5 = (f_4, f_1)$, $S_6 = (f_2, f_1)$. Their corresponding likelihoods are

$$\begin{aligned}
L(S_1) &= 0.4 \cdot 0.1 = 0.04, \\
L(S_2) &= 0.4 \cdot 0.5 \cdot (1 - 0.1) = 0.18, \\
L(S_3) &= 0.4 \cdot 0.9 \cdot (1 - 0.1) \cdot (1 - 0.5) = 0.162, \\
L(S_4) &= 0.1 \cdot 0.5 \cdot (1 - 0.4) = 0.03, \\
L(S_5) &= 0.1 \cdot 0.9 \cdot (1 - 0.4) \cdot (1 - 0.5) = 0.027, \\
L(S_6) &= 0.5 \cdot 0.9 \cdot (1 - 0.4) \cdot (1 - 0.1) = 0.243.
\end{aligned}$$

Therefore, $S_6 = (f_2, f_1)$ is the most likely k-topmost lines at q, even though the two lines are at the very bottom at x-coordinate q.

Let F be stochastic. Let $C = \{c_1, c_2, \ldots, c_m\}$ be the set of the intersection points generated by all line pairs of F, where $m = \binom{n}{2}$, and $c_i = (x_i, y_i)$. Without loss of generality, we assume that $x_1 < x_2 < \cdots < x_m$. If we draw a vertical line passing through each intersection, then the plane is partitioned into $m + 1$ open strips: $X_0 = (-\infty, x_1)$, $X_1 = (x_1, x_2)$, $X_2 = (x_2, x_3)$, …, $X_m = (x_m, +\infty)$. (Here, we only show the range of x-coordinates since the range of y-coordinates is always $(-\infty, +\infty)$. Refer to Figure 4.1 for an example.) Obviously, in each strip, the most likely k-topmost lines is the same at all x-coordinates. Thus, we can obtain a sequence $A_0, A_1, A_2, \ldots, A_m$, where A_i denotes the k-topmost lines of the strip X_i. These sequences actually depict

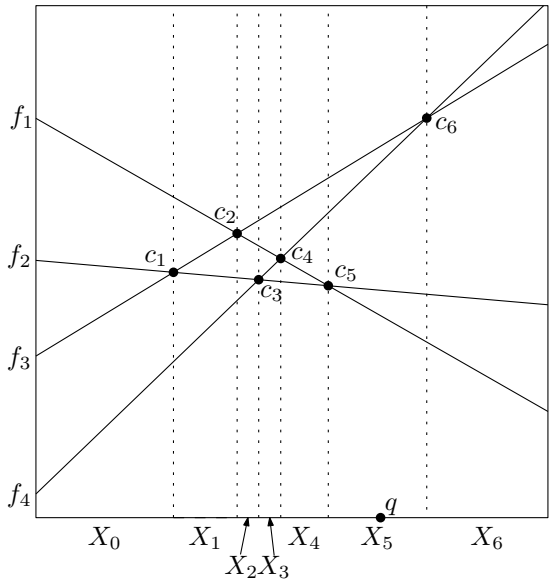


Figure 4.1: An example of a stochastic line arrangement

the entire most likely k-topmost lines for the most likely line arrangement at each x-coordinate. We now define

$$cnt = \sum_{i=1}^{m} (1 - \delta(A_{i-1}, A_i)),$$

where function $\delta(a, b)$ is 1 if $a = b$, and 0 otherwise. (Note that A_i's are all ordered sequences. Thus, not only different elements, but also different order will result in different sequences. E.g., $(f_1, f_2, f_3) \neq (f_1, f_3, f_2)$.) Intuitively, *cnt* counts the number of distinct k-sequences among all A_i's. Assume that the line set F is given but the existence probability of each line is a random variable, and is undetermined beforehand. We then have the following theorem.

Theorem 4.1. *Let F be a set of stochastic lines in $\mathbb{R}^2$, where the existence probabilities $p_1, p_2, \ldots, p_n$ satisfy an identical distribution and are independent of each other. Let cnt be the number of distinct sequences of the most-likely k-topmost lines in the arrangement of F (i.e., taken over all x-coordinates). Then the expected value, E_{cnt}, of cnt is $O(kn)$.*

Thus, we can spend $O(k)$ space for each k-element sequence, and therefore it is possible to store all the distinct A_i's (i.e., the set of all most likely k-topmost lines in

the arrangement of F) in $O(k \cdot kn) = O(k^2 n)$ expected space. Readers may wonder whether two consecutive sequences will have most elements in common so that we can apply persistence [29] to reduce the space, based on the "common" intuition that only two lines swap their positions after crossing an intersection Unfortunately, the answer is no, and we give an counterexample below in which the two consecutive sequences are totally differently.

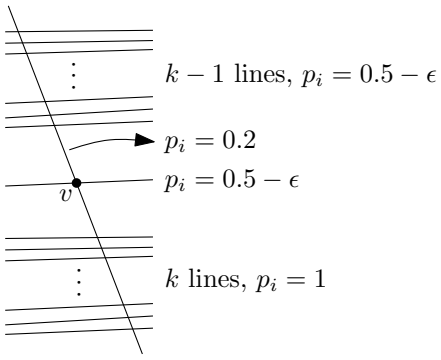


Figure 4.2: An example illustrating that the difference between two consecutive sequences can be huge.

As shown in Figure 4.2, there are $k-1$ lines at the top, one line in the middle and k lines at the bottom, where the top and middle lines have existence probability $p_i = 0.5 - \epsilon$, and the bottom lines have $p_i = 1$. Here ϵ is a very small positive number. Also, there is a line of negative slope, that has existence probability $p_i = 0.2$, cutting all the lines mentioned above; let v be the intersection point of this line with the middle line. Now, let us consider the two strips just to the left and right of v. Just to the left of v, it is clear that the bottom k lines are the best and the corresponding likelihood is $0.8 \cdot (0.5 + \epsilon)^k$. To the right, the highest k lines have probability $(0.5 - \epsilon)^k$ and the bottom k have probability $0.8 \cdot (0.5 + \epsilon)^k$. Any other combination is worse. When k is given, we can always find a sufficiently small ϵ such that $(0.5 - \epsilon)^k > 0.8 \cdot (0.5 + \epsilon)^k$. Thus, the most likely k-topmost lines just to the left of v (i.e., the k bottom lines) are completely different from the most likely k-topmost lines just to the right of v (the k highest lines).

Remark. The reason that we are interested in the expected size of *cnt* instead of the worst case size is that we can show (by concrete example) that, in the worst case, *cnt*

can be as large as $\Theta(n^2)$ even when $k = 1$. A detailed worst case example will be given and discussed in Section 4.4.1.

4.2 Proof of Theorem 4.1

In this section, we first introduce some basic definitions regarding the intersection points of the stochastic lines and their probability distribution. We then show some critical lemmas, and finally use them to establish the proof of Theorem 4.1.

Definition 4.1. *An intersection point $c_i \in C$ is called* **valid** *if $A_{i-1} \neq A_i$, and* **invalid** *otherwise.*

By Definition 4.1, *cnt* can be regarded as the number of valid intersection points in C. In other words, we can define m random variables, Y_1, Y_2, ..., Y_m, where

$$Y_i = \begin{cases} 0, \text{if } c_i \text{ is invalid,} \\ 1, \text{if } c_i \text{ is valid.} \end{cases}$$

Then, *cnt* is also a random variable, and can be written as

$$cnt = Y_1 + Y_2 + \cdots + Y_m. \tag{4.1}$$

Definition 4.2. *The* **depth** *of an intersection $c_i \in C$ is defined as the number of lines in F which are strictly above c_i. (A line f_j is strictly above $c_i = (x_i, y_i)$ if and only if $f_j(x_i) > y_i$.)*

We use d_1, d_2, ..., d_m to denote the depth of c_1, c_2, ..., c_m respectively. For example, in Figure 4.1, the depths of c_1, ..., c_6 are 1, 0, 2, 1, 2, 0, respectively. Also, note that the range for any d_i is $[0, n-2]$ since two lines are always needed to form an intersection. The first lemma below shows the relationship between the number and the depths of the intersection points.

Lemma 4.1. *For a particular depth $d \leq n-2$, the number of intersection points in C with depths no more than d is at most*

$$(n-1) + (n-2) + (n-3) + \cdots + (n-d-1) = (2n-d-2)(d+1)/2.$$

(See Section 4.6.1 for a proof.)

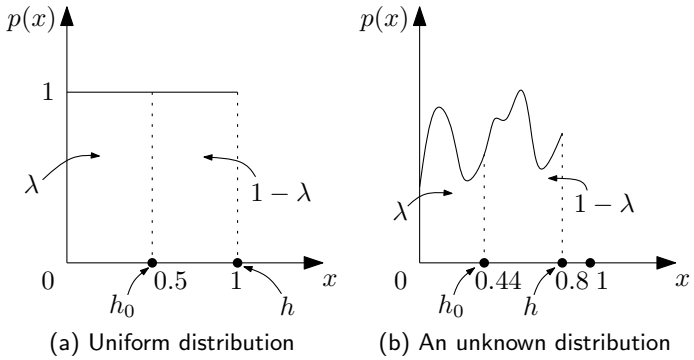
Regarding the probability distribution of each stochastic line, we let $p : [0,1] \to [0,+\infty)$ be the distribution function of p_1, p_2, ..., p_n, which satisfies $\int_0^1 p(x)dx = 1$. Suppose $S = \{s | \int_s^1 p(x)dx > 0\}$. Let h be the least upper bound of S, i.e., $h = \sup S$, and

$$h_0 = \frac{h}{1+h}.$$

We also define

$$\lambda = \int_0^{h_0} p(x)dx,$$

where λ is a constant strictly between 0 and 1 and only depending on the given distribution. Figure 4.3a shows an example of a uniform distribution, where $h = 1$ and $h_0 = 0.5$; Figure 4.3b indicates an example of an unknown distribution where $h = 0.8$ and $h_0 = 4/9$.



(a) Uniform distribution (b) An unknown distribution

Figure 4.3: Two examples illustrating the probability distribution

We then introduce a crucial necessary condition for an intersection point to be valid.

Lemma 4.2. *An intersection point $c_i = (x_i, y_i)$ is valid only if the number of lines that are strictly above c_i and have existence probability greater than h_0 is less than k.* (See Section 4.6.2 for a proof.)

Define a map $u : \mathbb{N} \to (0,1]$ as

$$u(d) = \begin{cases} 1 & d < k, \\ \sum_{j=0}^{k-1} \binom{d}{j} \lambda^{d-j}(1-\lambda)^j & d \geq k. \end{cases} \tag{4.2}$$

Function $u(d)$ indeed denotes the probability for an intersection point with depth d to have at most $k-1$ lines above it with p-values greater than h_0. Note that we do not put any constraint on the variable d, and treat $u(d)$ as very general function in order to do some relaxation later.

Then, the necessary condition in Lemma 4.2 immediately implies that

$$\Pr\{c_i \text{ is valid}\} \leq u(d_i).$$

Recall the definition of $cnt = Y_1 + Y_2 + \cdots + Y_m$, we have

$$\mathrm{E}_{cnt} = \sum_{i=1}^{m} \Pr\{c_i \text{ is valid}\} \leq \sum_{i=1}^{m} u(d_i).$$

In order to prove Theorem 4.1, we still need to find an upper bound for $\sum_{i=1}^{m} u(d_i)$, which is shown in the following lemma.

Lemma 4.3.

$$\sum_{i=1}^{m} u(d_i) \leq \left(\sum_{d=0}^{n-2} u(d) \right) \cdot n.$$

(See Section 4.6.3 for a proof.)

With the lemmas above, we can finally prove Theorem 4.1 as follows.

Proof of Theorem 4.1.

Based on Lemma 4.3, we know that

$$\sum_{i=1}^{m} u(d_i) \leq \left(\sum_{d=0}^{n-2} u(d) \right) \cdot n.$$

We now just need to prove that $\sum_{d=0}^{n-2} u(d)$ is $O(k)$. According to Equation 4.2, we can represent $u(d)$, when $d \geq k$, in a recursive form as

$$u(d) = u(d-1) - \binom{d-1}{k-1} \lambda^{d-k} (1-\lambda)^k.$$

The underlying meaning of the recursive form is shown as follows: $u(d)$ depicts the probability for an intersection point of depth d to have at most $k-1$ lines above it that have existence probability greater than h_0. Let us focus on the lowest line which

is above c_i. (See Figure 4.4 for an example, where the lowest line above c_i is the marked in bold and dashed.) If we assume that line has probability at most h_0, then we immediately have $u(d) = u(d-1)$. However, the assumption is not always true, i.e., some portion of $u(d-1)$ could be invalid and therefore needs to be excluded. That invalid portion corresponds to the case when the lowest line above c_i has probability greater than h_0, and moreover, there are exactly $k-1$ lines among the top $d-1$ lines that have existence probability greater than h_0. Clearly, the probability of this invalid case is $\binom{d-1}{k-1}\lambda^{d-k}(1-\lambda)^k$.

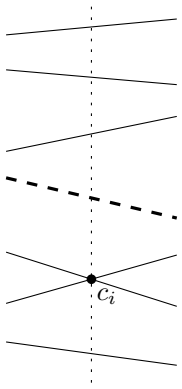


Figure 4.4: Illustrate the underlying meaning of the recursive form of $u(d)$

By applying the recursive form of $u(d)$, we have

$$\begin{aligned}
\sum_{d=k}^{\infty} d \cdot u(d) &= \sum_{d=k}^{\infty} d \cdot \left[u(d-1) - \binom{d-1}{k-1}\lambda^{d-k}(1-\lambda)^k \right] \\
&= \sum_{d=k}^{\infty} d \cdot u(d-1) - \sum_{d=k}^{\infty} d \cdot \binom{d-1}{k-1}\lambda^{d-k}(1-\lambda)^k \\
&= \sum_{d=k-1}^{\infty} (d+1)u(d) - \sum_{d=k}^{\infty} d \cdot \binom{d-1}{k-1}\lambda^{d-k}(1-\lambda)^k \\
&= \sum_{d=k}^{\infty} d \cdot u(d) + \sum_{d=k}^{\infty} u(d) + k \cdot u(k-1) - \sum_{d=k}^{\infty} d \cdot \binom{d-1}{k-1}\lambda^{d-k}(1-\lambda)^k.
\end{aligned}$$

Since $u(k-1) = 1$, the equation above implies that

$$\begin{aligned}\sum_{d=k}^{\infty} u(d) &= \sum_{d=k}^{\infty} d \cdot \binom{d-1}{k-1} \lambda^{d-k}(1-\lambda)^k - k \\ &= \frac{(1-\lambda)^k}{\lambda^k} \sum_{d=k}^{\infty} d \cdot \binom{d-1}{k-1} \lambda^d - k \\ &= \frac{(1-\lambda)^k}{\lambda^k} \cdot k \cdot \sum_{d=k}^{\infty} \binom{d}{k} \lambda^d - k.\end{aligned}$$

Moreover, it can be shown that

$$\sum_{d=k}^{\infty} \binom{d}{k} \lambda^d = \frac{\lambda^k}{(1-\lambda)^{k+1}}. \tag{4.3}$$

(The proof for Equation 4.3 is given in Section 4.6.4.)

Therefore,

$$\sum_{d=k}^{\infty} u(d) = \frac{(1-\lambda)^k}{\lambda^k} \cdot k \cdot \frac{\lambda^k}{(1-\lambda)^{k+1}} - k = \frac{\lambda k}{1-\lambda}.$$

Since $u(d) > 0$ is true for any d, it follows that

$$\sum_{d=k}^{n-2} u(d) < \sum_{d=k}^{\infty} u(d).$$

We then have

$$\begin{aligned}\sum_{d=0}^{n-2} u(d) &= \sum_{d=0}^{k-1} u(d) + \sum_{d=k}^{n-2} u(d) \\ &= k + \sum_{d=k}^{n-2} u(d) \\ &< k + \sum_{d=k}^{\infty} u(d) \\ &= k + \frac{\lambda k}{1-\lambda} = \frac{k}{1-\lambda} = O(k).\end{aligned}$$

Consequently, E_{cnt} is $O(kn)$, completing the proof. □

For quick reference, we include in Table 4.1 the key symbols used in this section.

Symbol	Meaning
F	the line set
f_i	the i-th line in F
p_i	the existence probability of f_i
k_i	the slope of f_i
C	the set of intersection points
c_i	the i-th point in C (sorted by x-coord)
d_i	the depth of c_i
cnt	see Equation 4.1
p	distribution function of $p_1, p_2, \ldots, p_n$
S	$\{s \mid \int_{i=s}^{1} p(x)dx > 0\}$
h	the least upper bound of S
h_0	$h/(1+h)$
λ	$\int_0^{h_0} p(x)dx$
$u(\cdot)$	see Equation 4.2

Table 4.1: List of main symbols used

4.3 An algorithm for computing the most likely k-topmost lines

In this section, we first propose an efficient algorithm for finding the most likely k-topmost lines of any strip, and then we show how to make use of it to compute the k-topmost lines over the entire line arrangement of stochastic lines.

4.3.1 Algorithm for one strip

The most likely k-topmost lines problem in one strip can be equivalently mapped to the following 1D problem. We are given n stochastic points on the x-axis, and the i-th point is at some position x_i and has existence probability p_i. (The value of the x_i's is unimportant; only their order matters.) Given an integer k, $1 \leq k \leq n$, we would like to report the most likely k-rightmost points among the n points.

W.l.o.g., let us assume $x_1 < x_2 < \cdots < x_n$. For any k-subsequence $x_{s_1}, x_{s_2}, \ldots, x_{s_k}$ from left to right, we can compute its likelihood to become the most likely k-rightmost

points as

$$\begin{aligned} L &= p_{s_1} \cdot \prod_{i=s_1+1}^{n} (1-p_i) \cdot \prod_{i=2}^{k} \frac{p_{s_i}}{1-p_{s_i}} \\ &= p_{s_1} \cdot \prod_{i=s_1+1}^{n} (1-p_i) \cdot \prod_{i=2}^{k} \bar{p}_{s_i}. \quad (\bar{p}_{s_i} \stackrel{\text{def.}}{=\!=} \frac{p_{s_i}}{1-p_{s_i}}) \end{aligned} \tag{4.4}$$

If we fix s_1, i.e., assume the leftmost point of the k-sequence is always x_{s_1}, then the first two terms will both be constants. By maximizing the last term, we get the most likely k-rightmost points where the leftmost point is x_{s_1}. We enumerate all possible x_{s_1} from x_{n-k+1} down to x_1, and maintain a min-heap that stores the $k-1$ points, among $x_{s_1+1}, \ldots, x_n$, with the largest $\bar{p}$ values. It is clear that, at any time of the enumeration, the current x_{s_1} together with the $k-1$ points in the heap will be the most likely k rightmost points whose leftmost point is s_1. Therefore, the one (x_{s_1}) with the largest likelihood together with the corresponding $k-1$ points in the heap will be the most likely k rightmost points taken over all n points. We formally describe the above processes in Algorithm 4. If we assume the n points are pre-sorted, then the bottleneck of the algorithm is the for-loop that performs $n-k+1$ heap operations, and stores those heaps, where the size of the heap is $k-1$. By applying persistence (using path copying alone will be sufficient here) [53], Line 13 can be done in $O(\log k)$ space and time per iteration. Thus, the total runtime for Algorithm 4 is $O(n \log k)$ excluding the pre-sorting time.

Remark. The arrays declared on Line 3-5 are in fact not needed. We define them in the pseudo-code mainly for the algorithm that is described in the next subsection. The reader may also question Line 18, which may be potentially invoked $O(n)$ times, and thus will result in an $O(nk)$ runtime. We can overcome this issue by simply running the for-loop twice. The first round only keeps track of the best position $i^\star$ without updating *argmax*, and in the second round, we update *argmax* exactly once, when $i = i^\star$.

4.3.2 Algorithm over the entire line arrangement

A simple approach for computing the most likely k-topmost lines over the entire line arrangement is to run Algorithm 4 for each strip, which will take $O(n^2 \log n + \binom{n}{2} n \log k) = O(n^3 \log k)$ time, where the first term $(n^2 \log n)$ is for computing all the intersection

Algorithm 4 Most_likely_k-rightmost_points

Input: n sorted 1D stochastic points with existence probabilities $p_1, p_2, \ldots, p_n$, and an integer k, where $n \geq k$.
Output: the most likely k-rightmost points as well as their likelihood, and three auxiliary arrays.
Let l be a new array of size $n-k+1$ with initial value 0.
Let h be a new pointer array of size $n-k+2$ with initial value NULL.
Let π be a new array of size $n-k+2$ with initial value 1.
$f_{n-k+1} \leftarrow \prod_{i=n-k+1}^{n} p_i$
$max \leftarrow -\infty$ ▷ maintain the global max likelihood
$argmax \leftarrow \emptyset$ ▷ maintain the most likely k rightmost points
Build a min-heap H on the $k-1$ points $x_{n-k+2}, \ldots, x_n$, where the keys are $\frac{p_i}{1-p_i}$'s.
Also maintain, for each node x of H, an extra field, named *prod*, recording the product of all the keys of x's subtree.
$prod \leftarrow \prod_{i=n-k+2}^{n} (1-p_i)$ ▷ maintain the middle term of Equation 4.4
for $i \leftarrow n-k+1$ **downto** 1 **do**
    $h_{i+1} \leftarrow H$
    $\pi_{i+1} \leftarrow prod$
    $l_i \leftarrow p_i * \pi_{i+1} * h_{i+1}.prod$
    if $l_i > max$ **then**
        $max \leftarrow l_i$
        $argmax \leftarrow \{x_i\} \cup h_{i+1}$
    end if
    $H.insert(x_i), H.extractMin()$
    $prod \leftarrow prod * (1-p_i)$
end for
$seq \leftarrow argmax.sorted$ ▷ sorted by coordinate of each point
return (max, seq, l, h, π)

points and pre-sorting, and the second term due to running Algorithm 4 $\binom{n}{2}$ times. This can be further improved to $O(n^2 \log n + nk^2 \log k)$, as we show below.

We still use the high level idea of the naive approach, but instead of running the algorithm $\binom{n}{2}$ times, we only call it once for the leftmost strip, and maintain the information, i.e., l, h and π arrays, through the rest of strips from left to right. Formally, let us assume that, in some strip, all the lines are labeled as $f_1, f_2, \ldots, f_n$ from bottom to top. Some two lines f_i and f_{i+1} form an intersection, and after crossing that intersection, the line sequence from bottom to top will be $f_1, f_2, \ldots, f_{i-1}, f_{i+1}, f_i, f_{i+2}, \ldots f_n$. Then it is straightforward to observe that only $f_i, l_i, f_{i+1}, l_{i+1}, \pi_{i+1}$ and h_{i+1} will change, i.e., only $O(1)$ entries of the arrays will change. (See Figure 4.5 for an example of $n = 6$ lines and $k = 3$. The only changes between the two strips are marked in red.)

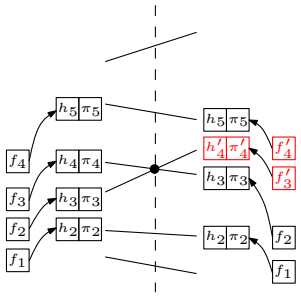


Figure 4.5: Maintaining the information between two consecutive strips, where the entries that will change are marked in red. (The figure is best viewed in color.)

Based on this, we propose our algorithm below (Algorithm 5) for computing the most likely k-topmost lines over the entire line arrangement. To analyze the runtime, first note that computing and pre-sorting all the intersection points as well as computing the most likely k-topmost lines in the leftmost strip takes $O(n^2 \log n)$ time. Second, excluding reporting the output, each iteration of the for-loop requires $O(\log k)$ time only due to performing $O(1)$ operations on heaps of size k, which in total costs $O(n^2 \log k)$. Last, as there are $O(nk)$ changes in expectation due to Theorem 4.1, reporting the most likely k-topmost lines over the entire line arrangement will take $O(nk \cdot k \log k) = O(nk^2 \log k)$ time. Therefore, we finally conclude that, in expectation, the entire algorithm runs in $O(n^2 \log n + nk^2 \log k)$ time.

It is interesting to note that there is an alternative way to judge whether it is

Algorithm 5 Most_likely_k-topmost_lines_in_the_arrangement

Input: n stochastic lines, $f_1, f_2, \ldots, f_n$, of existence probabilities $p_1, p_2, \ldots, p_n$, and an integer k, where $n \geq k$
Output: the most likely k-topmost lines over the entire line arrangement
Compute all the $m = \binom{n}{2}$ intersection points, denoted by $c_1, \ldots, c_m$, of lines $f_1, f_2, \ldots, f_n$, and sort them by increasing x-coordinate.
Rearrange $f_1, \ldots, f_n$ and $p_1, \ldots, p_n$ so that they denote the sequence of stochastic lines, from bottom to top, in the leftmost strip.
Let $\mathcal{F}_1, \ldots, \mathcal{F}_n$ denote the sequence of lines, from bottom to top, in the leftmost strip.
Let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ denote the corresponding probabilities.
$(pre\_max, pre\_seq, l, h, \pi) \leftarrow$ Most_likely_k_rightmost_points(p)
$pre\_x \leftarrow -\infty$
$ans \leftarrow pre\_seq$
for $i \leftarrow 1$ **to** m **do**
    Let lines f_j and f_{j+1} form the intersection point c_i.
    if $j + 1 \leq n - k + 2$ **then**
        $\pi_{j+1} \leftarrow \pi_{j+1} * (p_{j+1}/p_j)$
        $h_{j+1} \leftarrow (h_{j+1} \cup \{f_{j+1}\}) \setminus \{f_j\}$
        $l_j \leftarrow \max(l_j, p_{j+1} * \pi_{j+1} * h_{j+1}.prod)$
        if $j + 2 \leq n - k + 2$ **then**
            $l_{j+1} \leftarrow \max(l_{j+1}, p_j * \pi_{j+2} * h_{j+2}.prod)$
        end if
    end if
    $max \leftarrow \max(l_1, l_2, \ldots, l_{n-k+1})$
    if $(max > pre\_max)$ **or** $(max = pre\_max$ **and** $f_j, f_{j+1} \in pre\_seq)$ **then**
        Let $argmax$ be $\{f_t\} \cup h_{t+1}$, where $l_t = max$.
        $seq \leftarrow argmin.sort$ ▷ sort each line by y-coordinate in the strip
        Report seq as the most likely k-topmost lines over the interval $(pre\_x, c_i.x)$.
        $pre\_seq \leftarrow seq$
    end if
    $pre\_x \leftarrow c_i.x$
    swap(f_j, f_{j+1})
end for
Report pre_seq as the most likely k-topmost lines over the interval (pre_x, ∞).

time to report the current sequence. Indeed, we can apply the necessary condition in Lemma 4.2 to replace the condition of the if-statement on Line 21, i.e., we report the current sequence immediately if the necessary condition is true. By doing that, we still get the correct result, but some interval might be chopped up into several consecutive sub-intervals. Although more ordered sequences are likely to be output in this case, the expected runtime to report still remains the same, i.e., $O(nk^2 \log k)$. This is because the $O(nk)$ bound in terms of the number of changes is derived from the necessary condition. Hence, it is in general a loose bound, which means the number of sequences output by Algorithm 5 can be less than $O(nk)$ in expectation.

4.4 Application: Stochastic Voronoi Diagram in $\mathbb{R}^1$

We first introduce the stochastic version of Voronoi Diagram [50] in 1D, and build bridges between it and the most likely k-topmost lines when $k = 1$. On the one hand, we show that the size of 1D stochastic Voronoi Diagram can be large, which implicitly means that, in the worst case, the number of changes we studied in Theorem 4.1 can be large as well even when $k = 1$. On the other hand, by using Theorem 4.1, we can show that the size of the stochastic Voronoi Diagram in 1D has expected size $O(n)$.

We extend the standard Voronoi Diagram [50] in $\mathbb{R}^1$ to a stochastic version. In the conventional (non-stochastic) case, we are given n points (called *sites*), say $x_1, x_2, \ldots, x_n$, on the x-axis from left to right. It is clear that the midpoints of x_i and x_{i+1}, for $i = 1, \ldots, n-1$, form the boundaries of the Voronoi Diagram. Consequently, $(-\infty, (x_1+x_2)/2)$, $((x_1+x_2)/2, (x_2+x_3)/2)$, $((x_2+x_3)/2, (x_3+x_4)/2), \ldots, ((x_{n-2}+x_{n-1})/2, (x_{n-1}+x_n)/2)$, $((x_{n-1}+x_n)/2, +\infty)$ are the n open cells, and $x_1, x_2, \ldots, x_{n-1}, x_n$ are called the *generators*. Clearly, the space occupied is only linear.

However, if the points are stochastic, the conclusion is not so straightforward. We will first give a formal definition of the problem, then we show that the space can be quadratic, and give a worst case example in Section 4.4.1. Finally, we show how to reduce the stochastic 1D Voronoi Diagram to stochastic line arrangement to get a good expected bound. The reduction is given in Section 4.4.2.

The input, P, contains n stochastic 1D points. Let us assume the i-th point has x-coordinate x_i, and existence probability p_i. We assume all the points are pre-sorted

from left to right, i.e., $x_1 < x_2 < \cdots < x_n$.

