Range Searching on Uncertain Data*

Pankaj K. Agarwal
Duke University
Durham, NC, USA
pankaj@cs.duke.edu

Siu-Wing Cheng
HKUST
Hong Kong, China
scheng@cse.ust.hk

Yufei Tao
CUHK
Hong Kong, China
taoyf@cse.cuhk.edu.hk

Ke Yi
HKUST
Hong Kong, China
yike@cse.ust.hk

Abstract

Querying uncertain data has emerged as an important problem in data management due to the imprecise nature of many measurement data. In this paper we study answering range queries over uncertain data. Specifically, we are given a collection P of n uncertain points in $\mathbb{R}$, each represented by its one-dimensional probability density function (pdf). The goal is to build a data structure on P such that given a query interval I and a probability threshold τ, we can quickly report all points of P that lie in I with probability at least τ. We present various structures with linear or near-linear space and (poly)logarithmic query time. Our structures support pdf's that are either histograms or more complex ones such as Gaussian or piecewise algebraic.

1 Introduction

Range searching, namely preprocessing a set of points into a data structure so that all points within a given query range can be reported efficiently, is one of the most widely studied topics in computational geometry and database systems [2], with a wide range of applications. Most of the works to date deal with *certain* data, that is, the points are given their precise locations in $\mathbb{R}^d$. Recent years, however, have witnessed a dramatically increasing amount of attention devoted to managing *uncertain* data because many real-world measurements are inherently accompanied with uncertainty. Besides the recent efforts in the data management community (see the survey [15]), various issues related with data uncertainty have also been studied in artificial intelligence [20], machine learning [5], statistics [18], and many other areas.

A popular approach to model data uncertainty [13, 25] is to consider each *uncertain point* p as a probability distribution over space. It is usually assumed that the points are independent but it is not necessary. The generally agreed semantics for querying uncertain data is the *thresholding approach* [13, 16], i.e., for a

*A preliminary version of this paper appeared as "Indexing uncertain data" in *ACM Symposium on Principles of Database Systems (PODS)*, 2009. P. K. Agawal is supported by NSF under grants CNS-05-40347, CCF-06 -35000, IIS-07-13498, and CCF-09-40671, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, by an NIH grant 1P50-GM-08183-01, by a DOE grant OEG-P200A070505, and by a grant from the U.S.–Israel Binational Science Foundation. S.-W. Cheng is supported by HKRGC under grant GRF 612107; Y. Tao is supported by HKRGC under grants GRF 1202/06, GRF 4161/07, and GRF 4173/08; and K. Yi is supported by Hong Kong Direct Allocation Grant (DAG07/08).

particular threshold τ, retrieve all the tuples that appear in the query range with probability at least τ. This problem turns out to be nontrivial even in one dimension. The naïve approach of examining each point one by one and computing its probability of being inside the query is obviously very expensive. Note that the independence assumption among the uncertain points is irrelevant as far as range queries are concerned.

Problem definition. We now define our problem more formally. Let $P = \{p_1, \ldots, p_n\}$ be a set of n uncertain points in $\mathbb{R}$, where each p_i is specified by its probability density function (pdf) $f_i : \mathbb{R} \to \mathbb{R}^+ \cup \{0\}$. We assume that each f_i is a piecewise-uniform function, i.e., a *histogram*, consisting of at most s pieces for some integer $s \geq 1$. In practice, such a histogram can be used to approximate any pdf with arbitrary precision. In some applications each point p_i has a discrete pdf, namely, it could appear at one of a few locations, each with a certain probability. This case can also be represented by the histogram model using infinitesimal pieces around these locations, so the histogram model also incorporates the discrete pdf case. We will adopt the histogram model by default throughout the paper. For simplicity, we assume s to be a constant for most of the discussion. Some of our structures also support more complicated pdf's (such as Gaussian or piecewise algebraic), and we will explicitly say so for these structures.

Given the set P and the associated pdf's, the goal is to build a data structure on them so that for a query interval I and a threshold τ, all points p such that $\Pr[p \in I] \geq \tau$ are reported efficiently. We also consider the version where τ is fixed in advance. We refer to the former as the *variable threshold* version and the latter as the *fixed threshold* version of the problem. The latter version is useful since in many applications the threshold is always fixed at, say, 0.5. Moreover, the user can often tolerate some error ε in the probability. In this case we can build $1/\varepsilon$ fixed-threshold structures with $\tau = \varepsilon, 2\varepsilon, \ldots, 1$, so that a query with any threshold can be answered with error at most ε.