For every position $x = q$ on the x-axis, we define, for each point x_i, the likelihood for it to be the closest site to q as

$$L(x_i, q) = p_i \cdot \left(\prod_{\forall (j \neq i) \wedge (|x_j - q| < |x_i - q|)} (1 - p_j) \right).$$

Consequently, the one with the highest likelihood, i.e., $\text{argmax}_{x_i \in P} L(x_i, q)$, becomes the most likely nearest neighbor of q. Now, we can define the cell, $C(x_i)$, of x_i, in the stochastic Voronoi Diagram of P as follows.

$$C(x_i) = \{q | (\text{argmax}_{x_j \in P} L(x_j, q)) = x_i\}$$

In other words, $C(x_i)$ consists of all points $q \in \mathbb{R}^1$ for which x_i is the most-likely closest site. Point x_i is the generator of $C(x_i)$. The stochastic Voronoi Diagram of P is the union of the $C(x_i)$ taken over all $x_i \in P$.

Note that, since P is stochastic, $C(x_i)$ does not necessarily contain only one interval, and some $C(x_i)$ can even be empty. We use $|C(x_i)|$ to denote the number of disjoint intervals in it, where we make a slight abuse of the cardinality function. Then, it is straightforward that the 1D stochastic Voronoi Diagram requires at least $O(\sum_{i=1}^{n} |C(x_i)|)$ space.

How large can the space be? Suppose we draw the $m = \binom{n}{2}$ bisectors determined by every pair of points x_i and x_j. This partitions the entire x-axis into $m + 1$ intervals. For any point in an interval, the ordering of the sites by distance is the same as for any other point in the interval. It follows that all points in an interval have the same likelihood with respect to each site, so the most likely neighbor of all points in an interval is the same and the interval defines a cell in the Voronoi Diagram. Thus, it follows that $O(\sum_{i=1}^{n} |C(x_i)|) = O(m) = O(n^2)$. In Section 4.4.1, we show this bound is indeed tight in the worst case, so there is no hope to come up with a better worst case analysis.

4.4.1 The staircase graph and the worst-case example

In this section, we first show a useful tool called the *Staircase Graph* to best illustrate the stochastic Voronoi Diagram.

We denote the $m+1$ cells created by the $m = \binom{n}{2}$ pairwise bisectors by c_0, c_1, c_2, $\ldots$, c_m from left to right. As observed above, $L(x_i, q)$ will be the same for any q in a fixed cell. Therefore, with a slight abuse of notation, we define $L(x_i, c_j)$ to be the likelihood of x_i to be generator of cell c_j, where $1 \leq i \leq n$ and $0 \leq j \leq m$. Now, for each site x_i, the $m+1$ likelihood values, $L(x_i, c_0)$, $\ldots$, $L(x_i, c_m)$, form a stair-series. There are n sites, which in total form n different stair-series in the plane, and it is obvious that $\operatorname{argmax} L(x_i, c_j), 1 \leq i \leq n$ (the site corresponding to the topmost curve) will be the most likely generator for cell c_j. We call this collection of stair-series a *Staircase Graph*. The upper envelope of the staircase graph corresponds to the stochastic Voronoi Diagram.

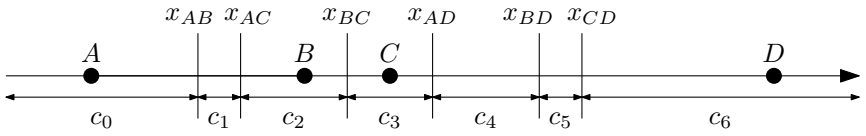


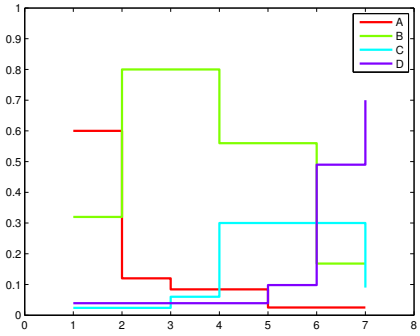
Figure 4.6: An example for illustrating the staircase graph

As a simple example, in Figure 4.6, there are four points, A, B, C and D, on the x-axis from left to right. Let us assume their existence probabilities are $p_A = 0.6$, $p_B = 0.8$, $p_C = 0.3$, and $p_D = 0.7$. All six mid-points are marked by vertical bars, and all seven cells, c_0, $\ldots$, c_6 are also marked. For any given query point $q \in c_0$, A is the nearest site with respect to it. The second nearest site is B, and the third and the fourth will be C and D, respectively. We can record this relative order for A, B, C and D in c_0 by a four-element sequence (A, B, C, D). As soon as q crosses the mid-point x_{AB}, the four-element sequence changes to (B, A, C, D), i.e., the order between A and B is swapped. This pattern holds true when q crosses any mid-point, say $x_{\alpha\beta}$, i.e., before crossing, α and β must be consecutive elements in the sequence, and after crossing, their order will be swapped. Based on this important observation, we can compute the sequences for each of the c_i, and consequently the likelihood for each site for each c_i can be calculated efficiently based on these sequences. Figure 4.7a shows, for each cell, the corresponding 4-element sequence and the likelihood for each site to be the nearest site in that interval. We also plot the four likelihood series (the last four columns of the table) to get the staircase graph shown in Figure 4.7b. From this figure, we know A is the most likely nearest site for cell c_0, B is the most likely nearest site for cells c_1, c_2, c_3,

and c_4, and D is the most likely nearest site for cells c_5 and c_6. Note, in this case, site C is not a generator for any cell, not even of the cell c_3 which contains it! One

Cell	Sequence	$L(A,\cdot)$	$L(B,\cdot)$	$L(C,\cdot)$	$L(D,\cdot)$
c_0	(A,B,C,D)	0.6000	0.3200	0.0240	0.0392
c_1	(B,A,C,D)	0.1200	0.8000	0.0240	0.0392
c_2	(B,C,A,D)	0.0840	0.8000	0.0600	0.0392
c_3	(C,B,A,D)	0.0840	0.5600	0.3000	0.0392
c_4	(C,B,D,A)	0.0252	0.5600	0.3000	0.0980
c_5	(C,D,B,A)	0.0252	0.1680	0.3000	0.4900
c_6	(D,C,B,A)	0.0252	0.1680	0.0900	0.7000

(a) 4-element sequences for all the cells and the likelihood values for all the sites



(b) The staircase graph. Note that the range of x-axis varies from 1 to 7, which corresponds to c_0, $c_1, \ldots, c_6$. (Note that some vertical segments from different stair-series are overlapping.)

Figure 4.7: The statistics table and the corresponding staircase graph

may observe that each stair-series in the staircase graph is a unimodal function. So the question becomes whether it is possible for these n stair-series to interlace one another in order to make the upper envelope sufficiently complicated (i.e., of size $\Theta(n^2)$). The answer is yes, and in the rest of this subsection, we give a concrete example.

A worst-case example: We generate the position and the existence probability of the i-th point as follows, where we consider $n > 3$ points, and $\epsilon > 0$ is a sufficiently

small real number, say $\epsilon < n^{-4}$.

$$p_i = \begin{cases} 1 & \text{if } i = n, \\ \dfrac{p_{i+1}}{1+p_{i+1}} - \epsilon & \text{if } i < n. \end{cases}$$

$$x_i = \begin{cases} 0 & \text{if } i = 1, \\ x_{i-1} + 10^{-i+1} & \text{if } i > 1. \end{cases}$$

In general, the pattern of the x_i sequence is $0, 0.1, 0.11, 0.111, 0.1111 \ldots$ By such construction, intuitively, if we focus on points x_1 and $x_2, x_3, \ldots, x_n$, points $x_2, x_3, \ldots, x_n$ will cluster together, and x_1 will be far away from them, i.e., the midpoint of x_1 and any x_i $(i > 1)$ must lie in the interval $(0, 0.1)$. Indeed, this property recursively applies to any suffix of the input, i.e., $x_i, x_{i+1}, \ldots, x_n$. In other words, if we focus on $x_i, x_{i+1}, \ldots, x_n$, points $x_{i+1}, x_{i+2}, \ldots, x_n$ will cluster, and point x_i will be far away from them. Consequently, the midpoint of x_i and any x_j $(j > i)$ must lie in the interval (x_i, x_{i+1}), and thus all the midpoints from left to right will be ordered as $x_{12}, x_{13}, \ldots, x_{1n}, x_{23}, x_{24}, \ldots, x_{2n}, \ldots, x_{n-1n}$, where x_{ij} denotes the midpoint of x_i and x_j. (Please refer to Figure 4.8 for a four-point example.)

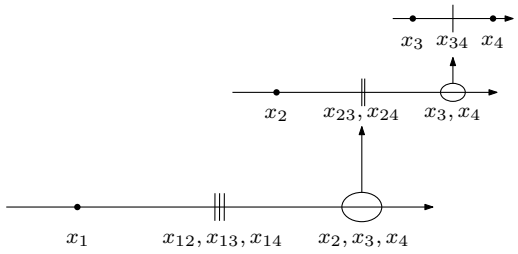


Figure 4.8: A recursive view of the worst case example where $n = 4$

Based on the special positions of the midpoints, one may observe that the n-element sequence in each cell from left to right varies exactly as the behavior of bubble sort. Formally, at the beginning of each "bubble" stage, the leftmost element x_i will be selected to move to position $n+1-i$ in the sequence by swapping with its right neighbor. We call each x_i the *moving element* of each stage, and we have the following important lemma.

Lemma 4.4. *The winner (the one with the largest likelihood) in any cell is the left neighbor of x_i in the corresponding n-element sequence, where x_i is assumed to be the*

moving element. (Note that, if x_i is the first element of the sequence, then the winner will be the last element. This situation happens only once at the very beginning of the entire process when the sequence is $(x_1, x_2, \ldots, x_n)$, and x_1 is the moving element.) (See Section 4.6.5 for a proof.)

Table 4.2 also shows a 4-point example.

c_0 :	1	2	3	$\underline{4}$
c_1 :	$\underline{2}$	$\boxed{1}$	3	4
c_2 :	2	$\underline{3}$	$\boxed{1}$	4
c_3 :	2	3	$\underline{4}$	$\boxed{1}$
c_4 :	$\underline{3}$	$\boxed{2}$	4	1
c_5 :	3	$\underline{4}$	$\boxed{2}$	1
c_6 :	$\underline{4}$	$\boxed{3}$	2	1

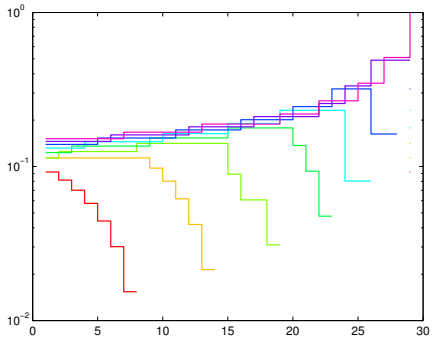
Table 4.2: A four-point example where each line corresponds to an n-element sequence of the corresponding cell. There are 7 cells in total due to $\binom{4}{2}$ midpoints. The moving element of each cell is in the box, and the winner is marked by underscore.

One can easily verify that, by the above construction, the most likely nearest site changes after crossing every midpoint, and thus the upper envelope of the staircase graph corresponding to the above dataset has size $\binom{n}{2} = \Theta(n^2)$. In other words, the $O(n^2)$ space bound for the 1D stochastic Voronoi Diagram is tight, as shown in this example.

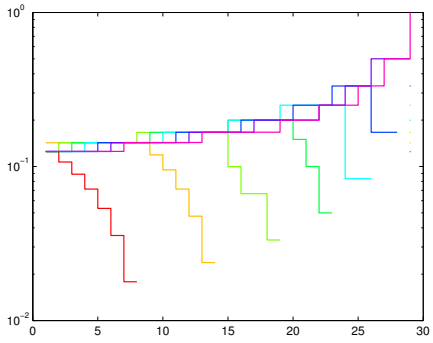
To illustrate the worst case, we generate a dataset using the definition above for $n = 8$. Figure 4.9a lists the detailed input data where we choose $\epsilon = 0.01$, and Figure 4.9b shows the corresponding staircase graph, in a pattern that has changes which scale as $\Theta(n^2)$ if we were to increase n. We can even increase the complexity of the upper envelope by reducing the value of ϵ, and it turns out when ϵ gets sufficiently small, the upper envelope can have $(n-1) + (n-2) + \cdots + 1 = \binom{n}{2}$ changes, which illustrates the fact that the generators of every consecutive pair of cells may be different in some stochastic Voronoi Diagram. Figure 4.9c indicates the case when we set $\epsilon = 0.0001$.

	1	2	3	4	5	6	7	8
x's	-1.0000	0	2.0000	2.1000	2.3000	2.3100	2.3300	2.3310
p's	0.0921	0.1137	0.1412	0.1782	0.2318	0.3189	0.4900	1.0000

(a) Position and existence probability for each point



(b) The staircase graph when $\epsilon = 0.01$.



(c) The staircase graph when $\epsilon = 0.0001$.

Figure 4.9: Worst case data set. Note that the y-axis is in log scale.

4.4.2 Reduction from stochastic Voronoi Diagram to stochastic line arrangement

In the previous section, we showed the size of stochastic Voronoi Diagram can be as bad as $\Theta(n^2)$. However, this worst case can rarely happen, and in most random cases, the size is usually linear or sub-linear. In this section, we develop a proof by reducing the 1D stochastic Voronoi Diagram problem to 2D stochastic line arrangement to show that the expected size of 1D stochastic Voronoi Diagram is again $O(n)$. In fact, the reader might have already noticed that the sequences involved above are very similar to the k-topmost lines in each strip.

The reduction goes as follows. We lift all the n points on the x-axis to the standard parabola $y = x^2$, and for each lifted point create the tangent to the parabola. We denote the tangents of the i-th site by f_i, and f_i and site i have the same existence probability. A well-known fact [27] is that, for any two tangents, say f_i and f_j, the x-coordinate their intersection is exactly the midpoint of site i and site j. Also, since all the sites are sorted from left to right on the x-axis, based on the property of parabola $f_1, \ldots, f_n$ will have increasing slopes. Moreover, we have assumed that no two sites

have the same mid-point, and hence no two intersections of the tangents will have the same x-coordinate. If we draw vertical lines passing through each intersection, we will have $m+1 = \binom{n}{2}+1$ strips, say X_0, X_1, ..., X_m, as we mentioned in Section 4.1. Then there is obviously a one-to-one correspondence between c_i and X_i. More importantly, the n-element sequence in c_i indicates exactly the order of those tangents from top to bottom. Finally, the most likely nearest site for each cell c_i is just the most likely 1-topmost line in strip X_i, and consequently, the stochastic Voronoi Diagram corresponds to the most likely 1-topmost line of the stochastic line arrangement taken over all X_i's.

Figure 4.10 illustrates an example, where A, B, C and D are four given sites, and A', B', C' and D' are the corresponding lifted points. The tangents are shown by bold colored lines, and black dots indicate the intersection points of any two tangents. It is clear that the projection (onto x-axis) of each intersection of two tangents is the midpoint of the two corresponding sites.

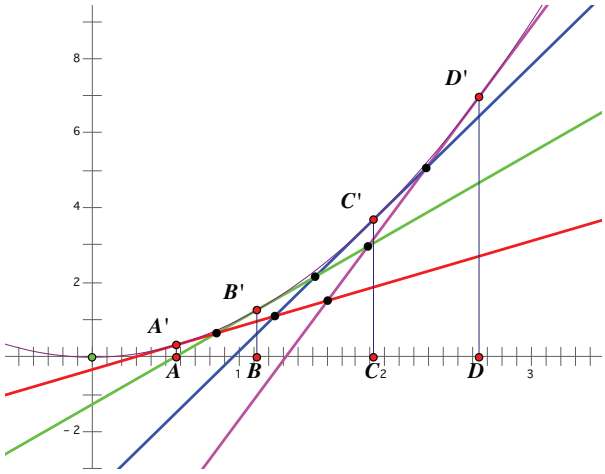


Figure 4.10: The reduction, where we lift all the points to $y = x^2$.

This reduction indeed indicates two facts, one negative and one positive. The negative fact is that we can reduce the worst case example in Section 4.4.1 to the corresponding stochastic line arrangement so that even the most likely top-1 line of the stochastic line arrangement has size $\Theta(n^2)$. Therefore there is no hope for us to derive a good worst case space bound for the stochastic line arrangement.

On the other hand, the positive fact is that, by Theorem 4.1, the expected size of the most likely top-1 line of any stochastic line arrangement is $O(1 \cdot n) = O(n)$. Thus, the stochastic Voronoi Diagram also has expected size $O(n)$. We should say that is under the assumption that there is a fixed probability distribution on the existence probabilities.

4.5 Application: Stochastic preference top-k query in $\mathbb{R}^2$

We propose the stochastic version of the preference top-k query in 2D. Interestingly, fetching the most likely top-k objects with some preference vector can be viewed as a dual version of computing the most likely k-topmost lines over a given set of stochastic lines.

Given n points in $\mathbb{R}^2$, the conventional (i.e., non-stochastic) *preference top-k query* outputs the k points with the largest score with respect to a user specified weighting vector w, where the score of a point, p, with respect to w is defined as the inner product $p \cdot w$. Now, assume that every point is stochastic, and has a certain existence probability. Then, given a specified weighting vector, the top-k points returned by the conventional query might not be very attractive because it is possible that the likelihood that all of them are present is very low. Instead, there may be have a different set, S, of k candidates, that has higher likelihood to be the set of top-k candidates. This means that all the points in S should be present, and any point that is not in S but has a score larger than at least one point in S must not be present. There are $\binom{n}{k}$ possible sets of size k, each with a certain likelihood to exist, and this is the likelihood of that set to become the top-k set. Specifically, we would like to know the one with the largest likelihood, and we call it the *most likely top-k points* with respect to the given weighting vector.

A naïve approach to compute the most likely top-k points with respect to some weighting vector is to enumerate all $\binom{n}{k}$ sets, and compute the likelihood of each set as the product of the existence probabilities of its members and the non-existence of its non-members. Then we choose the maximum, and the corresponding set will be the desired answer. This approach is of course exponential.

On the other hand, we can work in the dual space in which each point becomes a

line, and the query weighting vector becomes a vertical ray at some x-coordinate. The conventional preference top-k query reports the topmost k lines that are hit by the ray. If the problem becomes stochastic, the most likely top-k points with respect to some weighting vector is nothing but the most likely k-topmost lines at the corresponding x-coordinate. Therefore, we can pre-compute the most likely k-topmost lines over the entire dual line arrangement using Algorithm 5 and answer a stochastic preference top-k query efficiently via a simple binary search.

Note that Algorithm 5 reports ordered sequences, which means the reported k points are already sorted by their score. If the user wants the set only, we can omit the sorting on Line 23 so that the pre-processing time can be reduced to $O(n^2 \log n + nk^2)$.

4.6 Proofs

4.6.1 Proof for Lemma 4.1

We first create $(n - d - 1)$ line sets denoted by F_{d+2}, F_{d+3}, ..., F_n, where

$$F_i = \{f_j | f_j \in F, j \leq i\} = \{f_1, f_2, f_3, \ldots, f_i\}.$$

Obviously, these sets are all subsets (and "prefixes") of F satisfying

$$F_{d+2} \subset F_{d+3} \subset F_{d+4} \subset \cdots \subset F_n = F.$$

Let $C_{d+2}, C_{d+3}, \ldots, C_n$ be their corresponding intersection point sets. We then use e_i to denote the number of intersection points with the depth no more than d in C_i. (It should be noted that the depth of an intersection point in C_i is defined in terms of the line set F_i.) We shall prove, inductively, that

$$e_i \leq (i-1) + (i-2) + (i-3) + \cdots + (i-d-1) = \frac{(2i-d-2)(d+1)}{2}$$

for $i = d+2, d+3, \ldots, n$. Clearly, this strong conclusion implies the lemma when $i = n$.

In the case of $i = d + 2$, the conclusion is trivially true since there are in total $\binom{d+2}{2} = (d+2)(d+1)/2$ intersections in C_{d+2}. Now assume that the conclusion is true for the case of $i = t - 1$, where $d + 2 \leq t - 1 < n$, we shall show that it is also true for $i = t$.

Consider the line $f_t \in F_t$. Some intersections in C_t are generated by f_t (with another line), while the others are not. Accordingly, we can partition C_t into two subsets

$$C_t = C'_t \cup C''_t,$$

where C'_t denotes the f_t-generated intersections points and C''_t denotes other intersections. Then e_t can be naturally represented as

$$e_t = e'_t + e''_t,$$

where e'_t denotes the number of intersections with depths no more than d in C'_t and e''_t denotes that in C''_t.

We first consider the size of e'_t. The intersection points in C'_t are generated by f_t so that $|C'_t| = t - 1$. According to our assumption $k_1 < k_2 < \cdots < k_n$, f_t has the largest slope among all the lines in F_t. It readily follows that the $t-1$ intersections in C'_t have the depths of $0, 1, \ldots, t-2$ for just once of each, where the depths $0, 1, \ldots, d$ are what we are interested in. This implies that

$$e'_t = d + 1.$$

Next, consider the size of e''_t. Since e''_t is composed of all the intersection points generated by $\{f_1, f_2, \ldots, f_{t-1}\}$, it is equivalent to C_{t-1}. In other words, for each $c \in C''_t$, we can find a corresponding element in C_{t-1}, say $\bar{c}$. Because of the appearance of f_t in F_t, the depth of c may be equal or greater than the depth of $\bar{c}$. But it is obviously that the depth of c can never be smaller than the depth of $\bar{c}$. Thus, we assert that

$$e''_t \leq e_{t-1}.$$

By the induction hypothesis, we have

$$e_{t-1} \leq (t-2) + (t-3) + \cdots + (t-d-2).$$

Then,

$$\begin{aligned} e_t &= e'_t + e''_t \\ &\leq (d+1) + e_{t-1} \\ &\leq (d+1) + (t-2) + (t-3) + \cdots + (t-d-2) \\ &= (t-2+1) + (t-3+1) + \cdots + (t-d-2+1) \\ &= (t-1) + (t-2) + \cdots + (t-d-1) \\ &= (2t-d-2)(d+1)/2, \end{aligned}$$

which completes the proof.

4.6.2 Proof for Lemma 4.2

We shall prove, by contraposition, that $A_{i-1} = A_i$ (i.e., c_i is invalid) if $|H_i| \geq k$. Assume c_i is generated by two lines f_α and f_β. We first define two label sets

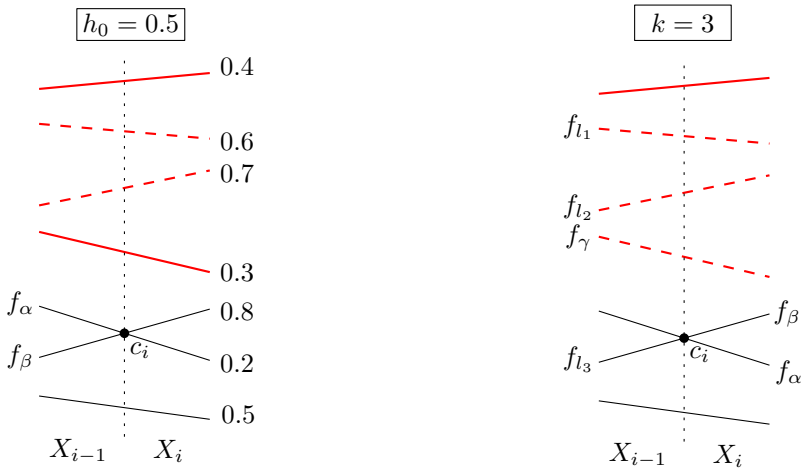
$$\begin{aligned} L_1 &= \{l | f_l(x_i) > y_i\}, \\ L_2 &= \{l | f_l(x_i) \leq y_i\}. \end{aligned}$$

Intuitively, L_1 contains the labels of all the lines that are strictly above c_i, while L_2 contains the labels of all the lines which are strictly below or pass through c_i. Note that L_2 includes both α and β. Recalling the definition of H_i, we have $H_i \subseteq L_1$ since H_i depicts those special lines in L_1 that have p-values greater than h_0. As an example, please refer to Figure 4.11a, where the labels of the red lines belong to L_1, and the labels of the black lines are in L_2. Also, if we assume that $h_0 = 0.5$, then based on the p-values to the right of each line, H_i contains the labels of all dashed red lines.

Let F_1 and F_2 be the sets of lines whose labels are in L_1 and L_2, respectively. Obviously, in both strips X_{i-1} and X_i, the y-values of the lines in F_1 are greater than the y-values of the lines in F_2.

Now consider A_{i-1}, the most likely k-topmost lines in X_{i-1}. Suppose

$$A_{i-1} = (f_{l_1}, f_{l_2}, \ldots, f_{l_k}).$$



(a) Red lines belong to L_1, black lines belong to L_2, and all lines in H_i are shown by dashed red lines.

(b) In this particular case, assume that $A_{i-1} = (f_{l_1}, f_{l_2}, f_{l_3})$ for $k = 3$, and we can construct a new line arrangement $A'_{i-1} = (f_{l_1}, f_{l_2}, f_\gamma)$ by the method introduced in the proof.

Figure 4.11: Two examples illustrating the proof of Lemma 4.2

We shall show that $l_1, l_2, \ldots, l_k$ are all in L_1, i.e., A_{i-1} consists of lines from F_1 only. In fact, we just need to prove that $l_k \in L_1$ since $f_{l_1} > f_{l_2} > \cdots > f_{l_k}$ in X_i, i.e., line f_{l_k} is the lowest in X_i among $f_{l_1}, f_{l_2}, \ldots, f_{l_k}$. For a contradiction, assume $l_k \notin L_1$ (i.e., $l_k \in L_2$). Since we assume that $|H_i| \geq k$, accordingly to the fact that $H_i \subseteq L_1$, we can assert

$$H_i \setminus \{l_1, l_2, \ldots, l_k\} \neq \emptyset.$$

This implies that we can find a label $\gamma \in L_1$ such that

$$(p_\gamma > h_0) \wedge (\gamma \neq l_1, l_2, \ldots, l_k).$$

Now we remove the element f_{l_k} from A_{i-1}, and add f_γ in the appropriate position to get a new sequence

$$A'_{i-1} = (f_{l_1}, f_{l_2}, \ldots, f_\gamma, \ldots, f_{l_{k-1}}).$$

Here, if $f_\gamma > f_{l_1}$ (or $f_\gamma < f_{l_{k-1}}$) in X_{i-1}, A'_{i-1} should be written as $(f_\gamma, f_{l_1}, f_{l_2}, \ldots, f_{l_{k-1}})$ (or $(f_{l_1}, f_{l_2}, \ldots, f_{l_{k-1}}, f_\gamma)$). Please refer to Figure 4.11b for an example.

Now we show that A'_{i-1} indeed has greater likelihood than A_{i-1} to be the k-topmost lines in X_{i-1}, which contradicts to the fact that A_{i-1} is the most likely k-topmost lines in X_{i-1}.

We use r to denote the likelihood of A_{i-1} being the k-topmost lines in X_{i-1} and use r' to denote the corresponding likelihood for A'_{i-1}. Then it follows that

$$\begin{aligned} r' &= \frac{r}{p_{l_k}} \cdot \frac{p_\gamma}{1-p_\gamma} \cdot \frac{1}{\prod_{\forall f_j \text{ between } f_\gamma \text{ and } f_{l_k}} (1-p_j)} \\ &\geq \frac{r}{p_{l_k}} \cdot \frac{p_\gamma}{1-p_\gamma}. \end{aligned}$$

Since $p_{l_k} \leq h$ (because h is the supremum) and $p_\gamma > h_0$ (because $p_\gamma \in H_i$), we further have

$$\begin{aligned} r' &\geq \frac{r}{p_{l_k}} \cdot \frac{p_\gamma}{1-p_\gamma} \\ &> \frac{r}{h} \cdot \frac{h_0}{1-h_0} \\ &= r, \end{aligned}$$

which obviously is a contradiction.

Thus, we can assert that $l_k \in L_1$, so that $l_1, l_2, \ldots, l_k$ are all in L_1. Similarly, for A_i, we have the same conclusion, i.e., if we assume that $A_i = (f_{l'_1}, f_{l'_2}, \ldots, f_{l'_k})$, then $l'_1, l'_2, \ldots, l'_k$ are all in L_1 as well.

Finally, because f_α and f_β are in F_2 and the lines of A_{i-1} and A_i are all in L_1, the exchange of ranks of f_α and f_β has no impact on the k-topmost lines probabilities of A_{i-1} and A_i. Consequently, we can conclude that $A_{i-1} = A_i$, i.e., c_i is invalid, completing the proof (by contraposition).

4.6.3 Proof for Lemma 4.3

In the proof of this lemma, for convenience, we rearrange the order of the m intersections by the depths instead of x-coordinates. In other words, we assume that $d_1 \leq d_2 \leq \cdots \leq d_m$ (instead of the previous assumption $x_1 < x_2 < \cdots < x_m$). Consider an m-element

sequence

$$\underbrace{0,0,\ldots,0}_{(n-1)\times 0\text{'s}},\underbrace{1,1,\ldots,1}_{(n-2)\times 1\text{'s}},\underbrace{2,2,\ldots,2}_{(n-3)\times 2\text{'s}},\ldots,\underbrace{i,i,\ldots,i}_{(n-i-1)\times i\text{'s}},\ldots,\underbrace{n-3,n-3}_{2\times(n-3)\text{'s}},\underbrace{n-2.}_{1\times(n-2)}$$

We use d'_i to denote the i-th element of the sequence above. By Lemma 4.1, it is easy to verify that $d'_i \leq d_i$ for $i = 1, 2, \ldots, m$. Furthermore, since $u(d)$ is a non-increasing function (proof is omitted here), we assert $u(d'_i) \geq u(d_i)$. Consequently, we have

$$\begin{aligned}
\sum_{i=1}^{m} u(d_i) &\leq \sum_{i=1}^{m} u(d'_i) \\
&= \sum_{d=0}^{n-2} u(d)\cdot(n-d-1) \\
&\leq \left(\sum_{d=0}^{n-2} u(d)\right)\cdot n.
\end{aligned}$$

4.6.4 Proof for Equation 4.3

$$\begin{aligned}
\sum_{d=k}^{\infty}\binom{d}{k}\lambda^d &= \binom{k}{k}\lambda^k + \binom{k+1}{k}\lambda^{k+1} + \binom{k+2}{k}\lambda^{k+2} + \ldots \\
&= \binom{k-1}{k-1}\left(\lambda^k + \lambda^{k+1} + \lambda^{k+2} + \lambda^{k+3} + \ldots\right) + \\
&\quad \binom{k}{k-1}\left(\lambda^{k+1} + \lambda^{k+2} + \lambda^{k+3} + \lambda^{k+4} + \ldots\right) + \\
&\quad \binom{k+1}{k-1}\left(\lambda^{k+2} + \lambda^{k+3} + \lambda^{k+4} + \lambda^{k+5} + \ldots\right) + \\
&\quad \binom{k+2}{k-1}\left(\lambda^{k+3} + \lambda^{k+4} + \lambda^{k+5} + \lambda^{k+6} + \ldots\right) + \ldots \\
&= \binom{k-1}{k-1}\cdot\frac{\lambda^k}{1-\lambda} + \binom{k}{k-1}\cdot\frac{\lambda^{k+1}}{1-\lambda} + \binom{k+1}{k-1}\cdot\frac{\lambda^{k+2}}{1-\lambda} + \ldots \\
&= \frac{\lambda}{1-\lambda}\cdot\sum_{d=k-1}^{\infty}\binom{d}{k-1}\lambda^d.
\end{aligned}$$

If we recursively apply the above equation k times, we have

$$\begin{aligned}
\sum_{d=k}^{\infty}\binom{d}{k}\lambda^d &= \frac{\lambda}{1-\lambda}\cdot\sum_{d=k-1}^{\infty}\binom{d}{k-1}\lambda^d \\
&= \frac{\lambda^2}{(1-\lambda)^2}\cdot\sum_{d=k-2}^{\infty}\binom{d}{k-2}\lambda^d \\
&= \frac{\lambda^3}{(1-\lambda)^3}\cdot\sum_{d=k-3}^{\infty}\binom{d}{k-3}\lambda^d \\
&\cdots \\
&= \frac{\lambda^k}{(1-\lambda)^k}\cdot\sum_{d=0}^{\infty}\binom{d}{0}\lambda^d \\
&= \frac{\lambda^k}{(1-\lambda)^k}\cdot\left(1+\lambda+\lambda^2+\lambda^3+\dots\right) \\
&= \frac{\lambda^k}{(1-\lambda)^k}\cdot\frac{1}{1-\lambda} = \frac{\lambda^k}{(1-\lambda)^{k+1}}.
\end{aligned}$$

4.6.5 Proof for Lemma 4.4

In the following proof, for simplicity, we use $1, 2, \dots, n$ to represent $x_1, x_2, \dots, x_n$. The first n-element sequence is $1, 2, 3, \dots, n$, and the likelihood for element w to become the winner is

$$L(w) = p_w \cdot \prod_{i=1}^{w-1}(1-p_i), \text{where } p_i = \frac{p_{i+1}}{1+p_{i+1}} - \epsilon \text{ and } p_n = 1.$$

If we ignore the term $-\epsilon$, i.e., $p_i = p_{i+1}/(1+p_{i+1})$, then it is easy to verify that $L(1) = L(2) = \cdots = L(n) = p_1$. With the additional term $-\epsilon$, the above equation becomes $L(1) < L(2) < \cdots < L(n)$, which implies that point x_n is the winner of the first sequence.

For the rest of the sequences, we consider the following general format

$$i+1, i+2, \dots, j, i, j+1, j+2, \dots, n, i-1, i-2, \dots, 2, 1, \tag{4.5}$$

where i is the moving element, and $i < j \leq n$. We will prove that the winner of this sequence is j.

Consider the following $(n-1)$-element sequence generated by removing i from Sequence 4.5.

$$i+1, i+2, \ldots, j, j+1, j+2, \ldots, n, i-1, i-2, \ldots, 2, 1, \tag{4.6}$$

By a similar argument as above, we have $L(i+1) < L(i+2) < \cdots < L(n)$. Moreover, since $p_n > p_{i-1} > p_{i-2} > \cdots > p_2 > p_1$, we also have $L(n) > L(i-1) > L(i-2) > \cdots > L(2) > L(1)$. Therefore, point n has the largest likelihood in Sequence 4.6.

Now, let us insert i back into the sequence to get back Sequence 4.5. Clearly, $L(i+1), L(i+2), \ldots, L(j)$ will not change, and $L(i) < L(j)$ because i is to the right of j and $p_i < p_j$. In addition, we argue that $L(j+1), L(j+2), \ldots, L(n), L(i-1), L(i-2), \ldots, L(1)$ will all drop significantly due to the impact of inserting i so that even the previous largest likelihood $L(n)$ will be less than $L(j)$. Consequently, $L(j)$ is the largest likelihood of Sequence 4.5, and x_j is the winner.

To see why, although intuitively $L(i+1) < L(i+2) < \cdots < L(n)$ in Sequence 4.6, they are very close to each other because ϵ is sufficiently small. By inserting i back into the sequence, all of $L(j+1), \ldots, L(n)$ now need to be multiplied by an additional factor $(1-p_i)$ that can be small enough (by judiciously choosing a proper ϵ) to ensure that $L(n)$ is even smaller than $L(i+1)$. We give a formal proof below.

If we define $p'_n = 1$, and $p'_i = p'_{i+1}/(1+p'_{i+1})$, then we have the following two facts that can be easily proved by induction.

1. $p'_{n-i} = 1/(i+1)$,
2. $0 < p'_{n-i} - p_{n-i} \leq i\epsilon$.

By choosing any $\epsilon < n^{-4}$, we have

$$\begin{aligned}
L(n) &= p_n \cdot \prod_{j=i}^{n-1} (1 - p_j) \\
&= \prod_{j=1}^{n-i} (1 - p_{n-j}) \quad (\text{since } p_n = 1) \\
&\leq \prod_{j=1}^{n-i} (1 - p'_{n-j} + j\epsilon) \\
&= \prod_{j=1}^{n-i} (1 - 1/(j+1) + j\epsilon) \\
&\leq \prod_{j=1}^{n-i} (1 - 1/(j+1)) + \sum_{j=1}^{n-i} j\epsilon \quad (\text{since } 1 - p'_{n-j} + j\epsilon \leq 1 \text{ for } j = 1, 2, \ldots, n) \\
&= p'_i + \epsilon(n-i+1)(n-i)/2 \\
&< 1/(n-i+1) + n^2\epsilon.
\end{aligned}$$

Here we used the fact that $\prod_{i=1}^{n} (a_i + b_i) \leq \prod_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$ if $0 \leq a_i + b_i \leq 1$ for $i = 1, 2, \ldots, n$, which can be proved by mathematical induction.

Then,

$$\begin{aligned}
L(n) - L(i+1) &< 1/(n-i+1) + n^2\epsilon - 1/(n-i) \\
&< n^2\epsilon - 1/[(n-i)(n-i+1)] \\
&< n^2\epsilon - 1/n^2 \\
&< 0 \quad (\text{since } \epsilon < n^{-4}).
\end{aligned}$$

Thus, with respect to Sequence 4.5, we can conclude that $L(i+1) > L(k)$ for any $k \in \{j+1, j+2, \ldots n, i-1, i-2, \ldots, 2, 1\}$. Moreover, we know $L(j) > L(j-1) > \cdots > L(i+1)$, and it is also easy to observe that $L(j) > L(i)$. So we finally conclude $L(j)$ is the largest likelihood, i.e., x_j is the winner.

Chapter 5

Stochastic closest pair problem and most likely nearest neighbor search in tree space

In this chapter, we further generalize the most likely nearest neighbor search problem in $\mathbb{R}^1$ that we have solved previously and study two new problems, namely, the stochastic closest pair (SCP) problem and k most likely nearest neighbor (k-LNN) search in so-called tree space. For the former, we propose the first algorithm for computing the ℓ-threshold probability and the expectation of the closest pair distance for a realization of the input stochastic points. For the latter, we study the k most likely Voronoi Diagram (k-LVD), where we show the combinatorial complexity of k-LVD is $O(nk)$ under two reasonable assumptions, leading to a logarithmic query time for k-LNN.

5.1 Preliminaries

A *tree space* $\mathcal{T}$ is represented by a t-vertex positively-edge-weighted tree T where the weight of each edge depicts its "length". Formally, $\mathcal{T}$ is the geometric realization of T, in which each edge weighted by w is isometric to the interval $[0, w]$. There is a natural metric over $\mathcal{T}$ which defines the distance $dist(x, y)$ as the length of the (unique) simple path between x and y in $\mathcal{T}$. See Figure 5.1 for an example of tree space.

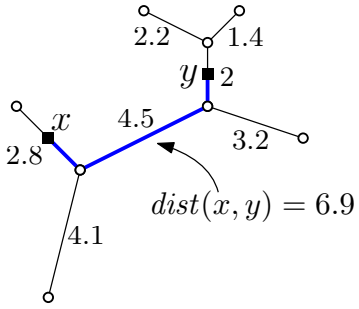


Figure 5.1: A tree space and the unique simple path (in blue) between x and y. Since x and y are the midpoints of the edges they lie on, the length of the path is $0.5 \cdot 2.8 + 4.5 + 0.5 \cdot 2 = 6.9$.

Similar to the model in Chapter 4, we study the problems under existential uncertainty: each input stochastic point has a fixed location (in $\mathcal{T}$) associated with an (independent) existence probability. Let S be the given set of n stochastic points in $\mathcal{T}$ each of which is associated with an existence probability. A *realization* of S refers to a random sample of S in which each point is sampled with its existence probability.

5.2 The stochastic closest pair problem

Let $\mathcal{T}$ be a tree space represented by a t-vertex edge-weighted tree T and let $S = \{a_1, \ldots, a_n\} \subset \mathcal{T}$ be a set of stochastic points where a_i has an existence probability π_{a_i}. We use $\kappa(S)$ to denote the random variable indicating the closest pair distance of a realization of S (if the realization is of size less than 2, we simply set its closest pair distance to be 0).

5.2.1 Computing the threshold probability

We study the problem of computing the probability that $\kappa(S)$ is at least ℓ for a given threshold ℓ. We call this quantity the *ℓ-threshold probability* or simply *threshold probability* of $\kappa(S)$, and denote it by $C_{\geq \ell}(S)$. We show that $C_{\geq \ell}(S)$ can be computed in $O(t + n \log n + \min\{tn, n^2\})$ time. This result gives us an $O(t + n^2)$ upper bound for $t = \Omega(n)$ and an $O(n \log n + tn)$ bound for $t = O(n)$. In the rest of this section, we

first present an $O(t + n^3)$-time algorithm for computing $C_{\geq \ell}(S)$, and then show how to improve it to achieve the desired bound. For simplicity of exposition, we assume $a_1, \ldots, a_n$ have distinct locations in $\mathcal{T}$.

An $O(t + n^3)$-time algorithm

In order to conveniently and efficiently handle the stochastic points in a tree space, we begin with a preprocessing step, which reduces the problem to a more regular setting.

Theorem 5.1. *Given $\mathcal{T}$ and S, one can compute in $O(t+n \log n)$ time a new tree space $\mathcal{T}' \subseteq \mathcal{T}$ represented by an $O(n)$-vertex weighted tree T' s.t. $S \subset \mathcal{T}'$ and every point in S is located at some vertex of T'.* (See Section 5.4.1 for a proof.)

By the above theorem, we use $O(t+n \log n)$ time to compute such a new tree space. Using this tree space as well as the $O(n)$-vertex tree representing it, the problem becomes more regular: every stochastic point in S is located at a vertex. We can further put the stochastic points in one-to-one correspondence with the vertices by adding dummy points with existence probability 0 at the "empty" vertices (i.e., vertices not coinciding with points of S; see Section 5.4.1). In such a regular setting, we then consider how to compute the ℓ-threshold probability. For convenience, we still use T to denote the representation of the (new) tree space and $S = \{a_1, \ldots, a_n\}$ the stochastic dataset (though the actual size of S may be larger than n due to the additional dummy points, it is still bounded by $O(n)$). Since the vertices of T are now in one-to-one correspondence with the points in S, we also use a_i to denote the corresponding vertex of T.

As we are working on a tree space, a natural idea for solving the problem is to exploit the recursive structure of the tree and to compute $C_{\geq \ell}(S)$ in a recursive fashion. To this end, we need to define an important concept called *witness*. We make T rooted by setting a_1 as its root. The subtree rooted at a vertex x is denoted by T_x. Also, we use $V(T_x)$ to denote the set of the stochastic points lying in T_x, or, equivalently, the set of the vertices of T_x. The notations $\bar{p}(x)$ and $ch(x)$ are used to denote the parent of x and the set of the children of x, respectively (for convenience we set $\bar{p}(a_1) = a_1$).

Definition 5.1. *Let $dep(a_i)$ be the depth of a_i in T, i.e., $dep(a_i) = dist(a_1, a_i)$. For any a_i and a_j, we define $a_i \prec a_j$ if $dep(a_i) < dep(a_j)$, or $dep(a_i) = dep(a_j)$ and $i < j$. Clearly, the relation $\prec$ is a strict total order over S (also, over the vertices of T). For*

any subset $S' \subseteq S$ and any vertex a_i of T, we define the ***witness*** *of a_i with respect to S', denoted by $\omega(a_i, S')$, as the smallest vertex in $V(T_{a_i}) \cap S'$ under the $\prec$-order. If $V(T_{a_i}) \cap S' = \emptyset$, we say $\omega(a_i, S')$ is not defined.*

See Figure 5.2 for an illustration of witness. We say a subset $S' \subseteq S$ is *legal* if the closest pair distance of S' is at least ℓ.

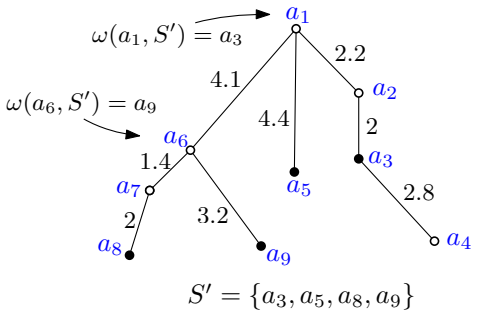


Figure 5.2: An illustration of witness

The following lemma allows us to verify the legality of a subset by using the witnesses, which will be used later.

Lemma 5.1. *For any $S' \subseteq S$, we have S' is legal if and only if every point $a_i \in S \backslash \{a_1\}$ satisfies one of the following three conditions:*
(1) *$\omega(a_i, S')$ is not defined;*
(2) *$\omega(a_i, S') = \omega(\bar{p}(a_i), S')$;*
(3) *$dist(\omega(a_i, S'), \omega(\bar{p}(a_i), S')) \geq \ell$.*
We say that S' is ***locally legal*** *at a_i whenever a_i satisfies one of the above conditions.* (See Section 5.4.2 for a proof.)

In order to compute $C_{\geq \ell}(S)$, we define, for all $x \in S$ and $y \in V(T_{\bar{p}(x)})$,

$$P_y(x) = \begin{cases} \Pr\limits_{S' \subseteq_{\mathrm{R}} V(T_x)} [S' \text{ is legal} \wedge \omega(x, S') = y], & \text{if } y \in V(T_x), \\ \Pr\limits_{S' \subseteq_{\mathrm{R}} V(T_x)} [S'' \text{ is legal} \wedge \omega(\bar{p}(x), S'') = y], & \text{if } y \in V(T_{\bar{p}(x)}) \backslash V(T_x). \end{cases},$$

where $S'' = S' \cup \{y\}$. Here the notation $\subseteq_{\mathrm{R}}$ means that the former is a realization of the latter, i.e., a random sample obtained by sampling each point with its existence

probability. With the above, we immediately have that $C_{\geq \ell}(S) = \sum_{i=1}^{n} P_{a_i}(a_1) - P_0$, where P_0 is the probability that a realization of S contains exactly one point. We then show how $P_y(x)$ can be computed in a recursive way.

Lemma 5.2. *For $x \in S$ and $y \in V(T_x)$, we have that*

$$P_y(x) = Q \cdot \prod_{c \in ch(x)} P_y(c),$$

where $Q = \pi_x$ if $x = y$ and $Q = 1 - \pi_x$ if $x \neq y$. (See Section 5.4.3 for a proof.)

Lemma 5.3. *For $x \in S$ and $y \in V(T_{\bar{p}(x)}) \backslash V(T_x)$, we have that*

$$P_y(x) = \prod_{a_i \in V(T_x)} (1 - \pi_{a_i}) + \sum_{z \in \Gamma} P_z(x),$$

where $\Gamma = \{z \in V(T_x) : y \prec z \text{ and } dist(z, y) \geq \ell\}$. (See Section 5.4.4 for a proof.)

By the above two lemmas, the values of all $P_y(x)$ can be computed as follows. We enumerate $x \in S$ from the greatest to the smallest under $\prec$-order. For each x, we first compute all $P_y(x)$ for $y \in V(T_x)$ by applying Lemma 5.2. After this, we are able to compute all $P_y(x)$ for $y \in V(T_{\bar{p}(x)}) \backslash V(T_x)$ by applying Lemma 5.3. The entire process takes $O(n^3)$ time. Once we have the values of all $P_y(x)$, $C_{\geq \ell}(S)$ can be computed straightforwardly. Including the time for preprocessing, this gives us an $O(t + n^3)$-time algorithm for computing $C_{\geq \ell}(S)$.

Improving the runtime

We first show how to improve the runtime of the above algorithm to $O(t + n^2)$. Note that computing all $P_y(x)$ for $x \in S$ and $y \in V(T_x)$ takes only $O(n^2)$ time in total, as we can charge the time for computing $P_y(x)$ to the pairs (y, c) for $c \in ch(x)$ and thus each pair of vertices is charged at most a constant amount of time. So the bottleneck is the computation of $P_y(x)$ for $y \in V(T_{\bar{p}(x)}) \backslash V(T_x)$. For a specific $x \in S$, we want to compute all $P_y(x)$ for $y \in V(T_{\bar{p}(x)}) \backslash V(T_x)$ in linear time. To achieve this, we review the formula given in Lemma 5.3. Assume that $V(T_x) = \{z_1, \ldots, z_m\}$ where $z_1 \prec \cdots \prec z_m$, and $V(T_{\bar{p}(x)}) \backslash V(T_x) = \{y_1, \ldots, y_r\}$ where $y_1 \prec \cdots \prec y_r$. Define

$$\Gamma_{y_i} = \{z \in V(T_x) : y_i \prec z \text{ and } dist(z, y_i) \geq \ell\}$$

for $i \in \{1, \ldots, r\}$. Then $P_{y_i}(x)$ is just the sum of $\prod_{j=1}^{m}(1-\pi_{z_j})$ and all $P_z(x)$ for $z \in \Gamma_{y_i}$.

Lemma 5.4. *Each set Γ_{y_i} is a suffix of the sequence $(z_1, \ldots, z_m)$, namely, $\Gamma_{y_i} = \{z_j, z_{j+1}, \ldots, z_m\}$ for some $j \in \{1, \ldots, m\}$. Furthermore, we have that $\Gamma_{y_1} \subseteq \cdots \subseteq \Gamma_{y_k} \supseteq \cdots \supseteq \Gamma_{y_r}$ for some $k \in \{1, \ldots, t\}$.* (See Section 5.4.5 for a proof.)

The above observation gives us the idea to efficiently compute $P_{y_1}(x), \ldots, P_{y_r}(x)$. Instead of computing $P_{y_i}(x)$ straightforwardly using the formula given in Lemma 5.3, we compute each $P_{y_i}(x)$ by modifying $P_{y_{i-1}}(x)$. Specifically, we first compute $P_{y_1}(x)$ straightforwardly and then begin to compute $P_{y_2}(x), \ldots, P_{y_r}(x)$ in order. If $\Gamma_{y_i} \subseteq \Gamma_{y_{i-1}}$, we compute $P_{y_i}(x)$ by subtracting all $P_z(x)$ for $z \in \Gamma_{y_{i-1}} \backslash \Gamma_{y_i}$ from $P_{y_{i-1}}(x)$. Otherwise, if $\Gamma_{y_i} \supseteq \Gamma_{y_{i-1}}$, we compute $P_{y_i}(x)$ by adding all $P_z(x)$ for $z \in \Gamma_{y_i} \backslash \Gamma_{y_{i-1}}$ to $P_{y_{i-1}}(x)$. According to Lemma 5.4, in the entire process, each $P_z(x)$ for $z \in \{z_1, \ldots, z_m\}$ is at most added and subtracted once. Therefore, with the sequence $(z_1, \ldots, z_m)$ in hand, it is easy to compute $P_{y_1}(x), \ldots, P_{y_r}(x)$ in $O(n)$ time. Note that the sequence $(z_1, \ldots, z_m)$ can be easily obtained in $O(n)$ time, if we sort all the points $a_1, \ldots, a_n$ in $\prec$-order at the beginning of the algorithm. This improves the overall time complexity to $O(t+n^2)$.

Indeed, we can further improve the runtime to $O(t+n\log n+\min\{tn, n^2\})$. In other words, we show that $C_{\geq \ell}(S)$ can be computed in $O(n \log n + tn)$ time when $t = O(n)$. To achieve this, we recall the original tree space (before the preprocessing) which is represented by a t-vertex tree. Intuitively, if t is significantly smaller than n, then most stochastic points in S are located inside the interiors of the edges of the original tree. In this case, after the preprocessing, we will have a lot of "chain" structures in the new tree T. This gives us the insight to further improve our algorithm.

Definition 5.2. *A **chain** of T is a sequence of vertices $(b_1, \ldots, b_k)$ satisfying*
(1) *b_i is the only child of b_{i-1} for $i \in \{2, \ldots, k\}$;*
(2) *b_k has at most one child;*
(3) *b_1 is either the root or the only child of $\bar{p}(b_1)$.*
(See Figure 5.3 for an example of chain.) *A chain is **maximal** if it is not properly contained in another chain. A vertex of T is called **chain vertex** if it is contained in some chain. Otherwise, it is called **non-chain vertex**.*

Lemma 5.5. *If $\mathcal{T}$ is a tree space represented by a t-vertex tree and $\mathcal{T}' \subseteq \mathcal{T}$ is also a*

tree space represented by a rooted tree T, then the number of the non-chain vertices of T is $O(t)$. (See Section 5.4.6 for a proof.)

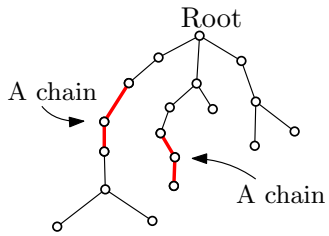


Figure 5.3: An example of chains.

One can easily verify that when removing all the non-chain vertices (and their adjacent edges) from T, each connected component of the remaining forest corresponds to a maximal chain of T. Thus, the number of the maximal chains of T is also bounded by $O(t)$.

Now we explain why the chains of T are helpful for us. Let $(b_1, \ldots, b_k)$ be a chain of T. For convenience of exposition, we assume b_k has a (unique) child b_{k+1} and b_1 has the parent b_0. Our previous algorithm takes $O(kn)$ time to compute all $P_y(x)$ for $x \in \{b_1, \ldots, b_k\}$ and $y \in V(\bar{p}(x))$. To improve the runtime, we want that these values can be computed in $O(n)$ time. This seems impossible as the number of the values to be computed is $\Theta(kn)$ in worst case. However, instead of computing these values explicitly, we can compute them implicitly. Note that $P_y(b_i)$ is defined only when $y \in \{b_{i-1}, \ldots, b_k\} \cup V(T_{b_{k+1}})$. Set $\sigma_0 = 1$ and $\sigma_i = \prod_{j=1}^{i}(1 - \pi_{b_j})$ for $i \in \{1, \ldots, k\}$. Let $b_i \in \{b_1, \ldots, b_k\}$ be a vertex in the chain. By Lemma 5.2, we observe the following. First, for any $y \in V(T_{b_{k+1}})$, we have that $P_y(b_i) = P_y(b_{k+1}) \cdot \sigma_k/\sigma_{i-1}$. Furthermore, we have that $P_{b_i}(b_i) = \pi_{b_i} \cdot P_{b_i}(b_{i+1})$ and

$$P_{b_j}(b_i) = P_{b_j}(b_j) \cdot \frac{\sigma_{j-1}}{\sigma_{i-1}} = P_{b_j}(b_{j+1}) \cdot \frac{\pi_{b_j}\sigma_{j-1}}{\sigma_{i-1}}$$

for $j \in \{i+1, \ldots, k\}$. Thus, as long as we know $\sigma_1, \ldots, \sigma_k$ and $P_{b_0}(b_1), \ldots, P_{b_{k-1}}(b_k)$, any $P_y(x)$ with $x \in \{b_1, \ldots, b_k\}$ can be computed in constant time (note that the values of $P_y(b_{k+1})$ are already in hand when we deal with the chain). In other words, to implicitly compute all $P_y(x)$ for $x \in \{b_1, \ldots, b_k\}$, it suffices to compute $\sigma_1, \ldots, \sigma_k$ and $P_{b_0}(b_1), \ldots, P_{b_{k-1}}(b_k)$, and associate to each b_i the values of σ_i and $P_{b_{i-1}}(b_i)$. Clearly, one

can easily compute $\sigma_1, \ldots, \sigma_k$ in $O(k)$ time. We then show that $P_{b_0}(b_1), \ldots, P_{b_{k-1}}(b_k)$ can be computed in $O(n)$ time. Define $A_i = \{z \in V(T_{b_i}) : dist(z, b_{i-1}) \geq \ell\}$, then $A_k \subseteq A_{k-1} \subseteq \cdots \subseteq A_1$ and each A_i is a suffix of the $\prec$-order sorted sequence of the vertices in $V(T_{b_0})$. Now by Lemma 5.3, one can deduce that

$$P_{b_{i-1}}(b_i) = (1 - \pi_{b_i}) \cdot P_{b_i}(b_{i+1}) + \sum_{z \in A_i \setminus A_{i+1}} Q_z \cdot P_z(b_{i+1}),$$

where $Q_z = \pi_{b_i}$ if $z = b_i$ and $Q_z = 1 - \pi_{b_i}$ otherwise. Thus, if the computation is taken in the order $P_{b_{k-1}}(b_k), \ldots, P_{b_0}(b_1)$, then each $P_{b_{i-1}}(b_i)$ can be easily computed in $O(|A_i \setminus A_{i+1}|)$ time. In this way, we use $O(n)$ time to implicitly compute all $P_y(x)$ for $x \in \{b_1, \ldots, b_k\}$. It turns out that the computation task for any chain can be done in $O(n)$ time.

With this in hand, it is not difficult to compute all $P_y(x)$ in $O(tn)$ time. We enumerate $x \in S$ from the greatest to the smallest under $\prec$-order. For each x visited, if x is a non-chain vertex, we use $O(n)$ time to explicitly compute all $P_y(x)$ in the previous way. If x is the deepest vertex of a chain, i.e., x has no child or its child is a non-chain vertex, then we find the maximal chain containing x and implicitly complete the computation task for this chain in $O(n)$ time. Otherwise, if x is a chain vertex but not the deepest one, we just skip it as all $P_y(x)$ have been implicitly computed previously. The entire process takes $O(tn)$ time, as there are $O(t)$ non-chain vertices and $O(t)$ maximal chains. Including the time for preprocessing and sorting $a_1, \ldots, a_n$, we solve the problem in $O(n \log n + tn)$ time. Combining with the case $t = \Omega(n)$, we finally conclude the following.