Applications. The problem of range searching over uncertain data was first introduced by Cheng et al. [13] and has numerous applications in practice. For example, a certain measurement, say temperature, may be taken by multiple sensors in a sensor network. Due to various imprecision factors, the readings of these sensors may not be identical, in which case the temperature of a location can be conveniently modeled as a pdf. In this context, a query in our problem would retrieve "all the locations whose temperatures are between 100 and 120 degrees with probability at least 50%". It is not hard to see that there are many similar scenarios involving uncertain data. In fact, our problem is also important even in several traditional applications where no uncertainty seems to exist. For instance, consider a movie rating system (such as the one at Amazon) where each reviewer can give a rating from 1 to 10. A query of our problem would find "all the movies such that at least 90% of the ratings it receives are at least 8".

Previous results. The problem of range searching on uncertain data has received much attention in the database community over the last few years. The earliest work [13] considered the above problem in a simpler form, namely, where each $f_i(x)$ is a uniform distribution — a special case of our definition in which the histogram consists of only one piece. For the fixed-threshold version with threshold $0 < \tau \leq 1$, they proposed a structure of $O(n\tau^{-1})$ size with $O(\tau^{-1} \log n + k)$ query time, where k is the output size. These bounds depend on τ^{-1}, which can be arbitrarily large. This structure does not extend to histograms consisting of two or more pieces. They presented heuristics for the variable threshold version without any performance guarantees. Tao et al. [25] considered the problem in two and higher dimensions, and presented some data structures based on space partitioning heuristics. They prune points whose probability of being inside the query range is either too low or too high, but the query procedure visits all points of P in the worst case. Finally, yet another heuristic is presented in [22], but it is still the same as a sequential scan in the worst case.

Cheng et al. [13] also showed that the fixed-threshold version of the problem is at least as difficult as 2D halfplane range-reporting (i.e., report all points lying in a query halfplane), and that it can be reduced to 2D simplex queries (report all points lying in a query triangle). However the complexities of these two problems differ significantly: With linear space, a halfplane range-reporting query can be answered in time $\Theta(\log n + k)$ [11], while the latter takes $\Omega(\sqrt{n})$ time [12]. So there is a significant gap between the current upper and lower bounds for range searching over uncertain data.

Also related is the work by Singh et al. [24], who considered the problem of querying uncertain data that are categorical, namely, each random object takes a value from a discrete, unordered domain. The structures presented there are again heuristic solutions.

Our results. In this paper, we make a significant theoretical step towards understanding the complexity of range searching on uncertain data. We present linear or near-linear size data structures for both the fixed and variable threshold versions of the problem, with logarithmic or polylogarithmic query times. Specifically, we obtain the following results.

For the fixed-threshold version, we present a linear-size structure that answers a query in $O(\log n + k)$ time (Section 2). These bounds are clearly optimal (in the comparison model of computation). We first show that this problem can be reduced to a so-called *segments-below-point* problem: storing a set of segments in $\mathbb{R}^2$ so that all segments lying below a query point can be reported quickly. Then we present an optimal structure for the segments-below-point problem — a linear-size structure with $O(\log n + k)$ query time. This result shows that the fixed-threshold version has exactly the same complexity as the halfplane range-reporting problem, closing the large gap left in [13]. In Section 3 we present a simpler structure of size $O(n\alpha(n)\log n)$ and query time $O(\log n+k)$. This structure extends to more general pdf's, such as Gaussian distributions or other piecewise algebraic pdf's.

For the variable-threshold version, we use a different reduction and show that it can be solved by carefully storing a number of points in $\mathbb{R}^3$ in a structure for answering halfspace range queries. Combining with the recent result of Afshani and Chan [1] for 3D halfspace range reporting, we obtain a structure for the variable-threshold version of our problem with $O(n \log^2 n)$ space and $O(\log^3 n + k)$ query time (Section 4). Although the bounds have extra log factors in this case, our result shows that this problem is still significantly easier than 2D simplex queries.

Finally, we show that our structures can be dynamized, supporting insertions and deletions of (uncertain) points with a slight increase in the query time.

2 Fixed-Threshold Range Queries

We present an optimal structure for answering range queries on uncertain data where the probability threshold τ is fixed. Our structure uses linear space and answers a query in the optimal $O(\log n + k)$ time. These bounds do not depend on the particular value of τ. We first describe in Section 2.1 the reduction to the segments-below-point problem. Next we describe a segment-tree based data structure that uses linear space and answers a query in $O(\sqrt{n} + k)$ time or uses $O(n \log n)$ space and answers a query in $O(\log n + k)$ time (Section 2.3). We then improve this structure to achieve linear size and $O(\log n + k)$ query time simultaneously (Section 2.4). We conclude this section by describing how we make the structure dynamic.