Theorem 5.2. *Given an edge-weighted tree T with t vertices and a set S of n stochastic points in its tree space $\mathcal{T}$, one can compute the ℓ-threshold probability of the closest pair distance of S, $C_{\geq \ell}(S)$, in $O(t + n \log n + \min\{tn, n^2\})$ time.*

5.2.2 Computing the expected closest pair distance

Based on our algorithm for computing the threshold probability, we further study the problem of computing the expected closest pair distance of S, i.e., the expectation of $\kappa(S)$. It is easy to see that our algorithm in Section 5.2.1 immediately gives us an $O(t + \min\{tn^3, n^4\})$-time algorithm to compute $\mathbf{E}[\kappa(S)]$. This is because the random

variable $\kappa(S)$ has at most $\binom{n}{2}$ distinct possible values and hence we can compute $\mathbf{E}[\kappa(S)]$ via $O(n^2)$ threshold probability "queries" with various thresholds ℓ (note that after preprocessing our algorithm answers each threshold probability query in $O(\min\{tn, n^2\})$ time).

If we want to compute the exact value of $\mathbf{E}[\kappa(S)]$ (via threshold probability queries), $\Theta(n^2)$ queries are necessary in worst case. So it is natural to ask whether we can use fewer queries to approximate $\mathbf{E}[\kappa(S)]$. In the rest of this section, we show that one can use $O(\varepsilon^{-1}n)$ threshold probability queries to achieve a $(1+\varepsilon)$-approximation for $\mathbf{E}[\kappa(S)]$, which in turn gives us an $O(t+\varepsilon^{-1}\min\{tn^2, n^3\})$-time approximation algorithm for computing $\mathbf{E}[\kappa(S)]$.

For simplicity of exposition, we assume that the stochastic points in S are now in one-to-one correspondence with the vertices of T (this is what we have after preprocessing). We begin with a simple case, in which the *spread* of T, i.e., the ratio of the length of the longest edge to the length of the shortest edge is bounded by some polynomial of n. In this case, to approximate $\mathbf{E}[\kappa(S)]$ is fairly easy, and we only need $O(\varepsilon^{-1}\log n)$ threshold probability queries.

Definition 5.3. *For $\beta > \alpha > 0$ and $\tau > 1$, the (α, β, τ)-**jump** is defined as*

$$J = \{\alpha, \tau\alpha, \tau^2\alpha, \ldots, \tau^k\alpha, \beta\},$$

where $\tau^k\alpha < \beta$ and $\tau^{k+1}\alpha \geq \beta$.

Let $d_{\min}$ be the length of the shortest edge of T and $d_{\max}$ be the sum of the lengths of all edges of T. Also, let J be the $(d_{\min}, d_{\max}, 1+\varepsilon)$-jump. Suppose $J = \{\ell_1, \ldots, \ell_{|J|}\}$. Then we do $|J|$ threshold probability queries using the thresholds $\ell_1, \ldots, \ell_{|J|}$, and compute

$$E = \sum_{i=1}^{|J|} C_{\geq \ell_i}(S) \cdot (\ell_i - \ell_{i-1})$$

as an approximation of $\mathbf{E}[\kappa(S)]$ (where $\ell_0 = 0$). Note that $|J| = O(\log_{1+\varepsilon} \frac{d_{\max}}{d_{\min}}) = O(\log_{1+\varepsilon} n) = O(\varepsilon^{-1}\log n)$. It is easy to verify that $E \leq \mathbf{E}[\kappa(S)] \leq (1+\varepsilon)E$.

The problem becomes interesting when the spread of T is unbounded. In this case, although the above method still correctly approximates $\mathbf{E}[\kappa(S)]$, the number of the

threshold probability queries is no longer well-bounded. Imagine that the $O(n^2)$ possible values of $\kappa(S)$ are distributed as ℓ, $(1+\varepsilon)\ell$, $(1+\varepsilon)^2\ell$, etc. Then the $(d_{\min}, d_{\max}, 1+\varepsilon)$-jump J is of size $\Omega(n^2)$. Moreover, for guaranteeing the correctness, it seems that we cannot "skip" any element in J. However, as one will realize later, such an extreme situation can never happen. Recall that we are working on a weighted tree and the $O(n^2)$ possible values of $\kappa(S)$ are indeed the pairwise distances of the vertices of the tree. As such, these values are not arbitrary, and our insight here is to exploit the underlying properties of the distribution of these values.

Let $e_1, \ldots, e_{n-1}$ be the edges of T where e_i has the length (weight) w_i. Assume $w_1 \leq \cdots \leq w_{n-1}$. We define an index set $I = \left\{m : \sum_{i=1}^{m-1} w_i < w_m\right\}$. Suppose $I = \{m_1, \ldots, m_k\}$ where $m_1 < \cdots < m_k$. Note that $m_1 = 1$. For convenience, we set $m_{k+1} = n$. We design our threshold probability queries as follows. Let J_i be the $(w_{m_i}, s_i, 1+\varepsilon)$-jump where $s_i = \sum_{j<m_{i+1}} w_j$, and $J = J_1 \cup \cdots \cup J_k$. Suppose $J = \{\ell_1, \ldots, \ell_{|J|}\}$ and set $\ell_0 = 0$. Similarly to the previous case, we do $|J|$ threshold probability queries using the thresholds $\ell_1, \ldots, \ell_{|J|}$, and compute

$$E = \sum_{i=1}^{|J|} C_{\geq \ell_i}(S) \cdot (\ell_i - \ell_{i-1})$$

as an approximation of $\mathbf{E}[\kappa(S)]$. We first verify the correctness, i.e., $E \leq \mathbf{E}[\kappa(S)] \leq (1+\varepsilon)E$. The fact $E \leq \mathbf{E}[\kappa(S)]$ can be easily verified. To see the inequality $\mathbf{E}[\kappa(S)] \leq (1+\varepsilon)E$, we define a piecewise-constant function $h : \mathbb{R}^+ \cup \{0\} \to [0,1]$ as

$$h(\ell) = \begin{cases} C_{\geq \ell_i}(S) & \text{if } (1+\varepsilon)\ell_i < \ell \leq (1+\varepsilon)\ell_{i+1}, \\ 0 & \text{if } \ell > (1+\varepsilon)l_{|J|}, \\ 1 & \text{if } \ell = 0. \end{cases}$$

Then it is clear that $(1+\varepsilon)E = \int_0^\infty h(\ell)d\ell$. We claim that $\int_0^\infty h(\ell)d\ell \geq \int_0^\infty C_{\geq \ell}(S)d\ell$, hence we have $\mathbf{E}[\kappa(S)] \leq (1+\varepsilon)E$. Note that the jumps $J_1, \ldots, J_k$ are disjoint and each of them contains a consecutive portion of the sequence $\ell_1, \ldots, \ell_{|J|}$. Furthermore, if ℓ_i and ℓ_{i+1} belong to different jumps, then there is no possible value of $\kappa(S)$ within the range (ℓ_i, ℓ_{i+1}), i.e., $C_{\geq \ell}(S)$ is constant when $\ell \in [\ell_i, \ell_{i+1})$. With this observation, it is not difficult to verify that $h(\ell) \geq C_{\geq \ell}(S)$ for any $\ell \geq 0$. Consequently, we have $\mathbf{E}[\kappa(S)] \leq (1+\varepsilon)E$, which implies the correctness of our method. Now the only thing

remaining is to bound the number of the threshold probability queries, which we show in Lemma 5.6.

Lemma 5.6. *For each jump J_i, we have $|J_i| = O(\varepsilon^{-1}(m_{i+1} - m_i))$. As a result, the total number of the threshold probability queries, $|J|$, is $O(\varepsilon^{-1}n)$.* (See Section 5.4.7 for a proof.)

Indeed, the above method can be extended to a much more general case, in which the stochastic dataset S is given in any metric space $\mathcal{X}$ (not necessarily a tree space). In this case, one can still define the threshold probability $C_{\geq\ell}(S)$ as well as the expected closest pair distance $\mathbf{E}[\kappa(S)]$ in the same fashion. Our conclusion is the following.

Theorem 5.3. *Given a set S of n stochastic points in a metric space $\mathcal{X}$, one can $(1+\varepsilon)$-approximate the expected closest pair distance of S, $\mathbf{E}[\kappa(S)]$, via $O(\varepsilon^{-1}n)$ threshold probability queries.* (See Section 5.4.8 for a proof.)

For the expected closest pair distance in tree space, we can eventually conclude the following by plugging in our algorithm in Section 5.2.1 for computing $C_{\geq\ell}(S)$.

Corollary 5.1. *Given a tree space $\mathcal{T}$ represented by a weighted tree T with t vertices and a set S of n stochastic points in $\mathcal{T}$, one can compute a $(1+\varepsilon)$-approximation for the expected closest pair distance of S, $\mathbf{E}[\kappa(S)]$, in $O(t + \varepsilon^{-1}\min\{tn^2, n^3\})$ time.*

5.3 The most likely nearest neighbor search problem

In this section, we study the k most likely nearest neighbor (k-LNN) search in a tree space. Again, let $\mathcal{T}$ be a tree space represented by a t-vertex weighted tree T and $S = \{a_1, \ldots, a_n\} \subset \mathcal{T}$ be the given stochastic dataset where the point a_i has an existence probability π_{a_i}. The k-LNN search problem can be defined as follows. Let $q \in \mathcal{T}$ be any point. For each $a_i \in S$, define $NNP_q(a_i)$ as the probability that the nearest neighbor of q in a realization of S is a_i. Clearly, the nearest neighbor of q in a realization is a_i iff a_i is in the realization and any point closer to q is not in the realization. Therefore, we have

$$NNP_q(a_i) = \pi_{a_i} \cdot \prod_{x \in \Gamma} (1 - \pi_x),$$

where $\Gamma = \{x \in S : dist(q, x) < dist(q, a_i)\}$. Given a query point $q \in \mathcal{T}$, the goal of the k-LNN search is to report the k-LNN of q, which is a k-sequence $(a_{i_1}, \ldots, a_{i_k})$ of points in S such that $NNP_q(a_{i_1}) \geq \cdots \geq NNP_q(a_{i_k}) \geq NNP_q(a_j)$ for all $j \notin \{i_1, \ldots, i_k\}$. For convenience, we assume $NNP_q(a_i) \neq NNP_q(a_j)$ for any $q \in \mathcal{T}$ and $a_i \neq a_j$ so that the k-LNN of any query point $q \in \mathcal{T}$ is uniquely defined.

A standard tool for nearest neighbor search is the Voronoi diagram. In the stochastic setting, we seek the most likely Voronoi diagram (LVD), the concept of which is for the first time introduced in [58]. The k-LVD partitions the query space into connected cells such that points in the same cell have the same k-LNN. Figure 5.4 presents an example of 1-LVD in a tree space.

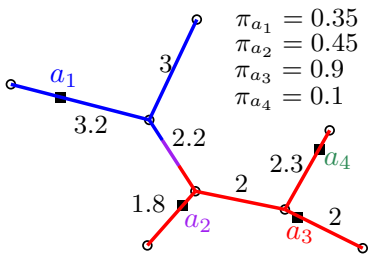


Figure 5.4: A tree-space 1-LVD with 3 cells

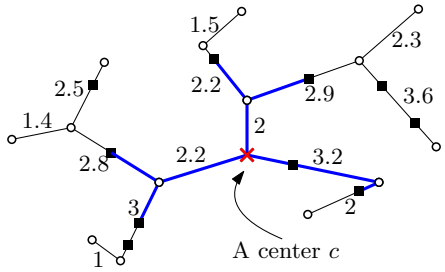


Figure 5.5: A degree-3 center involving 5 points.

5.3.1 The size of the tree-space LVD

We use $\Psi_{\mathcal{T}}^S$ to denote the k-LVD of S on $\mathcal{T}$, i.e., the collection of the cells. Formally, $\Psi_{\mathcal{T}}^S$ can be defined as follows. For any k-sequence $\eta = (a_{i_1}, \ldots, a_{i_k})$, let Ψ_η be the set of the connected components of the subspace $\{q \in \mathcal{T} : \eta \text{ is the } k\text{-LNN of } q\}$. Then $\Psi_{\mathcal{T}}^S$ is the union of Ψ_η over all possible η. Clearly, the size of $\Psi_{\mathcal{T}}^S$ significantly influences the space efficiency of the LVD-based algorithm for k-LNN search. Let $m_{ij} \in \mathcal{T}$ be the "midpoint" of a_i and a_j, i.e., the midpoint of the path between a_i and a_j in $\mathcal{T}$. It is easy to see that the k-LNN only changes nearby these $\binom{n}{2}$ midpoints. However, this does not immediately imply that the size of $\Psi_{\mathcal{T}}^S$ is bounded by $O(n^2)$. The reason is that $O(n^2)$ points do not necessarily decompose $\mathcal{T}$ into $O(n^2)$ pieces (cells), unless these points are located only in the interiors of the edges. The rigorous proof for the $O(n^2)$ upper-bound can be seen later as a direct corollary of Lemma 5.7.

Definition 5.4. *For any two midpoints m_{ij} and $m_{i'j'}$, we define $m_{ij} \equiv m_{i'j'}$ iff m_{ij} and $m_{i'j'}$ have the same location in $\mathcal{T}$ and $dist(a_i, m_{ij}) = dist(a_j, m_{ij}) = dist(a_{i'}, m_{i'j'}) = dist(a_{j'}, m_{i'j'})$. Clearly, $\equiv$ is an equivalence relation over the midpoints. We call the equivalence classes (under $\equiv$) **centers** of S and use $[m_{ij}]$ to denote the center that contains m_{ij}. A stochastic point $a_i \in S$ is said to be **involved** by a center c if $c = [m_{ij}]$ for some j. The **degree** of a center c, denoted by $deg(c)$, is defined as the number of the connected components of $\mathcal{T} \backslash \hat{c}$ that contain at least one point involved by c, where $\hat{c}$ denotes the point in $\mathcal{T}$ corresponding to c, and each such component is called a **branch** of c. A center c is said to be **critical** if $\hat{c}$ is not in the interior of any cell $C \in \Psi_{\mathcal{T}}^S$ and there exists at least one point involved by c that is in the k-LNN of $\hat{c}$.* (See Figure 5.5 for an intuitive illustration of a center.)

Lemma 5.7. *Let Γ be the set of the critical centers and $\xi = \sum_{c \in \Gamma} deg(c)$. Then $|\Psi_{\mathcal{T}}^S| \leq \xi + 1$.* (See Section 5.4.9 for a proof.)

The above lemma immediately gives us the $O(n^2)$ upper bound for the size of $\Psi_{\mathcal{T}}^S$. Indeed, a center c of S contains at least $\Omega(deg(c) \cdot m)$ midpoints, where m is the number of the points involved by c, so $\xi + 1$ is at most $O(n^2)$. Unfortunately, this upper bound is tight, following from the $\Omega(n^2)$ worst-case lower bound for the size of the 1-dim 1-LVD given in Section 4.4.1 (note that the 1-dim LVD is a special case of the tree-space LVD). Surprisingly, we show that, if we make reasonable assumptions for the existence probabilities of the stochastic points or consider the average case, the size of $\Psi_{\mathcal{T}}^S$ is significantly smaller. Our results are:

- If the existence probabilities of all points in S are *constant-far from* 0, i.e., there is a fixed constant $\varepsilon > 0$ such that $\pi_{a_i} \geq \varepsilon$ for all $a_i \in S$, then the size of the k-LVD $\Psi_{\mathcal{T}}^S$ is $O(kn)$. Note that this assumption about the existence probabilities is natural and reasonable. In applications, an extremely small existence probability means the data point is highly unreliable. Such a point can be considered as a noise and removed from the dataset.

- The average-case size of the k-LVD $\Psi_{\mathcal{T}}^S$ is $O(kn)$. For the average-case analysis we assume that the existence probabilities of the points in S are i.i.d. random variables drawn from any fixed distribution (e.g., the uniform distribution among

$[0,1]$). In other words, we consider the expectation of $|\Psi_{\mathcal{T}}^{S}|$ when $\pi_{a_1},\ldots,\pi_{a_n}$ are such random variables. The interesting point is that the $O(kn)$ upper bound is totally independent of the structure of $\mathcal{T}$ and the locations of the stochastic points. The randomness is only applied to the existence probabilities in our average-case analysis.

To prove these bounds requires new ideas. By Lemma 5.7, to bound the size of $\Psi_{\mathcal{T}}^{S}$, it suffices to bound the degree-sum of the critical centers. Intuitively, if a center c is far from the points it involves (compared with other points in S), then c is less likely to be critical, as the c-involved points are less likely to be in the k-LNN of $\hat{c}$. Along with this intuition, we define the following.

Definition 5.5. *For any center c, the **diameter** of c, denoted by $diam(c)$, is defined as the distance from $\hat{c}$ to the c-involved points. Let $A \subset \mathcal{T}$ be a finite set. We define the **depth** of c with respect to A as $dep_A(c) = |\{x \in A : dist(x,c) < diam(c)\}|$, i.e., the number of the points in A which are closer to c than the c-involved points.*

Our idea here is to first bound the "contribution" (degree-sum) of the "shallow" centers, and then further bound the degree-sum of the critical ones. Specifically, we investigate in Lemma 5.8 the degree-sum of the *d-shallow centers* of S, i.e., the centers of depth less than d with respect to S.

Lemma 5.8. *For $1 \le d \le n-1$, the degree-sum of the d-shallow centers of S is at most $8dn$.* (See Section 5.4.10 for a proof.)

Now we are ready to prove the $O(kn)$ bound for $|\Psi_{\mathcal{T}}^{S}|$ under the "constant-far from 0" assumption about the existence probabilities.

Lemma 5.9. *If the existence probabilities of the points in S are constant-far from 0, then a center of S is critical only if it is $O(k)$-shallow.* (See Section 5.4.11 for a proof.)

Theorem 5.4. *If the existence probabilities of the points in S are constant-far from 0, then the size of the k-LVD $\Psi_{\mathcal{T}}^{S}$ is $O(kn)$.*

Proof. Suppose the existence probabilities $\pi_{a_1},\ldots,\pi_{a_n}$ are constant-far from 0. Lemma 5.9 shows that all the critical centers of S are $O(k)$-shallow. By further applying

Lemma 5.8, the degree-sum of the critical centers is $O(kn)$. Finally, by Lemma 5.7, the size of $\Psi_{\mathcal{T}}^{S}$ is $O(kn)$. □

To prove the bound for the average-case size requires more efforts. Let f be a *fixed* probability distribution function whose support is in $(0, 1]$ and μ be the supremum of the support of f. Define two constants $\mu_0 = \mu/(1+\mu)$ and $\lambda = 1 - \int_{-\infty}^{\mu_0} f(x)dx$. Clearly, if X is a random variable drawn from f, then $\lambda = \Pr[X > \mu_0]$. Note that λ is always positive by definition. The following lemma clarifies the meaning of μ_0.

Lemma 5.10. *Suppose $\pi_{a_1}, \ldots, \pi_{a_n}$ are i.i.d. random variables drawn from f. For any center c of S, the event "c is critical" does* ***not*** *happen if there are k (distinct) points $a_{i_1}, \ldots, a_{i_k}$ in S closer to $\hat{c}$ than the c-involved points such that $\pi_{a_{i_1}}, \ldots, \pi_{a_{i_k}}$ are all greater than μ_0.* (See Section 5.4.12 for a proof.)

Theorem 5.5. *The average-case size of $\Psi_{\mathcal{T}}^{S}$ is $O(kn)$, given that the existence probabilities of the points in S are i.i.d. random variables drawn from a* ***fixed*** *distribution.*

Proof. Suppose the existence probabilities $\pi_{a_1}, \ldots, \pi_{a_n}$ are drawn independently from f. Lemma 5.10 implies that, if c is a center of S with $dep_S(c) = d \geq k$, then

$$\Pr[c \text{ is critical}] \leq u_d = \sum_{i=0}^{k-1} \binom{d}{i} \lambda^i (1-\lambda)^{d-i}.$$

Then by applying Lemma 5.7, we have

$$\mathbf{E}[|\Psi_{\mathcal{T}}^{S}|] \leq \sum_{c} \Pr[c \text{ is critical}] \cdot deg(c) \leq \sum_{c \in H_k} deg(c) + \sum_{d=k+1}^{n-1} \sum_{c \in H_d} (u_{d-1} - u_d) deg(c),$$

where H_d is the set of the d-shallow centers of S. Observe that

$$u_{d-1} - u_d = \binom{d-1}{k-1} \lambda^k (1-\lambda)^{d-k}.$$

Based on this and Lemma 5.8, we further have

$$\mathbf{E}[|\Psi_{\mathcal{T}}^{S}|] \leq 8kn + 8n \sum_{d=k+1}^{n-1} \binom{d-1}{k-1} \lambda^k (1-\lambda)^{d-k} d.$$

Note that

$$\sum_{d=k+1}^{n-1} \binom{d-1}{k-1} \lambda^k (1-\lambda)^{d-k} d = k \left(\frac{\lambda}{1-\lambda}\right)^k \sum_{d=k+1}^{n-1} \binom{d}{k} (1-\lambda)^d.$$

By an induction argument on k, it is not difficult to see that

$$\sum_{d=k+1}^{n-1} \binom{d}{k} (1-\lambda)^d < \sum_{d=k}^{\infty} \binom{d}{k} (1-\lambda)^d = \frac{(1-\lambda)^k}{\lambda^{k+1}}.$$

Finally, by combining the inequalities, $\mathbf{E}[|\Psi_{\mathcal{T}}^S|] \leq 8kn + \frac{8kn}{\lambda} = O(kn)$. □

5.3.2 Constructing LVD and answering queries

In this section, we show how to construct the k-LVD $\Psi_{\mathcal{T}}^S$ and use it to answer k-LNN queries. Let $e_1, \ldots, e_{t-1}$ be the edges of T. Assume each edge e_i has a specified "start point" s_i (which is one of its two endpoints) and the query point q is specified via a pair (i, δ), meaning the point on e_i with distance δ to s_i.

We first explain the data structure used for storing the k-LVD $\Psi_{\mathcal{T}}^S$ and answering queries. The LVD data structure is simple. First, it contains $|\Psi_{\mathcal{T}}^S|$ arrays (called *answer arrays*) each of which stores the k-LNN answer of one cell of $\Psi_{\mathcal{T}}^S$. This part takes $O(k|\Psi_{\mathcal{T}}^S|)$ space. In addition to that, we also need to record the structure of $\Psi_{\mathcal{T}}^S$. For each edge e_i of T, we use a sorted list L_i to store the "cell-decomposition" of e_i. Specifically, the intersection of each cell $C \in \Psi_{\mathcal{T}}^S$ and e_i is an "interval" (may be empty). These intervals are stored in L_i in the order they appear on e_i. Note that this part takes $O(t + |\Psi_{\mathcal{T}}^S|)$ space. Indeed, if an edge is decomposed into p pieces (intervals) by $\Psi_{\mathcal{T}}^S$, then it at least entirely contains $(p-2)$ cells of $\Psi_{\mathcal{T}}^S$ (so we can charge these $(p-2)$ pieces to the corresponding cells and the remaining two pieces to the edge). Therefore, the total space of the LVD data structure is $O(t + k|\Psi_{\mathcal{T}}^S|)$. To answer a query $q = (i, \delta)$, we first do a binary search in the list L_i to know which cell q locates in, and then use the answer array corresponding to the cell to output the k-LNN of q directly. The query time is clearly $O(\log |\Psi_{\mathcal{T}}^S| + k)$.

Next, we consider the construction of the LVD data structure. The first step of the construction is to compute all the centers of S and sort the centers in the interior of each edge e in the order they appear on e. We are able to get this done in $O(t + n^2 \log n)$ time (see Section 5.5.1). After the centers are computed and sorted, we begin to construct

the LVD data structure. Choose a vertex v of T. Starting at v, we do a walk in $\mathcal{T}$ along with the edges of T. The walk visits each edge of T exactly twice and finally goes back to v; see Figure 5.6. During the walk, we maintain a (balanced) binary search tree for $NNP_x(a_1), \ldots, NNP_x(a_n)$ w.r.t. the current location x. By exploiting this BST, we can work out the cell-decomposition of each edge e_i (i.e., the sorted list L_i) at the time we first visit e_i in the walk. Specifically, we track the k-LNN when walking along with e_i, which can be obtained by retrieving the k largest elements from the BST. Whenever the k-LNN changes, a new cell of $\Psi_{\mathcal{T}}^S$ is found, so we need to create a new answer array to store the k-LNN information. Also, we need to update the sorted list L_i. In this way, after we go through e_i (for the first time), L_i is correctly computed. At the second visit of e_i, we do nothing but maintain the binary search tree. When we finish the walk and go back to v, the construction of the LVD data structure is done. Clearly, in the process of the walk, we only need to maintain the binary search tree and retrieve the k-LNN when we arrive at (resp., leave from) a center of S from (resp., to) one of its branches. With a careful implementation and analysis (see 5.5.2), we can complete the entire walk and hence the entire LVD structure in $O(t+n^2 \log n+n^2 k)$ time. Combined with the bounds in Section 5.3.1, we then have the following results.

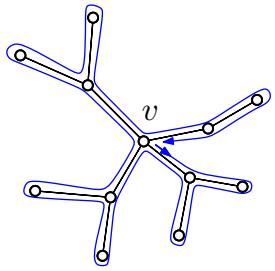


Figure 5.6: A walk in tree visiting each edge exactly twice.

Theorem 5.6. *Given a tree space $\mathcal{T}$ represented by a t-vertex weighted tree and a set S of n stochastic points in $\mathcal{T}$, one can construct in $O(t + n^2 \log n + n^2 k)$ time an LVD data structure to answer k-LNN queries in $O(\log n + k)$ time. The LVD data structure uses worst-case $O(t + kn^2)$ space and average-case $O(t + k^2 n)$ space. Furthermore, if the existence probabilities of the points in S are constant-far from 0, then the LVD data structure uses worst-case $O(t + k^2 n)$ space.*

5.4 Proofs

5.4.1 Proof of Theorem 5.1

Clearly, we can represent $\mathcal{T}$ by a new tree T' with $O(t+n)$ vertices such that each stochastic point in S lies at a vertex of T'. The tree T' is obtained by adding some new vertices to T for the stochastic points lying in the interiors of the edges and "breaking" those edges. It can be easily computed in $O(t + n \log n)$ time by sorting the stochastic points in the interior of each edge (in the order they appear on the edge). Next, we try to simplify T' to make it have $O(n)$ vertices. We say a vertex of T' is *empty* if there is no stochastic point lying at it. The first step is to delete the branches of T' which do not contain any stochastic points. Specifically, if T' has an empty leaf v, we then remove v and its adjacent edge from T'. We keep doing this until T' has no empty leaf. After this step, the underlying tree space of T' changes to be a subspace of the original $\mathcal{T}$. The second step is to compress the "empty chains" in T'. Specifically, if T' has a degree-2 empty vertex v with edges $e_1 = (v, v')$ and $e_2 = (v, v'')$, we replace v, e_1, e_2 with a single edge $e = (v', v'')$ whose weight is the sum of the weights of e_1 and e_2. Note that this operation does not change the underlying tree space. We keep doing this until T' has no degree-2 empty vertex. These two steps of simplification can be done in $O(t+n)$ time. In the resulting T', every empty vertex has a degree at least 3. Thus, T' has $O(n)$ vertices. Furthermore, T' represents a tree space $\mathcal{T}'$ such that $S \subset \mathcal{T}' \subseteq \mathcal{T}$ and each stochastic point in S is located at a vertex of T'.

5.4.2 Proof of Lemma 5.1

The "only if" part is easy to see. Assume that S' is legal. Let $x \in S \backslash \{a_1\}$ be any point. If x does not satisfy the condition (1) and (2), i.e., $\omega(x, S')$ is defined and $\omega(x, S') \neq \omega(\bar{p}(a_i), S')$, then it must satisfy the condition (3) because both $\omega(x, S')$ and $\omega(\bar{p}(a_i), S')$ are in S'. To show the "if" part, assume that S' is not legal. Then we can find distinct points $x, y \in S'$ such that $dist(x, y) < \ell$. Let z be the lowest common ancestor of x and y in T. Without loss of generality, we can assume $x \neq z$. Suppose $\hat{x}$ is the child of z such that $x \in V(T_{\hat{x}})$. We consider two cases, $\omega(z, S') \notin V(T_{\hat{x}})$ and $\omega(z, S') \in V(T_{\hat{x}})$ (note that $\omega(z, S')$ is defined since both x and y are in $V(T_z) \cap S'$). In the case of $\omega(z, S') \notin V(T_{\hat{x}})$, we show that $\hat{x}$ satisfies none of the three conditions. First,

because $x \in V(T_{\hat{x}}) \cap S'$, $\omega(\hat{x}, S')$ is clearly defined so that $\hat{x}$ violates the condition (1). Second, we have $\omega(\hat{x}, S') \neq \omega(\bar{p}(\hat{x}), S')$ since $\omega(\bar{p}(\hat{x}), S') = \omega(z, S') \notin V(T_{\hat{x}})$, which implies that $\hat{x}$ violates the condition (2). Thirdly, since $\omega(z, S') \notin V(T_{\hat{x}})$, we have

$$dist(\omega(\hat{x}, S'), \omega(\bar{p}(\hat{x}), S')) = dist(\omega(\hat{x}, S'), z) + dist(z, \omega(z, S')).$$

Further, by the definition of witness, $dep(\omega(\hat{x}, S')) \leq dep(x)$ and thus $dist(\omega(\hat{x}, S'), z) \leq dist(x, z)$. Similarly, $dep(\omega(z, S')) \leq dep(y)$, so $dist(\bar{p}(\hat{x}), \omega(z, S')) = dist(z, \omega(z, S')) \leq dist(z, y)$. Note that $dist(x, z) + dist(z, y) = dist(x, y) < l$. Therefore, we can conclude that $dist(\omega(\hat{x}, S'), \omega(\bar{p}(\hat{x}), S')) < l$, which implies that $\hat{x}$ violates the condition (3). In the case of $\omega(z, S') \in V(T_{\hat{x}})$, we notice that $y \neq z$; otherwise $\omega(z, S') = z \notin V(T_{\hat{x}})$. Suppose $\hat{y}$ is the child of z such that $y \in V(T_{\hat{y}})$. Then it is easy to see that $\hat{y}$ satisfies none of the three conditions, by applying the same argument used in the previous case (note that the situation here is dual to the previous case).

5.4.3 Proof of Lemma 5.2

By definition, when $y \in V(T_x)$, $P_y(x)$ is the probability that a realization $S' \subseteq_{\mathrm{R}} V(T_x)$ is legal and $\omega(x, S') = y$. If $x = y$, x must be in S' in order to have $\omega(x, S') = y$. Otherwise, if $x \neq y$, x must not be in S'. Thus, the meaning of the factor Q in the formula is clear. Then we consider the vertices in $V(T_x)$ other than x. Clearly, if S' is legal, then $S' \cap V(T_c)$ is also legal for any $c \in \mathrm{ch}(x)$. Also, if $\omega(x, S') = y$, then $w(c, S' \cap V(T_c)) = y$ if $y \in V(T_c)$ and $w(\bar{p}(c), (S' \cap V(T_c)) \cup \{y\}) = y$ if $y \notin V(T_c)$. Therefore, the probabilities of all the legal instances $S' \subseteq V(T_x)$ satisfying $\omega(x, S') = y$ are counted by the right-hand side of the formula. It suffices to show that the right-hand side does not overestimate the probability, i.e., every instance S' counted by the right-hand side truly satisfies the desired properties: S' is legal and $\omega(x, S') = y$. Let S' be an instance counted by the right-hand side. The property $\omega(x, S') = y$ is obviously satisfied. To see S' is legal, by Lemma 5.1, we only need to verify the local legality of S' at every vertex in $S \backslash \{a_1\}$. Since S' does not contain any vertices outside $V(T_x)$, the local legalities at x and all $a_i \notin V(T_x)$ clearly hold. Also, S' is locally legal at any $a_i \in V(T_x) \backslash (\mathrm{ch}(x) \cup \{x\})$, because each factor $P_y(c)$ forces $S' \cap V(T_c)$ to be legal. Now we verify that S' is locally legal at any $c \in \mathrm{ch}(x)$. If $y \in V(T_c)$, then $\omega(c, S') = \omega(x, S') = y$ and hence S' is legal at c. If $y \notin V(T_c)$, then the factor $P_y(c)$ forces $(S' \cap V(T_c)) \cup \{y\}$

to be legal and thus either $\omega(c, S')$ is not defined or $dist(\omega(c, S'), y) \geq \ell$, which implies that S' is legal at c.

5.4.4 Proof of Lemma 5.3

When $y \in V(T_{\bar{p}(x)}) \backslash V(T_x)$, $P_y(x)$ is the probability that a realization $S' \subseteq_{\mathrm{R}} V(T_x)$ satisfies the conditions that $S' \cup \{y\}$ is legal and $\omega(\bar{p}(x), S' \cup \{y\}) = y$. Clearly, the empty sample $S' = \emptyset$ satisfies the two conditions and its probability is computed by the first term of the formula. If S' is not empty, then $\omega(x, S')$ is defined and must be some vertex $z \in V(T_x)$. In this case, we need $y \prec z$ to guarantee $\omega(\bar{p}(x), S' \cup \{y\}) = y$. Also, we need $dist(z, y) \geq \ell$ to ensure the legality of $S' \cup \{y\}$. Therefore, z must be a vertex in Γ. Now it suffices to show that the right-hand side of the formula does not overestimate the probability. In other words, we want that, if $S' \subseteq V(T_x)$ is legal and $\omega(x, S') = z$ for some $z \in \Gamma$, then $\omega(\bar{p}(x), S' \cup \{y\}) = y$ and $S' \cup \{y\}$ is also legal. The former can be easily seen from the facts that $\omega(x, S') = z$ and $y \prec z$. To see the latter, by Lemma 5.1, we only need to verify that $S' \cup \{y\}$ is locally legal at x (the local legalities of $S' \cup \{y\}$ at any vertex other than x is clear). Note that $z \in \Gamma$, so we have $dist(\omega(x, S'), y) = dist(z, y) \geq \ell$, which completes the proof.

5.4.5 Proof of Lemma 5.4

Clearly, if $y_i \prec z_j$, then $y_i \prec z_{j'}$ for any $j' > j$. Also, if $dist(z_j, y_i) \geq \ell$, then $dist(z_{j'}, y_i) \geq \ell$ for any $j' > j$, because both the paths $z_j \to y_i$ and $z_{j'} \to y_i$ go through the vertex $\bar{p}(x)$. Thus, we know that $\Gamma_y = \{z_j, z_{j+1}, \ldots, z_m\}$ for some $j \in \{1, \ldots, m\}$. To show the remaining part of the lemma, we notice that $\Gamma_{y_i} = \Gamma'_{y_i} \cap \Gamma''_{y_i}$, where $\Gamma'_{y_i} = \{z \in V(T_x) : y_i \prec z\}$ and $\Gamma''_{y_i} = \{z \in V(T_x) : dist(z, y_i) \geq \ell\}$. Both Γ'_{y_i} and Γ''_{y_i} are suffixes of the sequence $(z_1, \ldots, z_m)$. Furthermore, we have $\Gamma'_{y_1} \supseteq \cdots \supseteq \Gamma'_{y_r}$ and $\Gamma''_{y_1} \subseteq \cdots \subseteq \Gamma''_{y_r}$. As such, we can conclude that $\Gamma_{y_1} \subseteq \cdots \subseteq \Gamma_{y_k} \supseteq \cdots \supseteq \Gamma_{y_r}$ for some $k \in \{1, \ldots, t\}$.

5.4.6 Proof of Lemma 5.5

Suppose the tree space $\mathcal{T}$ is represented by a t-vertex weighted tree T_0. Let e be an edge of T_0, and $\hat{e} \subseteq \mathcal{T}$ be the subspace corresponding to e. Assume that $v_1, \ldots, v_k$

are the vertices of T lying in $\hat{e}$ (sorted in the order they appear on $\hat{e}$). We claim that among $v_1, \ldots, v_k$, there are only constant number of non-chain vertices. If the root of T is not in $\{v_1, \ldots, v_k\}$, then only v_1, v_2, v_{k-1}, v_k can be non-chain vertices. Otherwise, if the root is some v_i, then only $v_1, v_2, v_{i-1}, v_i, v_{i+1}, v_{k-1}, v_k$ can be non-chain vertices. In both the cases, the number of the non-chain vertices is constant. Finally, since T_0 has $(t-1)$ edges, the total number of the non-chain vertices of T is bounded by $O(t)$.

5.4.7 Proof of Lemma 5.6

First, for any index $r \in [m_i, m_{i+1})$, we show that $w_r \leq 2^{r-m_i} \cdot w_{m_i}$. When $r = m_i$, the inequality clearly holds. Assume, inductively, that the inequality holds for any index less than r' ($m_i < r' < m_{i+1}$). Since $r' \notin I$ and $m_i \in I$, we then have

$$w_{r'} \leq \sum_{j=1}^{r'-1} w_j < w_{m_i} + \sum_{j=m_i}^{r'-1} 2^{j-m_i} \cdot w_{m_i} = 2^{r'-m_i} \cdot w_{m_i},$$

which completes the induction. It follows that

$$s_i = \sum_{j<m_{i+1}} w_j < w_{m_i} + \sum_{j=m_i}^{m_{i+1}-1} 2^{j-m_i} \cdot w_{m_i} = 2^{m_{i+1}-m_i} \cdot w_{m_i}.$$

Thus, $|J_i| = O(\log_{1+\varepsilon} \frac{s_i}{w_{m_i}}) = O(\varepsilon^{-1}(m_{i+1} - m_i))$. Since $|J| = \sum_{i=1}^{k} |J_i|$, we can immediately conclude that $|J| = O(\varepsilon^{-1} n)$.

5.4.8 Proof of Theorem 5.3

Suppose that the stochastic dataset $S = \{a_1, \ldots, a_n\}$ is given in a metric space $\mathcal{X}$ with the metric $d_{\mathcal{X}}$. Let $G_{\mathcal{X}}$ be the metric graph of S, i.e., a weighted complete graph with vertex-set S such that the weight of each edge (a_i, a_j) is equal to $d_{\mathcal{X}}(a_i, a_j)$. Also, let T be a minimum spanning tree of $G_{\mathcal{X}}$. We then directly apply the method in Section 5.2.2 to the tree T to compute the quantity E via $O(\varepsilon^{-1} n)$ threshold probability queries. (Note that the threshold probability queries are made with respect to the metric of $\mathcal{X}$, the tree T is only used for choosing thresholds.) We show that E gives us a $(1+\varepsilon)$-approximation for $\mathbf{E}[\kappa(S)]$. The fact $E \leq \mathbf{E}[\kappa(S)]$ can be easily verified. To see the inequality $\mathbf{E}[\kappa(S)] \leq (1+\varepsilon)E$, we review the analysis in Section 5.2.2. Again, we use $e_1, \ldots, e_{n-1}$ to denote the edges of T with lengths (weights) $w_1 \leq \cdots \leq w_{n-1}$. As

that in Section 5.2.2, we have the index set $I = \{m_1, \ldots, m_k\}$, the jumps $J_1, \ldots, J_k$, and $J = J_1 \cup \cdots \cup J_k = \{\ell_1, \ldots, \ell_{|J|}\}$. Now we only need to verify that if ℓ_i and ℓ_{i+1} belong to different jumps, then there is no possible value of $\kappa(S)]$ within the range (ℓ_i, ℓ_{i+1}). As long as this is true, we can use the same argument as in Section 5.2.2 to show $\mathbf{E}[\kappa(S)] \leq (1+\varepsilon)E$. Let $d_T(a_i, a_j)$ be the distance between a_i and a_j in T (i.e., the length of simple simple path between a_i and a_j in T). Assume for a contradiction that $\ell_i \in J_r$, $\ell_{i+1} \in J_{r+1}$, and there exists $x, y \in S$ such that $\ell_i < d_{\mathcal{X}}(x, y) < \ell_{i+1}$. Observe that $\ell_i = s_r = \sum_{j<m_{r+1}} w_j$ and $\ell_{i+1} = w_{m_{r+1}}$. Since $d_T(x, y) \geq d_{\mathcal{X}}(x, y) > \ell_i$, there must be an edge e_m with $m \geq m_{r+1}$ on the path between x and y in T. However, this contradicts the fact that T is a minimum spanning tree, because $d_{\mathcal{X}}(x, y) < \ell_{i+1}$. As such, there is no possible value of $\kappa(S)$ within the range (ℓ_i, ℓ_{i+1}). By applying the analysis in Section 5.2.2, it turns out that $\mathbf{E}[\kappa(S)] \leq (1+\varepsilon)E$.

5.4.9 Proof of Lemma 5.7

Let $x \in \mathcal{T}$ be any point. We use B_x to denote the (open) δ-ball about x with δ small enough such that $\hat{c} \in B_x$ only if $\hat{c} = x$ for any center c (not necessarily critical). We first notice that $NNP_q(a_i) \leq NNP_x(a_i)$ for any $q \in B_x$ and any $a_i \in S$. This is because if $dist(x, a_j) < dist(x, a_i)$ then $dist(q, a_j) < dist(q, a_i)$. We further claim that $NNP_q(a_i) < NNP_x(a_i)$ for $q \in B_x$ iff there is a center c (not necessarily critical) with $\hat{c} = x$ such that a_i is involved by c and q is in a branch of c other than the one that contains a_i. To see this, consider a point $a_j \in S$ with $dist(x, a_j) = dist(x, a_i)$ and $dist(q, a_j) < dist(q, a_i)$. Note that such a point always exists, otherwise $NNP_q(a_i) = NNP_x(a_i)$. It is evident that q and a_j locate in the same connected component of $\mathcal{T} \backslash x$, which is other than the component contains a_i. Thus, the center $c = [m_{ij}]$ satisfies the desired properties. Now let us prove the lemma. Recall that Γ is the set of the critical centers of S. We show that any connected subspace $U \subseteq \mathcal{T}$ intersecting with (exactly) p cells in $\Psi_{\mathcal{T}}^S$ satisfies the condition that $p \leq \sum_{c \in \Gamma, \hat{c} \in U} deg(c) + 1$. When $p = 1$, this is trivially true. Assume that for any $p < p'$ the argument holds, and consider the case $p = p'$. Let C be a cell satisfying $C \cap U \cap \overline{U \backslash C} \neq \emptyset$. Note that such a cell always exists, unless U only intersects with one cell and then $p = 1$ (as U is connected). Choose a point $x \in C \cap U \cap \overline{U \backslash C}$ and define $X = \{c \in \Gamma : \hat{c} = x\}$. Suppose $U \backslash x$ has l connected components $U_1, \ldots, U_l$ among which there are l' components not intersecting with C. We denote by p_i the

number of the cells in $\Psi_{\mathcal{T}}^{S}$ intersecting with U_i. Then we have

$$p \leq \sum_{i=1}^{l} p_i - (l - l') + 1.$$

This is because the sum of all the p_i's counts the cell C exactly $(l - l')$ times and other cells intersecting with U exactly once. It is easy to observe that $p_i < p$. Then by our induction hypothesis, we have

$$\sum_{i=1}^{l} p_i \leq \sum_{c \in \Gamma, \hat{c} \in U \setminus x} deg(c) + l = \sum_{c \in \Gamma \setminus X, \hat{c} \in U} deg(c) + l.$$

Thus, it follows that

$$p \leq \sum_{c \in \Gamma \setminus X, \hat{c} \in U} deg(c) + l' + 1.$$

It now suffices to show $l' \leq \sum_{c \in X} deg(c)$. Let U_i be a component not intersecting with C and $q \in U_i \cap B_x$ be any point. Since $q \notin C$ and $q \in B_x$, x and q have different k-LNNs. As such, there exists a stochastic point a_j in the k-LNN of x such that $NNP_q(a_j) < NNP_x(a_j)$ (otherwise x and q have the same k-LNN, according to our observation $NNP_q(\cdot) \leq NNP_x(\cdot)$ presented at the beginning of the proof). Since $NNP_q(a_j) < NNP_x(a_j)$, there is a center c with $\hat{c} = x$ such that a_j is involved by c and q is in one branch of c (again, this follows from our observation at the beginning). Note that $c \in X$ as it is critical (c involves a_j and a_j is in the k-LNN of x). We then charge U_i to the branch of c containing q. We do this for all the l' components not intersecting with C. It is easy to verify that each branch of each center $c \in X$ is charged at most once, which immediately implies that $l' \leq \sum_{c \in X} deg(c)$. Consequently, the argument holds for $p = p'$ and hence for any p. By setting $U = \mathcal{T}$, we conclude that $|\Psi_{\mathcal{T}}^{S}| \leq \xi + 1$.

5.4.10 Proof of Lemma 5.8

We first prove the special case when $d = 1$. We show that the degree-sum of all the 1-shallow centers (i.e., the centers of depth 0 with respect to S) is at most $2n - 2$. If $n = 1$, this claim is clearly true, as there is no center. Assume the claim holds for any $n < n_0$, and consider the case that $n = n_0$. Let c be a center with $dep_S(c) = 0$. Suppose $deg(c) = g$ and $S_c \subseteq S$ is the set of points involved by c. Without loss of generality, assume $a_1 \in S_c$. We observe the following three facts.

• For $a_i, a_j \notin S_c$, $dep_S([m_{ij}]) = 0$ only if a_i and a_j are in the same connected components of $\mathcal{T}\backslash\hat{c}$. To see this, assume that a_i and a_j locate in different connected components. Then $dist(a_1, m_{ij}) < dist(a_i, m_{ij}) = dist(a_j, m_{ij})$ and hence $dep_S([m_{ij}]) > 0$.

• For $a_i \in S_c$ and $a_j \notin S_c$, $dep_S([m_{ij}]) = 0$ only if a_i and a_j are in the same connected component of $\mathcal{T}\backslash\hat{c}$, or a_j is not in any branch of c. To see this, assume a_i and a_j are located in different connected components of $\mathcal{T}\backslash\hat{c}$ and a_j is in the branch of c containing a_1 (without loss of generality). Then $dist(a_1, m_{ij}) < dist(a_i, m_{ij}) = dist(a_j, m_{ij})$ and hence $dep_S([m_{ij}]) > 0$.

• Let $a_i \notin S_c$ be a point which does not locate in any branch of c. Then the degree of the center $[m_{1i}]$ does not change if we "delete" all the points in $S_c\backslash\{a_1\}$. Formally, set $S' = S\backslash S_c \cup \{a_1\}$ and denote by $[m'_{1i}]$ the center of S' that contains the midpoint of a_1 and a_i. Then $deg([m_{ij}]) = deg([m'_{ij}])$. This observation follows immediately from the fact that all the points in S_c locate in the same connected components of $\mathcal{T}\backslash m_{1i}$.

With these observations, we now bound the degree-sum of the 1-shallow centers of S (denoted by ϕ). Suppose that $\mathcal{T}\backslash\hat{c}$ has p connected components $U_1, \ldots, U_p$, where $S \cap U_i = R_i$. If U_i is a branch of c, we use λ_i to denote the degree-sum of the 1-shallow centers of R_i, otherwise λ_i denotes the degree-sum of the 1-shallow centers of $R_i \cup \{a_1\}$ (here the depths of the considered centers are with respect to R_i or $R_i \cup \{a_1\}$ instead of S). Based on the above three observations and the induction hypothesis, we have

$$\phi \leq \sum_{i=1}^{p} \lambda_i + g \leq 2\sum_{i=1}^{p} |R_i| - 2g + g \leq 2n - g \leq 2n - 2.$$

Thus, the case of $d = 1$ is verified. To prove the result for a general d, we use the sampling argument. We sample each point in S independently with probability $1/d$. Let S' be the resulting random sample and φ be a random variable indicating the degree-sum of the 1-shallow centers of S' (the depths of the considered centers are with respect to S'). The previous proof for $d = 1$ implies that $\mathbf{E}[\varphi] \leq 2n/d$. Clearly, each center of S' is "contributed" by some center of S. For each center c of S, define a random variable $\sigma(c)$ such that $\sigma(c) = 0$ if c does not contribute a 1-shallow center of S', and $\sigma(c) = deg(c')$ if c contributes a 1-shallow center c' of S'. The event $\sigma(c) = 0$ happens whenever there are at most one point involved by c being sampled to S', or there are points closer to $\hat{c}$ (than those involved by c) being sampled to S'. We claim that, for any d-shallow center c of S, $\mathbf{E}[\sigma(c)] = \Omega(deg(c)/d^2)$. To see this, we set $g = deg(c)$

and $\theta = dep_S(c) < d$. Without loss of generality, assume $a_1, \ldots, a_g \in S$ are involved by c and belong to distinct branches of c. Define another random variable τ such that $\tau = |S' \cap \{a_1, \ldots, a_g\}|$ if c contributes a 1-shallow center and there are at least two points among $a_1, \ldots, a_g$ being sampled to S', and $\tau = 0$ otherwise. Observe that $\sigma(c) \geq \tau$. Thus, we have

$$\mathbf{E}[\sigma(c)] \geq \mathbf{E}[\tau] = \left(1 - \frac{1}{d}\right)^{\theta} \left(\frac{g}{d} - \frac{g}{d}\left(1 - \frac{1}{d}\right)^{g-1}\right) \geq \frac{g}{4d^2},$$

since $\theta < d$ and $g \geq 2$. It follows that

$$\frac{1}{4d^2} \sum_{c \in H_d} deg(c) \leq \sum_{c \in H_d} \mathbf{E}[\sigma(c)] \leq \mathbf{E}[\varphi] \leq \frac{2n}{d},$$

where H_d is the set of the d-shallow centers of S. As a result, the degree-sum of the d-shallow centers of S is at most $8dn$, completing the proof.

5.4.11 Proof of Lemma 5.9

Suppose $\pi_{a_1}, \ldots, \pi_{a_n} \in [\varepsilon, 1]$ for a constant $\varepsilon > 0$. Let c be a critical center of S with $dep_S(c) = d$. Without loss of generality, we assume

- $dist(a_1, \hat{c}) \leq dist(a_2, \hat{c}) \leq \cdots \leq dist(a_d, \hat{c}) < diam(c)$,
- a_{d+1} is involved by c and in the k-LNN of $\hat{c}$.

We claim that $d = O(k)$. The claim is trivial when $d \leq k$, thus assume $d > k$. Since a_{d+1} is in the k-LNN of $\hat{c}$, there must exist $i \leq k$ such that $NNP_{\hat{c}}(a_i) < NNP_{\hat{c}}(a_{d+1})$. It then follows that

$$(1 - \varepsilon)^{d-i+1} \geq \prod_{j=i}^{d} (1 - \pi_{a_j}) \geq \pi_{a_{d+1}} \prod_{j=i}^{d} (1 - \pi_{a_j}) > \pi_{a_i} \geq \varepsilon.$$

As a result, $d < \log_{1-\varepsilon} \varepsilon + i - 1 \leq \log_{1-\varepsilon} \varepsilon + k = O(k)$.

5.4.12 Proof of Lemma 5.10

Without loss of generality, assume $a_1, \ldots, a_k$ are k points closer to $\hat{c}$ than the c-involved points and $\pi_{a_1}, \ldots, \pi_{a_k}$ are greater than μ_0. Let $x \in S$ be any point involved by c. Since π_x is drawn from f, we must have $\pi_x \leq \mu$ by definition. We now show that x is not in the k-LNN of $\hat{c}$. We have the inequality

$$\frac{NNP_{\hat{c}}(x)}{NNP_{\hat{c}}(a_i)} \leq \frac{\pi_x(1 - \pi_{a_i})}{\pi_{a_i}} < \frac{\mu(1 - \mu_0)}{\mu_0} = 1,$$

for $i \in \{1, \ldots, k\}$. It follows that there are at least k points in S which have greater probabilities of being the nearest neighbor of $\hat{c}$ than x. Thus, x is not in the k-LNN of $\hat{c}$. Since x is arbitrarily chosen, we know that c is not critical, which completes the proof.

5.5 Details for constructing LVD data structure

5.5.1 Computing and sorting the centers

First of all, we apply Theorem 5.1 to obtain a new tree-space $\mathcal{T}'$ represented by an $O(n)$-vertex tree T' such that $S \subset \mathcal{T}' \subseteq \mathcal{T}$ and each stochastic point in S is located at a vertex of T'. This step takes $O(t + n \log n)$ time. Note that all the centers of S must be in $\mathcal{T}'$, so we can first work on $\mathcal{T}'$ and then map the computed centers back to $\mathcal{T}$. Before computing the centers, we do some preprocessing on the tree T'. For all pairs (e, v) where e is an edge and v is a vertex of T', we figure out the side of e that v locates on. This can be easily done in $O(n^2)$ time with a careful implementation. Furthermore, for each vertex v of T', we create a sorted list B_v which contains all points in S sorted according to their distances to v. This step can also be done in $O(n^2)$ time as follows. Observe that, if v and v' are adjacent vertices connected by an edge e, we can modify the sorted list B_v to obtain the list $B_{v'}$. Specifically, we separate B_v into two sorted sublists each of which contains the stochastic points on one side of e. Then $B_{v'}$ can be computed by merging these two sorted sublists in $O(n)$ time. Based on this observation, we can first straightforwardly create the sorted list for one vertex of T' in $O(n \log n)$ time, and keep modifying it to obtain the lists for other vertices, which takes $O(n^2)$ time in total. After the preprocessing, we are ready to compute the centers of S. The centers lying at any vertex v of T' can be directly found from the sorted list B_v. To compute the centers lying in the interior of an edge $e = (v, v')$, we utilize the sorted list B_v (or $B_{v'}$). Again, we separate B_v into two sorted sublists (say B'_v and B''_v) each of which contains the stochastic points on one side of e. We notice that a center in the interior of e involves a set A' of stochastic points located at the vertices in B'_v and a set A'' of stochastic points located at the vertices in B''_v. The points in A' must have the same distance to v (say d'), so are the points in A'' (say d''). Furthermore, we must have $0 < d'' - d' < w$, where w is the weight (length) of e. With these observations,

one can easily apply a standard sliding window technique to compute the centers in the interior of e in $O(\alpha + n)$ time where α is the number of the centers computed. Thus, the computation for all edges takes $O(n^2)$ time. After the centers are computed, we sort the centers in the interior of each edge e in the order they appear on e. This part takes $O(n^2 \log n)$ time in worst case. The final step is to map the centers back to the original tree space $\mathcal{T}$. If $\mathcal{T}'$ is constructed by applying the method in Section 5.1, then it is easy to keep a "relation" between T' and T during the construction. For example, for each edge e of T', we can record the edges of T intersecting with e in the order the intersections appear on e. With this information, as long as the centers in the interior of each edge of T' is sorted, the entire mapping process can be done in $O(t + n^2)$ time. At the end, after we map the centers to $\mathcal{T}$, we need to do another sort for the centers in the interior of each edge of T. The overall time for computing and sorting the centers is $O(t + n^2 \log n)$.

5.5.2 Constructing the LVD during the walk

During the walk, the nearest neighbor probabilities of $a_1, \ldots, a_n$ change only when we arrive at (resp., leave from) a center c from (resp., to) one of its branches. At this time, we need to update the nearest neighbor probabilities, maintain the binary search tree, and (possibly) retrieve the k-LNN from the binary search tree. Let m_c be the number of the stochastic points involved by c. Note that only these m_c stochastic points may change their nearest neighbor probabilities (this may be not true if there are other centers which have the same location as c, but the changes of the nearest neighbor probabilities of the points involved by other centers can be charged to those centers instead of c). The update of the nearest neighbor probabilities can be easily done in $O(m_c)$ time, if we store (before the walk) for each branch of a center c the product of the non-existence probabilities of the c-involved points in this branch. The maintenance of the binary search tree is achieved by $O(m_c)$ deletion and insertion operations, and thus takes $O(m_c \log n)$ time. Finally, the time for retrieving the k-LNN from the binary search tree is $O(\log n + k)$. Therefore, every time in the walk we arrive at c from one of its branches we spend $O(m_c \log n + k)$ time. Similarly for every time we leave from c along one of the branches in the walk. During the walk, we arrive at (resp., leave from) c $O(deg(c))$ times in total. It follows that the time cost charged to c is

$O(deg(c) \cdot m_c \log n + deg(c) \cdot k)$. Since we have $\sum_c deg(c) \cdot m_c = O(n^2)$, the overall time cost for the walk is $O(t + n^2 \log n + n^2 k)$. (There are also some low-level details for implementing the walk, e.g., how to know whether we are arriving at a center from one of its branches, etc. Such issues can be easily handled with enough preprocessing work before the walk.)

Chapter 6

Range closest pair queries

In this chapter, we study the range closest pair query problem when the query range Q is a (1) p-sided axis-aligned rectangle for $p = 2, 3, 4$; (2) halfplane; and (3) radius-fixed disc. In addition, we present a generic framework for solving the range closest pair query approximately. Applications of this framework include solutions where the query region is (1) a disc, (2) any translated and/or scaled copy of a so-called fat axes-aligned rectangle, (3) any translated and/or scaled copy of a fat convex shape of $O(1)$ complexity.

The high-level idea of all our solutions is based on the concept of a candidate pair; the same idea is also used in [2, 40]. Formally, a *candidate pair* is a pair of points that is reported by at least one query. It is clear that among all the $\Theta(n^2)$ pairs in a set of n input points, we only need to consider the candidate pairs. This motivates the so-called *candidate pair-based* approach, which first proves a sub-quadratic (usually linear, or nearly linear) bound on the number of candidate pairs and then builds on them a proper data structure that can efficiently answer each query.

Throughout the chapter, the reader will find that different strategies are applied to help bound the number of candidate pairs: we may analyze the complexity directly, or under some reasonable assumptions, or in a subproblem decomposed from the original one. (Indeed, many of these combinatorial analyses are of independent interest.) Nevertheless, the goal is to prove that there can be only a small number of candidate pairs so that we can design an efficient data structure and reporting algorithm.

Finally, we introduce some global naming conventions used throughout the chapter.

Let S be a set of n input points in the plane and Q be the query range. Let $\mathcal{C}_S$ consist of all candidate pairs when the type of Q is fixed. Define $\text{dist}(\cdot,\cdot)$ as the L_2-distance metric. Also, for a point p in the plane, we write its abscissa and ordinate as $p.x$ and $p.y$, respectively.

6.1 Axes-aligned rectangle query

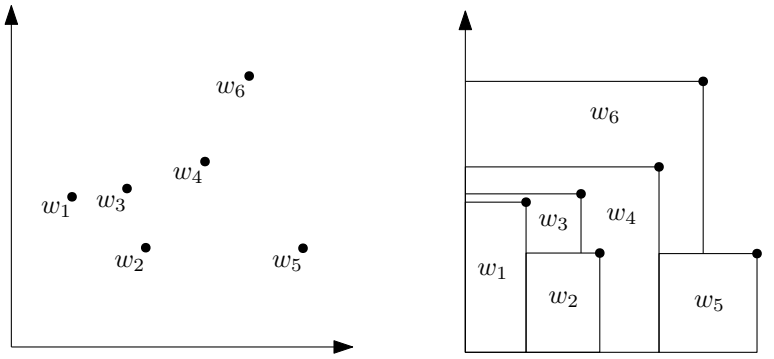
In this section, we consider the range closest pair problem with the query being a p-sided axes-aligned rectangle for $p = 2, 3, 4$.

6.1.1 Quadrant query

Let the query range $Q = [x, \infty) \times [y, \infty)$ be any northeast quadrant, and for convenience, we write $Q = (x, y)$ for short. From [40], we know the number of candidate pairs is $O(n)$. Let $\mathcal{C}_S = \{\theta_1, \theta_2, \ldots, \theta_m\}$ consist of all such pairs, where $\theta_i = (a_i, b_i)$ and $m = O(n)$. For each θ_i, define w_i to be the weighted quadrant $(-\infty, \min(a.x, b.x)] \times (-\infty, \min(a.y, b.y)]$ with the weight equal to $\text{dist}(a, b)$. Let $\mathcal{W} = \{w_1, \ldots, w_m\}$, and assume that the elements in $\mathcal{W}$ are sorted in increasing order of their weights. We then create a planar subdivision, $\mathcal{A}$, by successively overlaying the quadrants in $\mathcal{W}$. (See Figure 6.1.) Formally, the cell corresponding to the quadrant w_i, denoted by $c(w_i)$, is $w_i \setminus \bigcup_{j=1}^{i-1} w_j$, and it is easy to verify that the closest pair in $Q = (x, y)$ is θ_i if and only if (x, y) lies in $c(w_i)$. Therefore, the query is naturally mapped to a planar point location problem, which can be solved optimally by, for instance, persistent search trees [52].

Finally, we observe that every quadrant creates at most two intersections, hence, $\mathcal{A}$ has $O(m) = O(n)$ vertices, edges, and faces. Therefore, the point location structure occupies $O(n)$ space and can answer each query in $O(\log n)$ time. As such, we claim the following result.

Theorem 6.1. *A set S of n points in $\mathbb{R}^2$ can be preprocessed into a data structure of size $O(n)$ such that, for any quadrant query Q, the closest pair in $S \cap Q$ can be reported in $O(\log n)$ time.*



(a) Corners of weighted quadrants $\{w_1, \ldots, w_6\}$, indexed by increasing weight.

(b) The subdivision $\mathcal{A}$ induced by quadrants $\{w_1, \ldots, w_6\}$.

Figure 6.1: Illustrating weighted quadrants and their induced subdivision.

6.1.2 Strip query

Again, let $\mathcal{C}_S$ denote the candidate pair set when the query Q is a closed subset of $\mathbb{R}^2$ bounded by two vertical lines, which we will refer to as a *strip*. It has been proved in [55] that $|\mathcal{C}_S|$ is $O(n \log n)$, and we will further show that this bound is indeed tight. Construct a set of points $S_{\text{strip}} = \{a_0, a_2, \ldots, a_{n-1}\}$, where n is assumed to be a power of 2, and $s_i = (i/n, 3^{\text{mir}(i)})$. Here, $\text{mir}(i)$ is a function that mirrors the binary representation of i. Formally, let i be written as $(b_{k-1}b_{k-2}\ldots b_1 b_0)_2$ in base-2 where $k = \log n$; then $\text{mir}(i) = (b_0 b_1 \ldots b_{k-2}b_{k-1})_2$. We then have the following lemma.

Lemma 6.1. *The number of candidate pairs of S_{strip} is $\Omega(n \log n)$.*

Proof. First, it is easy to see that a_i and a_{i+1} form a candidate pair (w.r.t. the strip defined by the vertical lines containing a_i and a_{i+1}, respectively) as there are no other points in between. This contributes $n - 1$ pairs. Then, the idea is to evenly divide these n points into two subsets, i.e., $\{a_{2i}\}$ and $\{a_{2i+1}\}$, and separate (vertically) the two sets as far as possible. This way, the pairwise-distances crossing the two sets will be significantly larger than those generated from the same set. Thus, we can treat each subset as an independent instance of size $n/2$, i.e., each subset will contribute $n/2 - 1$ pairs (in the form of (a_i, a_{i+2})). We then recursively work on these two smaller

subproblems, and the same pattern recurs. See Figure 6.2 for an example.

Finally, it can be verified that the $3^{\mathrm{mir}(i)}$-ordinate constraint guarantees that the vertical distances between any two different groups (of the same size) are significantly large so that the two groups can be considered independently. (Any integer greater than 3 works.) Therefore, let $E(n)$ denote the number of candidate pairs in S_{strip}. Then $E(n) \geq 2E(n/2) + n - 1$, which implies that $E(n) = \Omega(n \log n)$. □

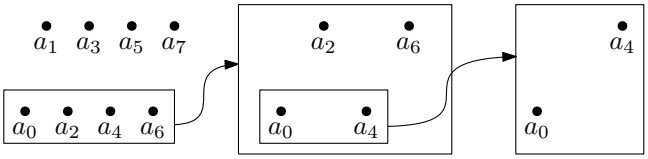


Figure 6.2: An example of eight points, recursively showing three groups whose sizes decrease at each step by a factor of 2. The left, middle, and right group contributes at least 7, 3, and 1 candidate pairs, respectively.

From Lemma 6.1 and the result in [55] it follows that the number of candidate pairs for strip queries in $\mathbb{R}^2$ is $\Theta(n \log n)$. We now describe an efficient query structure.

Let $Q = [u, v] \times (-\infty, \infty)$ be the query strip and (a, b) be a candidate pair. Then $(a, b) \in Q$ iff $-\infty < u \leq \min(a.x, b.x)$ and $\max(a.x, b.x) \leq v < \infty$, i.e., iff the point (u, v) is in the quadrant $(-\infty, \min(a.x, b.x)] \times [\max(a.x, b.x), \infty)$. For each candidate pair $\theta \in \mathcal{C}_S$ where $\theta = (a, b)$, we map it to a weighted quadrant $(-\infty, \min(a.x, b.x)] \times [\max(a.x, b.x), \infty)$ with the weight equal to $\mathrm{dist}(a, b)$. Thus, one can build a data structure similar to the one in Section 6.1.1 and solve the strip query in $O(\log |\mathcal{C}_S|) = O(\log(n \log n)) = O(\log n)$ time using $O(n \log n)$ space. We therefore obtain the following theorem, which improves the previous results in [55] by a $\log n$ factor in space.

Theorem 6.2. *A set S of n points in $\mathbb{R}^2$ can be preprocessed into a data structure of size $O(n \log n)$ such that, for any query strip Q, the closest pair in $S \cap Q$ can be reported in $O(\log n)$ time.*

Alternatively, we shall see, in Section 6.1.3, that there are only $O(n)$ candidate pairs if S is what we call $O(1)$-flat (see Definition 6.1). Then we immediately have the following.

Theorem 6.3. *If S is $O(1)$-flat, it can be preprocessed into a data structure size $O(n)$ such that, for any strip Q, the closest pair in $S \cap Q$ can be reported in $O(\log n)$ time.*

6.1.3 3-sided rectangular query

The candidate pair based method does not work well when Q is a 3-sided rectangle (say, $Q = [x_1, x_2] \times (-\infty, y]$), as Lemma 6.2 shows that in the worst case there can be a quadratic number of candidate pairs so that the space will be too high.

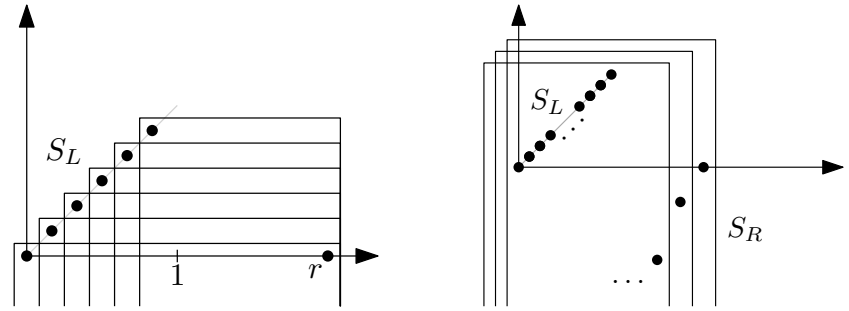
Lemma 6.2. $|\mathcal{C}_S| = \Omega(n^2)$ *if* $|S| = \Theta(n)$.

Proof. We create two point sets S_L and S_R, each of which contains n points. Let $S_L = \{l_0, \ldots, l_{n-1}\}$ and $S_R = \{r_0, \ldots, r_{n-1}\}$, where $l_j = (j/n, j/n)$ and $r_i = (2 - i/n, 1 - 3^i)$, for $i, j = 0, 1, \ldots, n-1$. (We note that the points in S_R are sorted in decreasing order of their abscissas.) We shall see that there are at least n^2 candidate pairs. Indeed, any $l_j \in S_L$ and $r_i \in S_R$, (l_j, r_i) forms a candidate pair. To see this, we fix $r_i = (2 - i/n, 1 - 3^i)$ and consider the following n 3-sided query rectangles, $Q_0^{(i)}, \ldots, Q_{n-1}^{(i)}$, where $Q_j^{(i)} = [(j - 0.5)/n, 2 - (i - 0.5)/n] \times (-\infty, (j + 0.5)/n]$; see Figure 6.3 for an example. It is clear that $Q_j^{(i)} \cap (S_L \cup S_R) = \{l_j, r_i, r_{i+1}, \ldots, r_{n-1}\}$, and one can verify that

1. $\text{dist}(l_j, r_i) < \text{dist}(l_j, r_{i+1}) < \cdots < \text{dist}(l_j, r_{n-1})$,
2. $\text{dist}(r_i, r_{i+1}) < \text{dist}(r_{i+1}, r_{i+2}) < \cdots < \text{dist}(r_{n-1}, r_{n-2})$, and
3. $\text{dist}(l_j, r_i) < \text{dist}(r_i, r_{i+1})$.

Therefore, the closest pair in $Q_j^{(i)} \cap (S_L \cup S_R)$ is (l_j, r_i). As such, there are at least n^2 distinct candidate pairs among the given $2n$ points. □

One may observe that, in the worst-case example above, S_R has a skewed distribution, i.e., the slope between consecutive points can be very large and unbounded. In fact, if we prevent any large consecutive slopes (in absolute values) defined by points in the input set, there can be only linear number candidate pairs. To formally see this, we introduce in Definition 6.1 a concept called *α-flat* and show in Lemma 6.3 that if S is $O(1)$-flat there can be only a linear number of candidate pairs.



(a) Fix some $r \in S_R$. For every $l \in S_L$, (l, r) is a candidate pair for some 3-sided query Q.

(b) Illustrating candidate pairs between S_L and S_R.

Figure 6.3: A worst-case example for a 3-sided rectangular query

Definition 6.1. *Let $\alpha \in [0, \pi/2)$ be a constant, and assume $S = \{s_1, \ldots, s_n\}$ is sorted in x-order. Then, S is* α-flat *if and only if, for all $1 \leq i < n$,*

$$\left| \frac{s_i.y - s_{i+1}.y}{s_i.x - s_{i+1}.x} \right| \leq \tan \alpha.$$

Corollary 6.1. *If $S = \{s_1, \ldots, s_n\}$ is α-flat, then for any $1 \leq i < j \leq n$ we have $|(s_i.y - s_j.y)/(s_i.x - s_j.x)| \leq \tan \alpha$.*

Lemma 6.3. *There are only $O(n)$ candidate pairs if S is $O(1)$-flat for any 3-sided query rectangle.*

Proof. Assume S is α-flat, where $\alpha = O(1)$. Let $\mathcal{C}_S = \mathcal{C}_S^+ \cup \mathcal{C}_S^-$ consist of all the candidate pairs of S, where the slope of each pair in $\mathcal{C}_S^+$ (resp. $\mathcal{C}_S^-$) is positive (resp. non-positive). For each $(a, b) \in \mathcal{C}_S^+$ where $a.x < b.x$, we charge its existence to b; symmetrically, for each $(a, b) \in \mathcal{C}_S^-$ where $a.x < b.x$, we charge it to a. We shall show that $|\mathcal{C}_S^-| = O(n)$, and via a symmetric argument, $|\mathcal{C}_S^+| = O(n)$, which implies that $|\mathcal{C}_S| = |\mathcal{C}_S^+| + |\mathcal{C}_S^-| = O(n)$.

To see why $|\mathcal{C}_S^-| = O(n)$, we fix the left endpoint, a, of any pair in $\mathcal{C}_S^-$ and argue that there can be only $O(1)$ b_i's such that $(a, b_i) \in \mathcal{C}_S^-$. Let the right endpoints be $b_1, b_2, \ldots, b_k$ that are sorted in clock-wise order around a. It then can be verified that

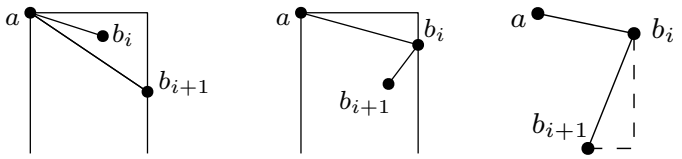
1. $b_i.x > b_{i+1}.x$,

2. $\text{dist}(a, b_i) \leq \text{dist}(b_i, b_{i+1})$, and

3. $b_i.x - b_{i+1}.x \geq \text{dist}(b_i, b_{i+1}) \cos\alpha \geq \text{dist}(a, b_i) \cos\alpha$.

To see (1), for a contradiction, assume $b_i.x < b_{i+1}.x$; thus, $\text{dist}(a, b_i) < \text{dist}(a, b_{i+1})$. This is impossible as the minimum 3-sided rectangle that contains a and b_{i+1} must also contain b_i, which means (a, b_{i+1}) can never be a candidate pair. Now, since b_{i+1} is to the left of b_i (and clockwise from b_i), the minimum 3-sided rectangle that contains a and b_i must also contain b_{i+1}, which proves (2). Once (1) and (2) are seen to be true, it is easy to verify (3) as S is α-flat. Please refer to Figure 6.4 for an example of all the three cases.

Finally, we have $\text{dist}(a, b_1) \geq b_1.x - a.x = (b_1.x - b_2.x) + (b_2.x - b_3.x) + \cdots + (b_{k-1} - b_k) + (b_k - a.x) \geq (b_1.x - b_2.x) + (b_2.x - b_3.x) + \cdots + (b_{k-1} - b_k) \geq (\cos\alpha)(\text{dist}(a, b_1) + \cdots + \text{dist}(a, b_{k-1})) \geq (k-1)(\cos\alpha)\text{dist}(a, b_1)$. Therefore, $k \leq 1 + \sec\alpha = O(1)$. □



(a) Case 1 (b) Case 2 (c) Case 3

Figure 6.4: Illustrating the three cases of Lemma 6.3

A natural way to answer a 3-sided rectangle query is to decompose it into several strip queries via a two-level range tree. Specifically, we create a two-level range tree, $\mathcal{T}$, on the points in S where the leaves are sorted in y-order. In each internal node $u \in \mathcal{T}$, we build a data structure discussed in Section 6.1.2 on those points that are leaves of u's subtree to handle any strip query. It is clear that each level of $\mathcal{T}$ costs $O(n)$ space, and thus, the total space occupied is $O(n \log n)$. When a 3-sided query $Q = [a, b] \times (-\infty, c]$ comes in, we first traverse $\mathcal{T}$ with the range $(-\infty, c]$ and identify $O(\log n)$ canonical nodes. Then, for each canonical node u, we launch a strip query with the range $[a, b]$ and find the closest pair inside (if exists). Finally, we collect the $O(\log n)$ pairs and report the one with the minimum pairwise distance. By Theorem 6.3, each canonical node takes $O(\log n)$ time, and the total query is therefore $O(\log^2 n)$.

Theorem 6.4. *If S is $O(1)$-flat and has n points in $\mathbb{R}^2$, then it can be preprocessed into a data structure of size $O(n \log n)$ such that, for any 3-sided rectangle Q, the closest pair in $S \cap Q$ can be reported in $O(\log^2 n)$ time.*

6.1.4 4-sided rectangular query

Since it has been shown that there can be $\Theta(n^2)$ candidate pairs for 3-sided rectangular queries, the same conclusion applies for 4-sided queries. Therefore, we still make the assumption that the input S is $O(1)$-flat. However, Lemma 6.3 is not necessarily true when $p = 4$. Thus, we cannot directly generalize the solution in Section 6.1.3 by using a three-level range tree. Instead, we show how to improve the results in [40] when S is $O(1)$-flat.

Specifically, the 4-sided query algorithm in [40] has an $O(\log^2 n)$ query time using $O(n \log^5 n)$ space. It relies on a so-called *anchored 3-sided rectangle query*, where the query Q is a 3-sided rectangle that always intersects with some vertical line ℓ. This problem was solved in $O(\log^2 n)$ time using $O(n \log^3 n)$ space. By applying Theorem 6.4, the space complexity can be improved to $O(n \log n)$ if S is $O(1)$-flat. Therefore, we obtain a solution for a 4-sided query with $O(\log^2 n)$ query time using $O(n \log^3 n)$ space. We note that this result is also better than the one in [55] with $O(\log^3 n)$ query time and $O(n \log^3 n)$ space.

Theorem 6.5. *If S is $O(1)$-flat and has n points in $\mathbb{R}^2$, then it can be preprocessed into a data structure of size $O(n \log^3 n)$ such that, for any 4-sided rectangle Q, the closest pair in $S \cap Q$ can be reported in $O(\log^2 n)$ time.*

6.1.5 Connection to range min-gap query

In this section, we provide some evidence that reflects the potential hardness of finding the closest pair in a query strip. We start by reviewing a common problem called *min-gap* and then generalize it to a query version.

Formally, the min-gap of an array $A[1..n]$ of $n \geq 2$ reals is defined as $\min_{1 \leq i < j \leq n} |A_i - A_j|$. In its query version, a *range min-gap query* receives two integers l and r, where $1 \leq l < r \leq n$, and outputs the min-gap of the subarray $A[l..r]$. One is allowed to preprocess A into some data structure and use it to answer each query efficiently.

Specifically, let $S(n)$ be the size of the underlying data structure and $T(n)$ be the query time. A data structure with $S(n) = O(n^2)$ and $T(n) = O(1)$ is trivial but occupies too much space, and thus is not of interest. Other than that, to our best knowledge, there is very little existing work on this topic, and perhaps, only the following generic framework solves the problem in sublinear time while using subquadratic space:

1. Using Mo's algorithm [1] (also known as square root decomposition), one can answer the range min-gap query in a reasonably satisfactory time, but in an offline fashion. That is, given m queries beforehand, Mo's algorithm can answer all the queries in $O((m+n)\sqrt{n}\log n)$ using $O(n)$ space. Via an amortized analysis, $T(m,n) = O((m+n)\sqrt{n}(\log n)/m) = \Omega(\sqrt{n})$. We elaborate in Section 6.5.

2. One can try to further improve the performance of Mo's algorithm by building a rectilinear minimum spanning tree on the mapped query intervals and answering each query via an Euler tour along the tree, but unfortunately this does not help with the worst case performance. (Again, please see Section 6.5 for more details.)

Therefore, even for an offline version, there does not seem to be a satisfactory solution in the literature, not to speak of an online version.

Next, we show that our strip closest pair query is at least as hard as the range min-gap by the following reduction. Given $A[1..n]$, we construct a set of points $S = \{s_1, s_2, \ldots, s_n\}$, where $s_i = (i\varepsilon, A_i)$ and $\varepsilon \to 0$ is a sufficient small positive constant. Now, the Euclidean distance between s_i and s_j is $\sqrt{(A_i - A_j)^2 + \varepsilon^2(i-j)^2} \approx |A_i - A_j|$. This way, when ε is small enough, the closest pair in the range $[l\varepsilon, r\varepsilon] \times (-\infty, \infty)$ will naturally yield the min-gap of the subarray $A[l..r]$. For instance, the reader can verify that it works for any $\varepsilon \leq \sqrt{g_2^2 - g_1^2}/n$, where $g_1 < g_2$ are the smallest and the second smallest gap in A. (Both of g_1 and g_2 can be computed efficiently in $O(n \log n)$ time.) By plugging in the results in Section 6.1.2, we obtain an online solution to the range min-gap query with $O(\log n)$ query time and $O(n \log n)$ space, which significantly improves the query time but occupies slightly more space. On the other hand, the range min-gap query shows that it may be difficult to further improve our results for the strip query, say, obtaining a linear-space solution with logarithmic or even $o(\sqrt{n})$ query time.

6.2 Halfplane query

In this section, we consider the range closest pair problem with the query Q being a halfplane. Our goal is to preprocess the dataset S into a data structure $\mathcal{D}$ such that for any given halfplane $Q : y \geq ux + v$, the closest pair of points in $S \cap Q$ can be reported efficiently. (Note that the halfplanes of the form $y \leq ux + v$ can be handled similarly by building another data structure symmetric to $\mathcal{D}$.)

By using duality, a non-vertical line $\ell : y = ux+v$ is mapped to the point $\ell^* = (u, -v)$ and a point $p = (s, t)$ is mapped to the line $p^* : y = sx-t$. Also, ℓ is below (resp. above) p iff ℓ^* is above (resp. below) p^*. Then handling halfplane queries can be transformed into a point-location problem. The line bounding the query halfplane $Q : y \geq ux + v$ corresponds to the point $Q^* = (u, -v)$ in the dual space (of $\mathbb{R}^2$). So if we decompose the dual space into "cells" such that the points (corresponding to the bounding lines of halfplanes in the original space) in each cell have the same answer for the closest pair, then any point-location technique can be applied to solve the problem directly. Now the crucial thing we need to consider is the structure of such a decomposition or arrangement and, more importantly, its complexity.

Since there are n lines in the dual space, one per point of S, a trivial upper bound on the complexity of the desired arrangement $\mathcal{A}$ is $O(n^2)$. However, by using additional properties of the problem at hand we show that, surprisingly, the complexity of $\mathcal{A}$ is in fact $O(n)$.

6.2.1 Complexity of the arrangement $\mathcal{A}$

The first result we need is that there can be only $O(n)$ candidate pairs.

Lemma 6.4. [2] *If (a, b) and (c, d) are both candidate pairs such that a, b, c, d are distinct points in S, then the segment $\overline{ab}$ does not properly intersect the segment $\overline{cd}$. Hence, the number of the candidate pairs is $O(n)$.*

Proof. Let Q_{ab} and Q_{cd} be the halfplanes in which the closest pair is (a, b) and (c, d), respectively. Then, Q_{ab} contains a, b and at least one of c or d. Similarly, Q_{cd} contains c, d and at least one of a or b. W.l.o.g., let $c \in Q_{ab}$ and $a \in Q_{cd}$. Since (a, b) is the closest pair in Q_{ab}, $|ab| \leq |bc|$. Similarly, $|cd| \leq |ad|$ as (c, d) is the closest pair in Q_{cd}.

Then, $|ab| + |cd| \leq |bc| + |ad|$. Now, for a contradiction, assume $\overline{ab}$ and $\overline{cd}$ properly intersect at e. By triangle inequality, $|ae| + |de| > |ad|$ and $|be| + |ce| > |bc|$. Thus, $|ab| + |cd| = (|ae| + |be|) + (|ce| + |de|) > |bc| + |ad|$, which is a contradiction.

The graph with vertex set S and edges consisting of line segments joining candidate pairs is planar (since the line segments are properly non-crossing). Thus, the number of edges, hence the number of candidate pairs, is $O(n)$. □

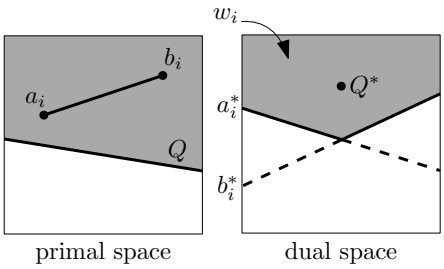
With this in hand, we now consider the complexity of $\mathcal{A}$. For a candidate pair (a, b), define its *length* as the L_2-distance between a and b in $\mathbb{R}^2$. Suppose we have m candidate pairs $\theta_1, \ldots, \theta_m$ sorted by their lengths (from the shortest to the longest) where $\theta_i = (a_i, b_i)$. If we observe each pair θ_i in the dual space, we get two lines a_i^* and b_i^* corresponding to a_i and b_i, respectively. The pair θ_i is contained in Q iff Q^* is below above a_i^* and b_i^*, i.e., Q^* is in the upward-open *wedge* generated by a_i^* and b_i^*, which we denote by w_i; see Figure 6.5a. As such, the closest pair answer for Q to be reported is the candidate pair θ_j with

$$j = \min\{i : Q \in w_i\}.$$

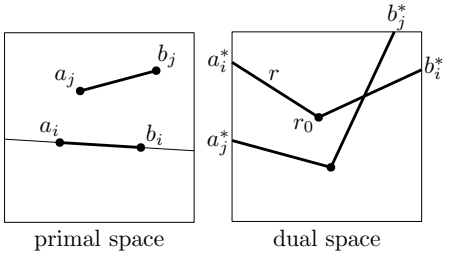
This observation gives us a new way to view the arrangement $\mathcal{A}$. We begin with the trivial decomposition $\mathcal{P}_0$ of the dual space (plane), i.e., a decomposition with only one face which is the entire plane. We construct a decomposition $\mathcal{P}_i$ by merging $\mathcal{P}_{i-1}$ and w_i as follows. Let o_{i-1} be the *outer face* of $\mathcal{P}_{i-1}$, i.e., the complement of $\bigcup_{j=1}^{i-1} w_j$.Then $\mathcal{P}_i$ is obtained from $\mathcal{P}_{i-1}$ by decomposing the face o_{i-1} via the wedge w_i. In other words, we obtain $\mathcal{P}_i$ by first removing the face o_{i-1} from $\mathcal{P}_{i-1}$ and then adding $o_{i-1} - w_i$ and all the connected components of $o_{i-1} \cap w_i$ and as new faces. Note that $o_{i-1} - w_i$ is the complement of $\bigcup_{j=1}^{i} w_j$. One can easily verify that $o_{i-1} - w_i$ is connected and becomes the outer face o_i of $\mathcal{P}_i$. In this way, we construct $\mathcal{P}_1, \ldots, \mathcal{P}_m$ in order. Now each $\mathcal{P}_i$ is a polygonal decomposition, and we use $\mathcal{A}_i$ to denote the corresponding arrangement. By the above argument, we know that $\mathcal{P}_m$ is the desired decomposition and hence $\mathcal{A}_m = \mathcal{A}$. We denote the complexity of $\mathcal{A}$ by $|\mathcal{A}|$. To bound $|\mathcal{A}|$, we prove the following result.

Theorem 6.6. $|\mathcal{A}_i| - |\mathcal{A}_{i-1}| = O(1)$. *In particular,* $|\mathcal{A}| = |\mathcal{A}_m| = O(m) = O(n)$.

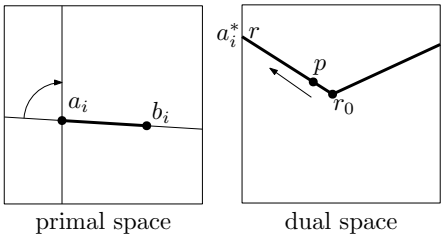
Proof. Let o_i be the outer face of $\mathcal{P}_i$, and c_i be the boundary of the wedge w_i (which consists of two rays emanating from the intersection point of a_i^* and b_i^*). We



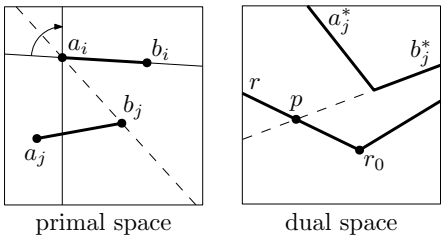
(a) Pair (a_i, b_i) is in halfplane Q iff Q^* is in the upward-open wedge w_i formed by a_i^* and b_i^*.



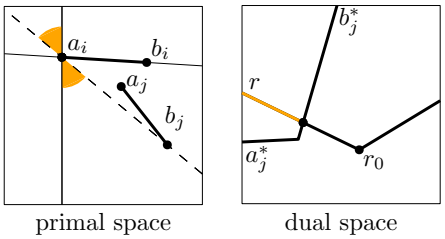
(b) The case for $j \in J_1$



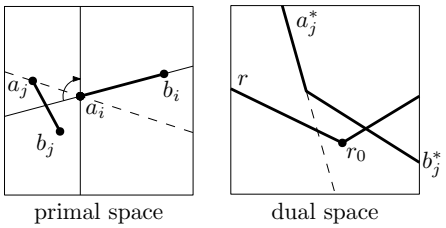
(c) Moving p and rotating line $a_i b_i$ clockwise around a_i



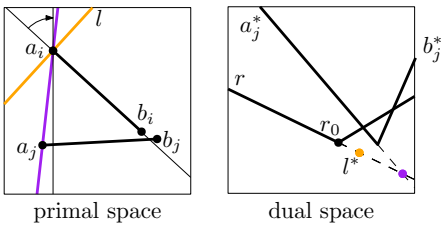
(d) An example for $j \in J_2$ where $r \cap w_j = \emptyset$.



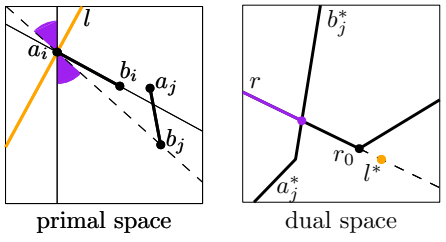
(e) An example for $j \in J_2$ where $r \cap w_j \neq \emptyset$.



(f) An example for $j \in J_3$ where $a_j b_j$ is to the left and $r \cap w_j = \emptyset$.

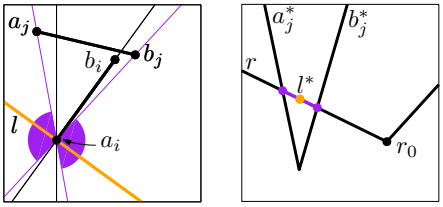


(g) An example for $j \in J_3$ where $l^* \notin r$ and $r \cap w_j = \emptyset$.



(h) An example for $j \in J_3$ where $l^* \notin r$ and $r \cap w_j \neq \emptyset$.

Figure 6.5: Illustrating the various cases in Theorem 6.4.



(i) An example for $j \in J_3$ where $l^* \in r$ and $r \cap w_j$ contains l^* but does not contain r_0 or the infinite end of r.

Figure 6.5: Illustrating the various cases in Theorem 6.4 (continued).

first note that, to deduce that $|\mathcal{A}_i| - |\mathcal{A}_{i-1}|$ is $O(1)$, it suffices to show that the number of connected components of $c_i \cap o_{i-1}$ is constant. This is because every connected component of $c_i \cap o_{i-1}$ contributes to $\mathcal{A}_i$ exactly one new face, a constant number of new vertices, and a constant number of new edges. Indeed, we only need to check one branch of c_i (i.e., one of the two rays of c_i), say the branch corresponding to a_i^* (we denote it by r). We will show that $r \cap o_{i-1}$ has $O(1)$ connected components. Without loss of generality, we may assume that a_i is to the left of b_i. Then each point on r corresponds to a line in the original space, which goes through the point a with the segment $\overline{ab}$ above it. Note that

$$r \cap o_{i-1} = r - \bigcup_{j=1}^{i-1} w_j = r - \bigcup_{j=1}^{i-1} (r \cap w_j),$$

and each $r \cap w_j$ is a connected subset of r. We consider each pair θ_j with $j < i$ and analyze the intersection $r \cap w_j$. There are three cases: (1) both a_j and b_j are above the line a_ib_i, (2) both of a_j and b_j are below a_ib_i, or (3) one of a_j and b_j is (strictly) above a_ib_i while the other is (strictly) below a_ib_i. We use J_1, J_2, J_3 to denote the index sets corresponding to the three cases (so $J_1 \cup J_2 \cup J_3 = \{1, \ldots, i-1\}$).

Case 1: If $j \in J_1$, the wedge w_j must contain the initial point r_0 of r (i.e., the intersection point of a_i^* and b_i^*, which is the dual of the line a_ib_i), because r_0 must be above both a_j^* and b_j^*. (See Figure 6.5b.)

Case 2: For $j \in J_2$, we claim that either $r \cap w_j$ is empty or it contains the infinite end of r (i.e., the point at infinity along r). Imagine that we have a point p moving

along r from r_0 to the infinite end of r. In the original space, p corresponds to a line rotating clockwise around a_i from the line a_ib_i to the vertical line through a_i; see Figure 6.5c. Note that $r \cap w_j$ contains p only when both a_j and b_j are above the dual line of p. But a_j and b_j are below the line a_ib_i for $j \in J_2$. When p is moving, the region below a_ib_i and above the dual line of p expands. As such, one can easily see that $r \cap w_j$ must contain the infinite end of r if it is nonempty. (See Figure 6.5d and 6.5e.)

Case 3: Finally, we consider $j \in J_3$. In this case, one point of θ_j is (strictly) above the line a_ib_i while the other one is (strictly) below a_ib_i. Thus, the segment $\overline{a_jb_j}$ must intersect the line a_ib_i. However, by Lemma 6.4, $\overline{a_jb_j}$ cannot intersect the segment $\overline{a_ib_i}$. So the intersection point of $\overline{a_jb_j}$ and the line a_ib_i is either to the left of a_i or to the right of b_i (recall that a_i is assumed to be to the left of b_i).

• If the intersection point is to the left of a_i, we argue that $r \cap w_j$ is empty. Observe that the dual line of any point on r is through a_i and below b_i, meaning that it must be above the intersection point (when the intersection point is to the left of a_i). In other words, the dual line of any point on r is above at least one of a_j and b_j, and thus any point on r is not contained in the wedge w_j, i.e., $r \cap w_j$ is empty. (See Figure 6.5f.)

• The trickiest case occurs when the intersection point of $\overline{a_jb_j}$ and the line a_ib_i is to the right of b_i. In such a case, we consider the line through a_i perpendicular to a_ib_i, which we denote by l. We first argue that both a_j and b_j must be on the same side of l as b_i. Since $\overline{a_jb_j}$ intersects the line a_ib_i to the right of b_i, at least one of a_j and b_j is on the same side of l as b_i. But we notice that $\overline{a_jb_j}$ cannot intersect l, otherwise the length of θ_j is (strictly) more than the length of θ_i, which contradicts the fact that $j < i$ (recall that $\theta_1, \ldots, \theta_m$ is sorted from the shortest to the longest). So the only possibility is that both a_j and b_j are on the same side of l as b_i. Now we have two sub-cases.

(i) l has no dual point l^* (i.e., l is vertical) or the dual point l^* of l is not on the ray r. In this case, we look at the point p moving along r from r_0 to the infinite end of r. Clearly, when p moves, the region to the right of l and above the dual line of p expands. Thus, either $r \cap w_j$ is empty or it contains

the infinite end of r. (See Figure 6.5g and 6.5h.)

(ii) The dual point of l is on r. Then $r \cap w_j$ may be a connected portion of r containing neither r_0 nor the infinite end of r. However, as b_i is above the line l in this case, we have that both a_j and b_j are above l. This implies that $r \cap w_j$ contains the dual point of l. (See Figure 6.5i.)

In sum, we conclude that for any $j \in \{1, \ldots, i-1\}$, the intersection $r \cap w_j$ might be (1) empty, (2) a connected subset of r containing r_0, (3) a connected subset containing the infinite end of r, or (4) a connected portion containing the dual point of l (if the dual point of l is on r). As such, the union $\bigcup_{j=1}^{i-1}(r \cap w_j)$ can have at most three connected components. Thus the complement of $\bigcup_{j=1}^{i-1}(r \cap w_j)$ in r, i.e., $r \cap o_{i-1}$, has at most two connected components. This in turn implies that $c_i \cap o_{i-1}$ has only a constant number of connected components, and hence $|\mathcal{A}_i| - |\mathcal{A}_{i-1}| = O(1)$. Finally, since $|\mathcal{A}_0|$ is $O(1)$ and $m = O(n)$, we immediately have $|\mathcal{A}| = |\mathcal{A}_m| = O(m) = O(n)$. □

6.2.2 Preprocessing and query algorithms

In this section, we first propose a sub-optimal incremental algorithm that is able to construct the wedge subdivision in $O(n \log^2 n)$ time. (In Section 6.2.3 we improve this to $O(n \log n)$.) We use an augmented balanced search tree, $\mathcal{D}$, as the underlying data structure to maintain the upper envelope of o_i after we insert each wedge w_i into the dual space. (The upper-envelope is x-monotone.) Recall that o_i is the outer face of $\mathcal{P}_i$. Here, each node in $\mathcal{D}$ represents a segment (or an infinite ray) on o_i. Specifically, we store in each node, p, the following fields:

1) u and v, indicating the line $y = ux + v$ that goes through the segment represented by p;

2) x_1 and x_2, where $x_1 < x_2$, indicating the range of the segment in the x-dimension;

3) w, the wedge corresponding to the segment represented by p.

It is clear that any vertical line will intersect the upper envelope exactly once, which naturally gives us a total order among all the segments on it. Therefore, any arbitrary number between x_1 and x_2 suffices to act as the key for comparison. With $\mathcal{D}$ in hand, we can efficiently tell, in $O(\log n)$ time, whether a point in the dual space is above or below the envelope. In addition, we define the following convenient helper functions,

which will be used repeatedly as black boxes.

1) *remove*$(low, high)$: This method cuts off the portion of the upper envelope whose x-coordinate is in the range $(low, high)$. This function will remove several existing segments from $\mathcal{D}$ and insert at most two back into $\mathcal{D}$. Therefore, it has $O(\log n)$ amortized-runtime if we charge each segment removal to the corresponding insertion.

2) *insert*$(\ell, low, high)$: This method inserts into $\mathcal{D}$ a segment (or a ray), whose underlying line is ℓ and range on the x-axis is $[low, high]$. We guarantee that the structure maintained by $\mathcal{D}$ remains x-monotone after the insertion. Clearly, this function takes $O(\log n)$ time.

We initialize $\mathcal{D}$ as a single root representing the horizontal line $y = \infty$. We then sort $w_1, \ldots, w_m$ by non-decreasing order of their lengths (i.e., the length of the corresponding segment in the primal plane) and insert them into $\mathcal{D}$ one by one. When a new wedge, $w_i = (a_i^*, b_i^*)$, comes in, we show how to handle all the possible cases separately according to the proof of Theorem 6.4. Again, for simplicity, we only consider inserting the left ray, r, of w_i, and recall that the finite end-point of r is r_0. We also compute the line, l, that passes through a_i and is perpendicular to $a_i b_i$, and denote its dual point as l^*. Now, consider the following two cases.

Case 1: $l^* \notin r$**.** Choose an auxiliary point $r_{-\infty}$, which lies on r and has sufficiently small x-coordinate, to represent the infinite part of r. We then have the following four sub-cases depending on whether r_0 and $r_{-\infty}$ are above and/or below $\mathcal{D}$.

Case 1a: Both of r_0 and $r_{-\infty}$ are below $\mathcal{D}$, indicating that $r \cap w_j = \emptyset$ for all $j < i$. (Note that if $r \cap w_j \neq \emptyset$ it must contain either r_0 or $r_{-\infty}$ since $l^* \notin r$.) In this case, we simply do *remove*$(-\infty, r_0.x)$ followed by *insert*$(r, -\infty, r_0.x)$.

Case 1b: r_0 is below $\mathcal{D}$ and $r_{-\infty}$ is above. This means that if $r \cap w_j \neq \emptyset$ it must contain $r_{-\infty}$ and can never contain r_0. Thus, r has a unique intersection, γ, with $\mathcal{D}$, such that the infinite ray $\gamma r_{-\infty}$ is above $\mathcal{D}$ and the segment $\overline{r_0\gamma}$ is below. With this invariant in hand, we can identify the segment in $\mathcal{D}$ that intersects with r, and hence γ, via binary search on those x-coordinates in the range $(\infty, r_0.x]$. Once γ is found, we perform *remove*$(\gamma.x, r_0.x)$ followed by *insert*$(r, \gamma.x, r_0.x)$.

Case 1c: r_0 is above $\mathcal{D}$ and $r_{-\infty}$ is below. This case is symmetric to Case 1b and is thus omitted.

Case 1d: Both r_0 and $r_{-\infty}$ are above $\mathcal{D}$. This is a tricky case since ray r may be

entirely above $\mathcal{D}$, or it will intersect $\mathcal{D}$ twice, contributing to a new piece of interval on the envelope. We first assume the latter happens. Let α and β be the two intersections and assume α is to the left of β. Then, we claim that ray $r_{-\infty}\alpha$ and segment $\overline{\beta r_0}$ are above $\mathcal{D}$, and segment $\overline{\alpha\beta}$ is below. Again, binary search on x-values can be applied to compute α and β, but with a more careful invariant: the underlying wedge with respect to the segment cut by the left boundary always contains the infinite part of r, i.e., $r_{-\infty}$ and the underlying wedge cut by the right boundary always contains r_0. Once we find an x-coordinate at which r is below $\mathcal{D}$, we terminate the search and solve two subproblems (one similar to Case 1b and the other similar to Case 1c) to identify α and β, respectively; otherwise, we adjust one of the boundaries accordingly and continue the search. Eventually, the binary search either successfully finds α and β or fails due to being out of range. If α and β exist, we maintain $\mathcal{D}$ by calling $remove(\alpha.x, \beta.x)$ and $insert(r, \alpha.x, \beta.x)$; otherwise, it means r is completely above $\mathcal{D}$, and therefore, we should leave $\mathcal{D}$ unchanged.

Remark. We note that doing binary searches on the above x-coordinates is not as straightforward as searching on a conventional static array because some x-coordinates (i.e., $\alpha.x$, $\beta.x$, and $\gamma.x$) are not known beforehand and thus must be computed and maintained dynamically. Therefore, a dynamic data structure must be applied here, which unavoidably introduces an extra $O(\log m)$ factor. Formally, we use an *order statistic tree* [24], $\mathcal{OS}$, to maintain all the x-coordinates as well as their ranks. To do a binary search for values in $\mathcal{OS}$ ranging from $[x_{low}, x_{high}]$, we instead do a binary search on their ranks. It is clear that each binary search takes at most $O(\log(rank(x_{high}) - rank(x_{low}) + 1)) = O(\log m)$ steps. In each step, we first do an inverse query on $\mathcal{OS}$ to retrieve the x-value with the middle rank and then query $\mathcal{D}$ with it to decide whether we should shift the left or the right boundary. Both queries can be answered in $O(\log m)$ time, and therefore, each binary search takes $O(\log^2 m)$ time.

Case 2: $l^* \in r$. We first check whether l^* is above or below $\mathcal{D}$. If below, none of $r \cap w_j$, $j < i$, contains l^* and it essentially boils down to Case 1. Otherwise, l^* breaks the problem into two subproblems, i.e., inserting ray $l^* r_{-\infty}$ and segment $\overline{l^* r_0}$, respectively. We only show how to handle the latter. If r_0 is below $\mathcal{D}$, $\overline{l^* r_0}$ will have a unique intersection, γ, with $\mathcal{D}$ such that segment $\overline{l^*\gamma}$ is above and segment $\overline{\gamma r_0}$ is below the envelope; this boils down to Case 1b. On the other hand, if r_0 is above $\mathcal{D}$, we have

a case similar to Case 1d.

Nevertheless, either case involves $O(1)$ number of tree traversals, *remove*/*insert* operations, and binary searches. Therefore, inserting $w_1, \ldots, w_m$ into $\mathcal{D}$ in total takes $O(m(\log m + \log^2 m)) = O(m \log^2 m) = O(n \log^2 n)$ time. To construct the entire subdivision, one can explicitly save all the segments (and their corresponding wedges) generated throughout the entire algorithm and build a point location structure [52] to support logarithmic-time query. There are at most $O(n)$ segments according to Lemma 6.4, resulting in an $O(n \log n)$ (resp. $O(n)$) overhead in time (resp. space). Therefore, the total preprocessing time is $O(n \log^2 n)$ and we only use linear space.

6.2.3 The refinement

The runtime of each binary search mentioned above takes $O(\log^2 n)$ instead of $O(\log n)$ because $\mathcal{OS}$, as a tree structure, does not offer any constant-time random accessor. In this section, we show how to eliminate the $O(\log n)$ factor overhead and hence improve the overall runtime to $O(n \log n)$. For brevity, we only show how to speed up the binary search in Case 1b. Case 1c is completely symmetric, and Case 1d can be handled in a similar way with more care.

We augment each node p of $\mathcal{D}$ with one more field, $p.max$, indicating the rightmost segment in the subtree rooted at p. Given a binary search range $R = [x_{low}, x_{high}]$, we traverse $\mathcal{D}$ and collect $O(\log m)$ canonical nodes as well as the single nodes on the paths, whose range is completely contained in R. Name these nodes $c_1, \ldots, c_s$, where $s = O(\log m)$. Note that these canonical nodes are naturally ordered from left to the right due to the binary search tree property. We scan from c_1 to c_s. At the i-th iteration, if $c_i.max$ intersects with r, we can directly compute γ and we are done. If $c_i.max$ is below r, we skip c_i and proceed to the next canonical node c_{i+1}. If $c_i.max$ is above r, the segment that intersects with r must reside in c_i and we can find it by a binary tree traversal. We first check whether the root of c_i intersects with r. If so, we are done. Otherwise, the root must be either below or above r. If it is below r, we proceed to its right child and repeat the same procedure; else, we recursively check its left child instead.

The refined "binary search" takes only $O(\log m) = O(\log n)$ time since there are $O(\log m)$ canonical nodes to check and the height of any node is always bounded by

$O(\log m)$.

Finally, we note that the duality transform does not handle halfplanes whose boundary lines are vertical. But these can be trivially handled in $O(n)$ space and $O(\log n)$ query time. (E.g., by separate preprocessing or by slight rotation.) As such, we conclude the following result.

Theorem 6.7. *Given a set S of n points in $\mathbb{R}^2$ together with its $O(n)$ candidate pairs, one can build an $O(n)$-space data structure in $O(n \log n)$ time such that each halfplane query can be answered in $O(\log n)$ time.*

6.3 Radius-fixed disc query

In this section, we investigate the range closest pair reporting problem for a set, S, of points in the plane, where the query range Q is a disc of some *fixed* radius. Formally, let r be the radius of Q, where r is pre-determined and can be treated as a constant. Furthermore, we say Q is a *long* query if the shortest pair-wise distance in Q is no smaller than r/α for some user-specified positive integer constant α; otherwise, Q is a *short* query. In the rest of this section, we show how to correctly and efficiently answer a query depending on whether it is long or short.

6.3.1 Handling long queries

When Q is long, it is important to observe that there is only a constant number of points in Q, i.e., $|S \cap Q| = O(1)$. Indeed, we can always cover Q by a $2\alpha \times 2\alpha$ grid of squares of size $\frac{r}{\alpha} \times \frac{r}{\alpha}$; see Figure 6.6. By the pigeon-hole principle, there can be at most 4 points (from $S \cap Q$) in each square, which proves that $|S \cap Q| \leq 16\alpha^2 = O(1)$. Therefore, we can build on S a circular range reporting structure [4] so that given any long query Q we first report all points in Q and then find the closest pair by brute-force. Such a data structure takes $O(n)$ space and answers each query in $O(\log n + \alpha^2) = O(\log n)$ time.

6.3.2 Handling short queries

Unlike the previous case, the number of points in $S \cap Q$ can be large when Q is short, so the same method does not apply. Instead, we carefully bound the number of candidates

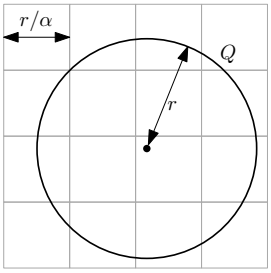


Figure 6.6: A $2\alpha \times 2\alpha$ grid covering Q

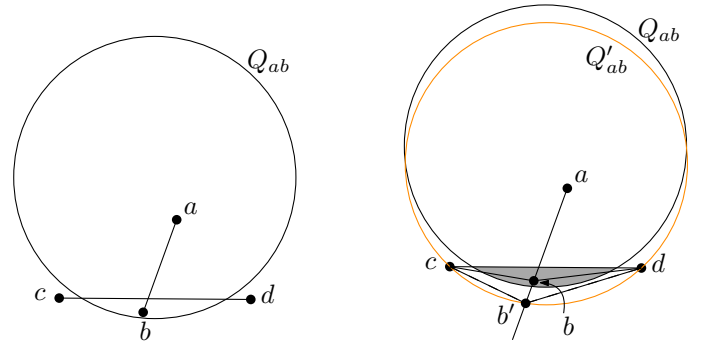
pairs that belong to some short query and build a data structure on those to answer the query. Formally, let $\mathcal{C}'_S = \{(a,b) \in \mathcal{C}_S : \text{dist}(a,b) < r/\alpha\}$. We will prove that $|\mathcal{C}'_S| = O(n)$.

Lemma 6.5. *If we treat each candidate pair in $\mathcal{C}'_S$ as a line segment, then no two segments intersect properly. Thus, $|\mathcal{C}'_S| = O(n)$.*

Proof. For a contradiction, assume the assertion is false. Then, there exist candidate pairs $(a,b) \in \mathcal{C}'_S$ and $(c,d) \in \mathcal{C}'_S$, where $\text{dist}(a,b) < r/\alpha$, $\text{dist}(c,d) < r/\alpha$, and $\overline{ab}$ properly intersects $\overline{cd}$. Choose the constant α sufficiently large so that $\text{dist}(a,b) \ll r$ and $\text{dist}(c,d) \ll r$. Let Q_{ab} (resp. Q_{cd}) be *any* disc of radius r that has (a,b) (resp. (c,d)) as the closest pair in it. Then, it is impossible to have both of Q_{ab} and Q_{cd} contain at least three points among a, b, c, d; the argument is similar to the one in Lemma 6.4 when Q is a halfplane. (See also [2].) W.l.o.g., assume Q_{ab} contains a and b, but neither c nor d; see Figure 6.7a. We then assert that Q_{cd} must contain b. This is easy to verify if the centers of Q_{ab} and Q_{cd} are on opposite sides of the line passing through $\overline{cd}$. We elaborate more in the following when the two centers lie on the same side.

Since both $\overline{ab}$ and $\overline{cd}$ are sufficiently short, we can always move Q_{ab} to a new position, named Q'_{ab}, such that a and b are in Q'_{ab} and both c and d lie exactly on the boundary of Q'_{ab}; see Figure 6.7b. Then we claim that b must be contained in Q'_{ab}. To see this, note that the region of Q_{ab} in which b resides (shaded in gray) is completely contained in Q'_{ab} as the two discs have the same size. It follows that to ensure that Q_{cd} contains both c and d, Q_{cd} must contain the region of Q'_{ab} bounded by $\overline{cd}$ (the one not containing a) and hence must contain b.

Once it is confirmed that $b \in Q_{cd}$, we immediately derive a contradiction by arguing that (c, d) can never be the candidate pair of Q_{cd} as both $\overline{bc}$ and $\overline{bd}$ are strictly shorter. Indeed, we extend the ray ab so that it intersects Q'_{ab} at b'. Since $\text{dist}(c, d) = r/\alpha \ll r$, $\angle cb'd$ must be greater than $\pi/2$, indicating that $\angle cbd$ is also be greater than $\pi/2$. Hence, $\overline{cd}$ must be the longest edge in the obtuse triangle Δbcd. □



(a) An example of Q_{ab}, where $c \notin Q_{ab}$ and $d \notin Q_{ab}$.

(b) Illustrating Q'_{ab}, which helps to show $b \in Q_{cd}$.

Figure 6.7: Illustrating the proof of Lemma 6.5.

Next, we show how to build the query structure for short queries. We begin with some definitions and observations. For each $\theta \in \mathcal{C}'_S$, let $\theta = (a, b)$ where $a, b \in S$. Define $\ell_\theta = c_a \cap c_b$, where c_a and c_b are the discs of radius r centered at a and b, respectively. We call ℓ_θ the *lune* of θ. (Note that our definition of a lune is slightly different from the standard definition in [27].) It is clear that θ can be reported only if the center of Q lies inside ℓ_θ. In addition, we treat the following statements as equivalent: (i) $\text{dist}(a, b) < r/\alpha$; (ii) $\theta = (a, b)$ is short; (iii) ℓ_θ is *fat*, as it is close in size to a disc of radius r. Finally,

(i) Let ℓ, ℓ_1, ℓ_2 be any fat lunes;

(ii) Let d be any disc with radius r;

(iii) Let $\partial\ell$, $\partial\ell_1$, $\partial\ell_2$, ∂d denote the corresponding shape boundaries;

(iv) Let ℓ^+, ℓ^- be the two extreme points of ℓ, where ℓ^+ is above ℓ^-. Similarly, define ℓ_1^+, ℓ_1^-, ℓ_2^+, ℓ_2^-.

Observation 6.1. *The boundary of a fat lune intersects at most twice with the boundary of a disc of radius r, i.e., $|\partial\ell \cap \partial d| \leq 2$.*

Proof. We break $\partial\ell$ into two circular arcs, φ_1 and φ_2, both of radius r. If ∂d intersects only one arc, then the statement is clearly true as two circles have at most two intersections. Now, assume ∂d intersects both arcs. (See Figure 6.8.) We show $|\partial d \cap \varphi_1| = |\partial d \cap \varphi_2| = 1$, and thus the statement holds. To see this, we denote by Φ_1 the disc (with radius r) that has φ_1 on its boundary and assume that d and Φ_1 intersect at α and β. We can show that if $\alpha \in \varphi_1$, then $\beta \notin \varphi_1$. Let $\widehat{\alpha\beta}$ be the arc on d that is inside Φ_1. Since $\ell \subset \Phi_1$, the other arc φ_2 must intersect with $\widehat{\alpha\beta}$ at some point γ. Now, $\widehat{\alpha\gamma}$ is inside the lune and $\widehat{\gamma\beta}$ is outside, which shows that $\beta \notin \varphi_1$. This proves that $|\partial d \cap \varphi_1| = 1$. Similarly, we have $|\partial d \cap \varphi_2| = 1$, and thus $|\partial\ell \cap \partial d| \leq 2$. □

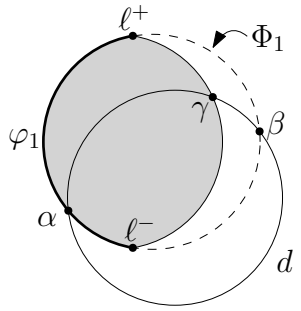


Figure 6.8: Illustrating Observation 6.1, where lune ℓ is shaded gray.

Observation 6.2. *If an extreme point of a fat lune lies in another fat lune, the boundaries of the lunes cross at most twice. That is, w.l.o.g., if $\ell_2^+ \in \ell_1$ or $\ell_2^- \in \ell_1$, then $|\partial\ell_1 \cap \partial\ell_2| \leq 2$.*

Proof. First, we note that if both $\ell_2^+ \in \ell_1$ and $\ell_2^- \in \ell_1$, then we have $\ell_2 \subset \ell_1$ and thus $|\partial\ell_1 \cap \partial\ell_2| = 0$. This is not difficult to see since $\partial\ell_1$ and $\partial\ell_2$ are generated by discs of the same radius. Then, w.l.o.g., assume $\ell_2^+ \in \ell_1$ and $\ell_2^- \notin \ell_1$. (See Figure 6.9.) In this case, we show that each branch of $\partial\ell_2$ will intersect with $\partial\ell_1$ exactly once, and, therefore, $|\partial\ell_1 \cap \partial\ell_2| = 2$. Fix a branch of $\partial\ell_2$ and consider the disc (of radius r) that generates it. Name the branch φ. By Observation 6.1, $\partial\ell_1$ can intersect the boundary of

the disc at most twice. In fact, since ℓ_2^+ is inside ℓ_1, there are exactly two intersections, which we call α and β. W.l.o.g., if $\alpha \in \varphi$, then we have $\beta \notin \varphi$. Indeed, since ℓ_2^+ is an endpoint of φ, it follows that $\widehat{\alpha\ell_2^+} \subset \varphi$ and $\widehat{\ell_2^+\beta} \cap \varphi = \{\ell_2^+\}$. As such, $|\varphi \cap \partial\ell_1| = 1$. Similarly, $|\varphi \cap \partial\ell_2| = 1$. □

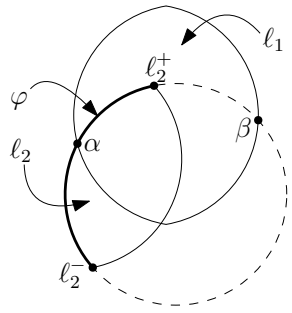


Figure 6.9: Illustrating Observation 6.2, where the fixed branch φ of ℓ_2 is shown bold.

Observation 6.3. *The boundaries of two fat lunes cross at most twice, i.e.,* $|\partial\ell_1 \cap \partial\ell_2| \leq 2$.

Proof. Let (a_1, b_1) and (a_2, b_2) be the (short) candidate pairs of ℓ_1 and ℓ_2, respectively. By Lemma 6.5, segment $\overline{a_1b_1}$ and $\overline{a_2b_2}$ cannot cross. It follows that either both a_1 and b_1 lie completely on one side of the line passing through $\overline{a_2b_2}$, or vice versa. W.l.o.g., assume a_1 and b_1 lie *above* $\overline{a_2b_2}$, and we draw an auxiliary disc, Φ, centered at ℓ_2^+ with radius r. For convenience of analysis, assume $\overline{a_2b_2}$ is parallel to the x-axis, and hence $\overline{\ell_2^+\ell_2^-}$ is vertical. We then complete the proof by separately considering the following four cases.

1. At least one of a_1 and b_1 is strictly above Φ. In this case, even ℓ_2^+ will not be contained in ℓ_1, and therefore, $|\partial\ell_1 \cap \partial\ell_2| = 0$.

2. At least one of a_1 and b_1 is out of both Φ and ℓ_2. It is not difficult to verify that ℓ_1 can only intersect with one branch of $\partial\ell_2$. By Observation 6.1, $\partial\ell_1$ can have at most two intersections even with the circle generating that branch, so with $\partial\ell_2$.

3. At least one of a_1 and b_1 is inside ℓ_2. Then Observation 6.2 directly applies and the statement holds.

4. The only case remaining is that both a_1 and b_1 are contained in $\Phi \setminus \ell_2$. Since $a_1, b_1 \in \Phi$, $\text{dist}(a_1, \ell_2^+) \leq r$ and $\text{dist}(b_1, \ell_2^+) \leq r$, and thus $\ell_2^+ \in \ell_1$, i.e., one endpoint of ℓ_2 lies inside ℓ_1. By applying Observation 6.2 we immediately conclude the statement.

□

Observation 6.3 shows that fat lunes belong to the family of *pseudo-discs* [13]. Therefore, according to [45], the union of a set of fat lunes has linear complexity. One can then build on the union a planar point location data structure so that we can quickly check whether a given point lies inside the union interior. (The standard point location structure for line segment, e.g. [52], can be generalized to circular arcs in a straightforward way. We omit the details.) As such, we conclude the following result.

Lemma 6.6. *Let $\mathcal{L} = \{\ell_{\theta_1}, \ldots, \ell_{\theta_k}\}$ be a set of fat lunes w.r.t. short candidate pairs $\theta_1, \ldots, \theta_k$. There exists a data structure occupying $O(k)$ space such that for any query point $q \in \mathbb{R}^2$ one can report whether $q \in \bigcup \mathcal{L}$ in $O(\log k)$ time.*

With Lemma 6.6 in hand, we can finally propose our algorithm and data structure to answer short queries efficiently. We build a balanced binary tree $\mathcal{T}$ in a bottom-up fashion where the leaves are the θ's from $\mathcal{C}'_S$, sorted in increasing order of their lengths. (Recall that the length of $\theta = (a, b)$ is $\text{dist}(a, b)$.) For each internal node $u \in \mathcal{T}$, let $\mathcal{L}_u$ be the collection of lunes corresponding to all the leaves in the subtree rooted at u. We then store in u a secondary data structure on $\mathcal{L}_u$ for quickly reporting whether a given point lies in $\bigcup \mathcal{L}_u$, as we described in Lemma 6.6. Clearly, the overall space is $O(n \log n)$ because each level of $\mathcal{T}$ uses $O(n)$ space and there are in total $O(\log n)$ levels.

To answer a (short) query Q, we set u to be the root of $\mathcal{T}$ and proceed as follows until u becomes a leaf. When u is an internal node, let v and w be its left and right child, respectively. We then check by querying the data structure stored in v whether the center of Q, which we denote by q, lies in $\bigcup \mathcal{L}_v$. If yes, we repeat this process by setting u to v. Otherwise, set u to w and proceed further. Once u becomes a leaf we do a final check to see whether q lies in the lune of u. If so, we report the candidate pair w.r.t. u as the output; otherwise, Q contains no pair at all. It is easy to check that, during the descent in $\mathcal{T}$, the secondary data structure is queried exactly once per level of $\mathcal{T}$, resulting in an $O(\log n \cdot \log n) = O(\log^2 n)$ query time.

6.3.3 Putting both cases together

Given a general query disc Q with a fixed radius r, we first assume it is long and query the circular range search data structure mentioned in Section 6.3.1. We keep reporting points that are contained in Q until we have exhausted all of them or have encountered more than $16\alpha^2$ points. This step takes $O(\log n)$ time. If the former case applies, we simply find the closest pair by brute-force and we are done; otherwise, Q must be short. We then query the data structure described in Section 6.3.2 and find the answer in $O(\log^2 n)$ time. As such, we conclude the following theorem.

Theorem 6.8. *A set, S, of n points in $\mathbb{R}^2$ can be preprocessed into a data structure occupying $O(n \log n)$ space such that, for any radius-fixed disc Q, the closest pair in $S \cap Q$ can be reported in $O(\log^2 n)$ time.*

6.4 A general approximation framework

In real-world applications, data is often imprecise due to noise or sensing limitations. Thus each data point can exist anywhere in a disc centered at the presumed location of the data point. Therefore, in a range closest pair query, input points that are sufficiently close to the query boundary might not actually be in the query range, so the closest pair in the query range may not be the true closest pair. Thus, it is natural to shrink the query region suitably and use the closest pair in the shrunken region as the baseline. That is, we want to output a pair in the query range whose distance is no more than the minimum one generated from the shrunken region. (Note, however, that this approximation does not necessarily guarantee an upper bound on the approximation ratio.) We first define formally the notion of shrinkage and then propose a generic approximation method.

Definition 6.2. *Given a closed region R, the δ-shrinkage of R is the following subregion*

$$\{x \in R : \inf_{y \notin R} \operatorname{dist}(x, y) \geq \delta\}.$$

That is, the δ-shrinkage of R consists of all points x such that the open disc of radius δ centered at x is contained in R.

Definition 6.3. *Given a closed region R, the radius of R is defined as*

$$\mathrm{rad}(R) = \sup_{x \in R} \inf_{y \notin R} \mathrm{dist}(x, y).$$

That is, the radius of R is the radius of the largest open inscribed disc of R.

Given a set S of n points in $\mathbb{R}^2$, a query Q, and a positive real ε, an *ε-approximation* returns some pair (a, b) in $Q \cap S$ that is no farther apart the closest pair in Q_ε, where Q_ε is defined as the $(\mathrm{rad}(Q)\varepsilon)$-shrinkage of Q. Our solution to this problem is based on a general framework that uses two fundamental structures in computational geometry, namely, a 2-level range reporting structure (RR) and a 2-level range minimum query (RMQ) structure. We build RR and RMQ on the set S during the preprocessing phase. Specifically, in RMQ, the weight of each point of S is equal to the shortest distance from this point to any other point of S. We assume that RR and RMQ can answer each query in $O(f(n) + k)$ and $O(g(n))$ time, respectively, where $f(n)$ and $g(n)$ are some functions of n, and k denotes the output size. With RR and RMQ ready, we give our approximation query algorithm as Algorithm 6, where for simplicity we assume that there are at least two points in Q_ε. If not, we return any pair in Q because the baseline is undefined. Also, to guarantee the performance, we require that the given query Q is $O(1)$-fat, where the fatness of Q is the ratio of the radius of the smallest circumscribed circle to the radius of the largest inscribed circle of Q.

Algorithm 6 Approximation query algorithm

```
1: function ε-Approximation(S, RR, RMQ, Q, ε) ▷ Assume that there are at least two points in Q_ε.
2:     Query RMQ with Q_ε and retrieve the point in Q_ε ∩ S with the minimum weight.
3:     Let (a, b) be the pair corresponding to the minimum-weight point.
4:     if dist(a, b) ≤ rad(Q)ε then
5:         return (a, b)
6:     else
7:         Query RR with Q_ε and report all points in Q_ε ∩ S.
8:         return the closest pair among these points using a standard single-shot closest pair algorithm.
9:     end if
10: end function
```

Correctness and runtime analysis:

The correctness of Algorithm 6 is trivial if Line 8 is triggered as we are solving the problem directly for Q_ε. We now show that the correctness also holds for Line 5. First, both of a and b are in Q by Definition 6.2 and 6.3, implying that (a, b) is a pair in Q. Next, it is easy to verify that $\text{dist}(a, b) \leq \text{dist}(\hat{a}, \hat{b})$, where $(\hat{a}, \hat{b})$ is the closest pair in Q_ε. Therefore, pair (a, b) is a valid approximation.

To bound the query time of Algorithm 6, we only need to analyze Line 2, 7, and 8. The runtime of Line 2 is clearly $O(g(n))$. It then suffices to analyze the runtime for Line 7 and 8. Indeed, we argue that there can be at most $O(1/\varepsilon^2)$ points in $Q_\varepsilon \cap S$ under this case, similar to the analysis in Section 6.3.1. Formally, we see that any pairwise distance in Q_ε is greater than $\text{rad}(Q)\varepsilon$ since $\text{dist}(a, b) > \text{rad}(Q)\varepsilon$. Then, by the pigeonhole principle, there are at most four points in the intersection between Q_ε and any $\text{rad}(Q)\varepsilon \times \text{rad}(Q)\varepsilon$ square. Also, since Q is fat, it can be verified that we can cover Q (and hence Q_ε) by $\left(\alpha \frac{\text{rad}(Q)}{\text{rad}(Q)\varepsilon}\right)^2 = O(1/\varepsilon^2)$ squares, where α is a constant depending on the fatness. Therefore, Line 7 takes $O(f(n)+1/\varepsilon^2))$ time, and Line 8 takes $O((1/\varepsilon^2)\log(1/\varepsilon))$ time. The total runtime is thus $O(f(n) + g(n) + (1/\varepsilon^2)\log(1/\varepsilon))$.

Applications:

We provide several applications of our general framework by adapting suitable black boxes for RR and RMQ when the query family consists of:

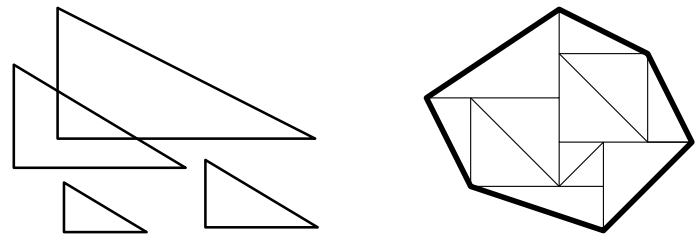
1. **Discs**. We use the structure in [4] for RR, which uses linear space and answers each query in $O(\log n + k)$ time. For RMQ, we apply the results in [51] that can answer each query in $O((\log n+1)\log n)$ time using $O(n^{1+\zeta})$ space, where ζ is any positive real. Thus, the overall query time is $O(\log^2 n + (1/\varepsilon^2)\log(1/\varepsilon))$ and the space used is $O(n^{1+\zeta})$.

2. **Fat axes-aligned rectangles**. We build standard 2D range trees (with fractional cascading) for RR and RMQ that answer RR and RMQ queries in $O(\log n + k)$ and $O(\log n)$ time, respectively. Therefore, the total query time is $O(\log n + (1/\varepsilon^2)\log(1/\varepsilon))$. The space occupied is $O(n \log n)$.

3. **Any translated and/or scaled copy of a fat right triangle (with two**

edges parallel to the coordinate axes); See Figure 6.10a for an example. Assume that the triangle hypotenuse has a slope of ℓ. Then, it suffices to answer RR or RMQ by creating a three-level range tree. That is, we build the levels w.r.t. the x-axis first, the y-axis, and finally the line $y = \ell x$, with fractional cascading applied at the last level. Thus, the total space occupied is $O(n \log^2 n)$ and the query time is $O(\log^2 n + (1/\varepsilon^2) \log(1/\varepsilon))$.

4. **Any translated and/or scaled copy of a fat convex shape (with constant complexity)**. Note that both the range reporting and the range minimum query are decomposable. Also, it is easy to see that any convex shape of this family can be partitioned into $O(1)$ right triangles with two edges parallel to the coordinate axes; see Figure 6.10b. For each family of right triangles in this partition, we build 3-level range trees for doing RR and RMQ. Given query Q, we take Q_ε and decompose it using the same aforementioned partition. Clearly, the answer for RMQ (resp. RR) w.r.t. Q_ε can be found by doing RMQ (resp. RR) w.r.t. each triangle in the partition. Therefore we can apply Algorithm 6. The query time is $O(\log^2 n + (1/\varepsilon^2) \log(1/\varepsilon))$ and the space is $O(n \log^2 n)$ as Q has constant complexity. Note that it is only necessary that Q (hence Q_ε) be fat; the triangles in the partition themselves need not be fat.



(a) A family of right triangles (with two edges parallel to both axes) under translation and scaling

(b) A fat convex shape with $O(1)$ complexity and its partition using right triangles

Figure 6.10: Illustrating cases 3 and 4.

6.5 Answering offline range min-gap query

In Section 6.1.5, we mentioned two approaches to the offline range min-gap query via Mo's algorithm [1]. We elaborate on these here. We first design a data structure $\mathcal{D}$ that supports the following three operations:
(1) insert a real into $\mathcal{D}$;
(2) remove a real from $\mathcal{D}$;
(3) output the min-gap of all the reals in $\mathcal{D}$.
To implement all the operations efficiently, one can augment a balanced binary search tree with three fields: max, min, and min-gap, representing the maximum, the minimum, and the min-gap in each subtree. Since all the three fields can be properly maintained in $O(1)$ time for each node, $\mathcal{D}$ can perform each of the three operations in $O(\log n)$ time.

Now, assume there are in total m range min-gap queries to answer, and each query is of the form $[l_i, r_i]$, where $1 \leq l_i < r_i \leq n$. According to Mo's algorithm, we need to order these intervals by $\lfloor l_i/\sqrt{n} \rfloor$ first, then by r_i. Since l_i, r_i, and $\lfloor l_i/\sqrt{n} \rfloor$ are all integers in the range of $[1, n]$, counting sort applies and hence takes $O(m + n)$ time. We then initialize $\mathcal{D}$ by inserting the reals in the subarray $A[l_1..r_1]$ and answer the first query. This step takes $O(n \log n)$ time. As we move from the i-th query to the $(i+1)$-th query, we need to maintain $\mathcal{D}$ by inserting A_j's for $j \in [l_{i+1}, r_{i+1}] \setminus [l_i, r_i]$ and removing A_j's for $j \in [l_i, r_i] \setminus [l_{i+1}, r_{i+1}]$. Then, $\mathcal{D}$ is ready to answer the $(i + 1)$-th query, and we repeat this process until all m queries have been reported. It can be shown that the total number of insertions and removals is bounded by $((m/\sqrt{n})\sqrt{n} + n)\sqrt{n} = (m + n)\sqrt{n}$. Thus, it takes $O((m + n)\sqrt{n} \log n)$ time to answer all the m queries.

As we see above, Mo's algorithm judiciously determines an order so that one does not need to change too many elements when switching between consecutive queries. In fact, we can often do better than that. If we treat each query $[l_i, r_i]$ as a point (l_i, r_i) in $\mathbb{R}^2$, the cost of moving from the i-th to the $(i + 1)$-th query is no more than the L_1-distance between (l_i, r_i) and (l_{i+1}, r_{i+1}). Therefore, to reduce the overall cost, we can map all the queries to points in the plane and compute their rectilinear minimum spanning tree [39, 64]. With the MST in hand, we do a Euler tour (starting from any vertex) in the tree and then answer each query according to the vertex-order along the tour. It is easy to check that the total cost is no more than twice the total tree

length. The planar rectilinear minimum spanning can be computed in $O(n \log n)$ time, and computing the Euler tour takes linear time. So, it is generally a good idea to apply this optimization to achieve a better sequence than the one from Mo's algorithm.

On the other hand, it is worth mentioning that this approach cannot improve the worst-case performance. Consider the following example. We are given a $\sqrt{n} \times \sqrt{n}$ grid, and imagine we have roughly $(\sqrt{n} \times \sqrt{n})/2 = \Theta(n)$ queries, located at the center of each square from the upper triangle of the grid. Since the L_1-distance between any two grid centers is at least $\sqrt{n}$, the total length of the MST is $\Omega(n\sqrt{n})$ as there are $\Theta(n)$ edges. This example shows the tightness of Mo's algorithm.

Chapter 7

Conclusion and future work

We summarize the contributions of this thesis and list some open problems for future work.

7.1 Summary of contributions

In Chapter 2, we investigated the preference top-k query problem, where one must preprocesses a dataset of points in $\mathbb{R}^d$ so that the user can efficiently retrieve the top-k candidates w.r.t. one's specific preference. We presented efficient algorithms in 2D and 3D and also considered two query variants, namely, range preference top-k query and preference top-k with fuzzy vectors. Furthermore, in Chapter 3, we proposed a new sampling-based approximation algorithm to answer the preference top-k query. We proved via theoretical analysis that in $\mathbb{R}^2$ the method samples only a small subset of the input while guaranteeing that the approximation error is within a user-specified tolerance. For $\mathbb{R}^3$ and $\mathbb{R}^4$ we provided experimental evidence for this claim.

In Chapter 4, we extended the concept of a line arrangement to the stochastic setting and investigated the most-likely k-topmost lines problem. We derived an upper-bound on the expected number of changes to the set of most-likely k-topmost lines, taken over the entire x-axis. We also showed, via a concrete example, the upper-bound can be quadratic in the worst-case even when $k = 1$. Moreover, we proposed an efficient algorithm to compute the most-likely k-topmost lines over the entire x-axis. Finally, we considered two related applications, namely, stochastic Voronoi Diagrams in $\mathbb{R}^1$ and

stochastic preference top-k queries in $\mathbb{R}^2$.

In Chapter 5, we generalized the idea of the stochastic Voronoi Diagram and its related problems from $\mathbb{R}^1$ to a general tree space. Specifically, we investigated two fundamental proximity problems under the stochastic setting, the closest-pair problem and nearest-neighbor search. For the former, we proposed the first algorithm for computing the ℓ-threshold probability and the expectation of the closest-pair distance of a realization of the stochastic input points. For the latter, we studied the k most-likely nearest-neighbor search (k-LNN) via a notion called the k most-likely Voronoi Diagram (k-LVD).

In Chapter 6, we further explored the proximity problems in query-retrieval mode and proposed efficient exact solutions to the range closest pair problem for queries such as a p-sided axes-aligned rectangle ($p = 2, 3, 4$), a halfplane, and a disc with fixed radius. We also presented a general approximation framework that is flexible enough to handle other query shapes. Some of our proofs (e.g., the number of candidate pairs for halfplane queries and radius-fixed discs (for short queries)) are of independent combinatorial interest.

7.2 Future work

We close this thesis by listing the following open problems.

1. In Chapter 3, the theoretical analysis of our approximation algorithm for preference top-k queries is only given in $\mathbb{R}^2$. It would be interesting to further generalize the analysis to higher dimensions.

2. In Chapter 4, we investigated the combinatorial complexity for a given set of stochastic lines. One direction for future work here is to study the problem in higher dimensions. Specifically, in $\mathbb{R}^d$, given n stochastic hyperplanes, we are interested in the most likely k-topmost hyperplanes with respect to the d-th dimension taken over all the points spanned by the first $d-1$ dimensions. As in Chapter 4, a subdivision can be computed in the subspace of the first $d-1$ dimensions such that any query point in a cell has the same set of the most likely k-topmost hyperplanes. However, the structure of the subdivision becomes subtle

when $d \geq 3$, and thus deriving an expected bound on its complexity becomes hard.

3. In Chapter 5, we showed that to compute efficiently two elementary statistics regarding the stochastic closest pair problem, namely, the ℓ-threshold probability and the expected closest pair distance. Symmetrically, it would be of interest to compute similar statistics with respect to the stochastic farthest pair problem.

4. In Chapter 6, we assume that all the candidate pairs are given beforehand during preprocessing for all our algorithms. It would be of interest to study how to identify these pairs efficiently.

5. Finally, it would be interesting to consider a problem that incorporates concepts and ideas from Chapters 5 and 6, i.e., the range closest pair problem in the stochastic setting where each point has an existential uncertainty. A question of interest then would be to determine the probability that the range closest pair has distance greater than some user-specified threshold. Though the single-shot problem has been proved to be #P-hard in [44], the hardness of the problem is unknown in the query-retrieval setting.

References

[1] `https://z.umn.edu/mosalgorithm/`.

[2] M. A. Abam, P. Carmi, M. Farshi, and M. Smid. On the power of the semi-separated pair decomposition. In *Workshop on Algorithms and Data Structures*, pages 1–12. Springer, 2009.

[3] P. Afshani, G. Brodal, and N. Zeh. Ordered and unordered top-k range reporting in large data sets. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms*, pages 390–400. SIAM, 2011.

[4] P. Afshani and T. M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 180–186. Society for Industrial and Applied Mathematics, 2009.

[5] P. Agarwal, B. Aronov, T. Chan, and M. Sharir. On levels in arrangements of lines, segments, planes, and triangles. *Discrete & Computational Geometry*, 19(3):315–331, 1998.

[6] P. Agarwal, B. Aronov, S. Har-Peled, J. Phillips, K. Yi, and W. Zhang. Nearest neighbor searching under uncertainty II. In *Proceedings of the 32nd ACM Symposium on Principles of Database Systems (PODS)*, pages 115–126. ACM, 2013.

[7] P. Agarwal, S.-W. Cheng, and K. Yi. Range searching on uncertain data. *ACM Transactions on Algorithms*, 8(4):43, 2012.

[8] P. Agarwal, M. de Berg, J. Matousek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi Diagrams. *SIAM Journal on Computing*, 27(3):654–667, 1998.

[9] P. Agarwal, A. Efrat, S. Sankararaman, and W. Zhang. Nearest-neighbor searching under uncertainty. In *Proceedings of the 31st ACM Symposium on Principles of Database Systems (PODS)*, pages 225–236. ACM, 2012.

[10] P. Agarwal, S. Har-Peled, S. Suri, H. Yıldız, and W. Zhang. Convex hulls under uncertainty. In *European Symposium on Algorithms (ESA)*, pages 37–48. Springer, 2014.

[11] P. Agarwal, N. Kumar, S. Sintos, and S. Suri. Range-max queries on uncertain data. In *Proceedings of the 35th ACM Symposium on Principles of Database Systems (PODS)*, pages 465–476. ACM, 2016.

[12] P. Agarwal and M. Sharir. Arrangements and their applications. In *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, 1998.

[13] P. K. Agarwal, J. Pach, and M. Sharir. State of the union (of geometric objects): A review. *Computational Geometry: Twenty Years Later. American Mathematical Society*, 2007.

[14] C. Aggarwal and P. Yu. A survey of uncertain data algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 21(5):609–623, 2009.

[15] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM Special Interest Group on Management of Data (SIGMOD)*, pages 539–550. ACM, 2003.

[16] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The International Journal on Very Large Data Bases (VLDB)*, 10(2-3):199–223, 2001.

[17] T. Chan. On levels in arrangements of surfaces in three dimensions. *Discrete & Computational Geometry*, 48(1):1–18, 2012.

[18] Y.-C. Chang, L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. Smith. The onion technique: indexing for linear optimization queries. In *Proceedings of the*

2000 ACM Special Interest Group on Management of Data (SIGMOD), volume 29, pages 391–402. ACM, 2000.

[19] S. Chaudhuri, G. Das, and V. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *Proceedings of the 2001 ACM Special Interest Group on Management of Data (SIGMOD)*, volume 30, pages 295–306. ACM, 2001.

[20] B. Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31:509–517, 1985.

[21] B. Chazelle and L. Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1:133–162, 1986.

[22] D. Chen, G.-Z. Sun, and N. Gong. Efficient approximate top-k query algorithm using cube index. In *Web Technologies and Applications*, pages 155–167. Springer, 2011.

[23] J. Chen and L. Feng. Efficient pruning algorithm for top-k ranking on dataset with value uncertainty. In *Proceedings of the 22nd ACM Conference on Information and Knowledge Management (CIKM)*, pages 2231–2236. ACM, 2013.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[25] H. Coxeter. *Introduction to geometry*. New York, London, 1961.

[26] N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: Diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

[27] M. de Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry*. Springer Verlag, 2nd edition, 2000.

[28] T. Dey. Improved bounds for planar k-sets and related problems. *Discrete & Computational Geometry*, 19(3):373–382, 1998.

[29] J. Driscoll, N. Sarnak, D. Sleator, and R. Tarjan. Making data structures persistent. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 109–121. ACM, 1986.

[30] P. Erdős, L. Lovász, A. Simmons, and E. Straus. Dissection graphs of planar point sets. *A Survey of Combinatorial Theory*, pages 139–149, 1973.

[31] R. Fagin. Combining fuzzy information from multiple systems. In *Proceedings of the 15th ACM Symposium on Principles of Database Systems (PODS)*, pages 216–226. ACM, 1996.

[32] R. Fagin. Fuzzy queries in multimedia database systems. In *Proceedings of the 17th ACM Symposium on Principles of Database Systems (PODS)*, pages 1–10. ACM, 1998.

[33] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.

[34] J. Fakcharoenphol, S. Rao, and K. Talwar. Approximating metrics by tree metrics. *ACM SIGACT News*, 35(2):60–70, 2004.

[35] M. Fink, J. Hershberger, N. Kumar, and S. Suri. Hyperplane separability and convexity of probabilistic point sets. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG)*. ACM, 2016.

[36] G. Frederickson. An optimal algorithm for selection in a min-heap. *Information and Computation*, 104(2):197–214, 1993.

[37] G. Frederickson and D. Johnson. The complexity of selection and ranking in $x + y$ and matrics with sorted columns. *Journal of Computer and System Science*, 24:197–208, 1982.

[38] T. Ge, S. Zdonik, and S. Madden. Top-k queries on uncertain data: on score distribution and typical answers. In *Proceedings of the 2009 ACM Special Interest Group on Management of Data (SIGMOD)*, pages 375–388. ACM, 2009.

[39] L. J. Guibas and J. Stolfi. On computing all north-east nearest neighbors in the L_1-metric. *Information Processing Letters*, 17(4):219–223, 1983.

[40] P. Gupta, R. Janardan, Y. Kumar, and M. Smid. Data structures for range-aggregate extent queries. *Journal of Computational Geometry*, 47(2, Part C):329 – 347, 2014.

[41] L. Huang and J. Li. Approximating the expected values for combinatorial optimization problems over stochastic points. In *International Colloquium on Automata, Languages, and Programming*, pages 910–921. Springer, 2015.

[42] I. Ilyas, G. Beskales, and M. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

[43] P. Kamousi, T. Chan, and S. Suri. Stochastic minimum spanning trees in Euclidean spaces. In *Proceedings of the 27th International Symposium on Computational Geometry (SoCG)*, pages 65–74. ACM, 2011.

[44] P. Kamousi, T. Chan, and S. Suri. Closest pair and the post office problem for stochastic points. *Computational Geometry*, 47(2):214–223, 2014.

[45] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete & Computational Geometry*, 1(1):59–71, 1986.

[46] N. Kumar, B. Raichel, S. Suri, and K. Verbeek. Most likely Voronoi Diagrams in higher dimensions. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 65. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[47] M. Löffler and M. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.

[48] L. Lovász. On the number of halving lines. *Ann. Univ. Sci. Budapest, Eötvös, Sec. Math*, 14:107–108, 1971.

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

[50] F. P. Preparata and M. Shamos. *Computational geometry: An introduction*. Springer, 1985.

[51] S. Rahul and R. Janardan. A general technique for top-k geometric intersection query problems. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):2859–2871, 2014.

[52] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, July 1986.

[53] M. Shamos and D. Hoey. Closest-point problems. In *16th Annual IEEE Symposium on Foundations of Computer Science*, pages 151–162, 1975.

[54] J. Shan, D. Zhang, and B. Salzberg. On spatial-range closest-pair query. In *International Symposium on Spatial and Temporal Databases*, pages 252–269. Springer, 2003.

[55] R. Sharathkumar and P. Gupta. Range-aggregate proximity queries. *Technical Report IIIT/TR/2007/80. IIIT Hyderabad, Telangana*, 500032, 2007.

[56] M. Sharir. On k-sets in arrangements of curves and surfaces. *Discrete & Computational Geometry*, 6(1):593–613, 1991.

[57] M. Smid. *Closest point problems in computational geometry.* Citeseer, 1995.

[58] S. Suri and K. Verbeek. On the most likely Voronoi Diagram and nearest neighbor searching. In *Proceedings of the 25th International Symposium on Algorithms and Computation (ISAAC)*, pages 338–350. Springer, 2014.

[59] S. Suri, K. Verbeek, and H. Yıldız. On the most likely convex hull of uncertain points. In *21st Annual European Symposium on Algorithms (ESA)*, pages 791–802. Springer, 2013.

[60] G. Tóth. Point sets with many k-sets. *Discrete & Computational Geometry*, 26(2):187–194, 2001.

[61] D. Xin, C. Chen, and J. Han. Towards robust indexing for ranked queries. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 235–246. VLDB Endowment, 2006.

[62] J. Xue, Y. Li, and R. Janardan. On the separability of stochastic geometric objects, with applications. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG)*. ACM, 2016.

[63] A. Yu, P. Agarwal, and J. Yang. Processing a large number of continuous preference top-k queries. In *Proceedings of the 2012 ACM Special Interest Group on Management of Data (SIGMOD)*, pages 397–408. ACM, 2012.

[64] H. Zhou, N. Shenoy, and W. Nicholls. Efficient minimum spanning tree construction without Delaunay triangulation. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, pages 192–197. ACM, 2001.