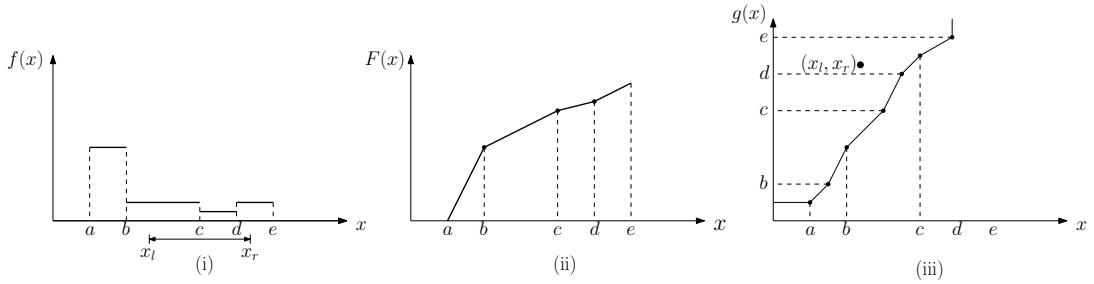

Figure 1: Reduction to the segments-below-point problem: (i) pdf, (ii) cdf, and (iii) threshold function.

2.1 A geometric reduction

Let p be an uncertain point in $\mathbb{R}$, and let $f : \mathbb{R} \to \mathbb{R}$ be its pdf.[1] Suppose the histogram of f consists of s pieces, and let

$$f(x) = y_i, \qquad \text{for } x_{i-1} \leq x < x_i, \quad i = 1, \ldots, s.$$

We set $x_0 = -\infty$, $x_s = \infty$, and $y_1 = y_s = 0$; see Figure 1 (i). The cumulative distribution function (cdf) $F(x) = \int_{-\infty}^{x} f(t)dt$ is a monotone piecewise-linear function consisting of s pieces; see Figure 1 (ii). Let the query range be $[x_l, x_r]$. The probability of p falling inside $[x_l, x_r]$ is $F(x_r) - F(x_l)$. We define a function $g : \mathbb{R} \to \mathbb{R}$, which we refer to as the *threshold function*. For a given $a \in \mathbb{R}$, let $g(a)$ be the minimum value b such that $F(b) - F(a) \geq \tau$. If no such b exists, $g(a)$ is set to ∞; see Figure 1 (iii).

Lemma 2.1 *The function $g(x)$ is non-decreasing and piecewise linear consisting of at most $2s$ pieces.*

Proof: Suppose we continuously vary x from $-\infty$ to ∞. For $x = -\infty$, $g(x) = \min\{y \mid F(y) = \tau\}$; $g(x)$ stays the same until x reaches x_1. As we increase x further, $g(x)$ increases linearly, with the slope depending on the pieces of the histogram f that contain x and $g(x)$. When either x or $g(x)$ passes through one of the x_i's, the slope changes. There are at most $2(s-1)$ such changes; see Figure 1. □

Given the description of the pdf f, the function g can be constructed easily. Once we have the threshold function g, the condition $\Pr[p \in [x_l, x_r]] \geq \tau$ simply becomes checking whether $x_r \geq g(x_l)$. Geometrically, this is equivalent to testing whether the point $(x_l, x_r) \in \mathbb{R}^2$ lies above the polygonal line representing the graph of g (see Figure 1). We construct the threshold function g_p for each point p in P. Let S be the set of at most $2ns$ segments in $\mathbb{R}^2$ that form the pieces of these n functions; S can be constructed in $O(n)$ time. We label each segment of g_p with p. The problem of reporting the points of P that lie in the interval $[x_l, x_r]$ with probability at least τ becomes reporting the segments of S that lie below the point $(x_l, x_r) \in \mathbb{R}^2$: If the procedure returns a segment labeled with p, we return the point p. Each polygonal line being x-monotone, no point is reported more than once.

We thus have the following problem at hand: Let S be a set of n segments in $\mathbb{R}^2$. Build a data structure on S so that for a query point $q \in \mathbb{R}^2$, the set of segments in S lying directly below q, denoted by $S[q]$, can be reported efficiently. For simplicity, we assume the coordinates of the endpoints of S to be distinct; this assumption can be removed using standard techniques. We call this problem the *segments-below-point* problem.

[1] Through this paper we do not distinguish between a function and its graph.

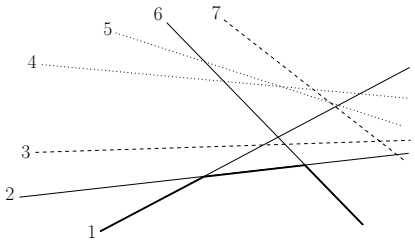

Figure 2: The data structure for a set of lines: thick polygonal chain is the lower envelope of S; $L_1(S) = \{1, 2, 6\}$, $L_2(S) = \{3, 7\}$, $L_3(S) = \{4, 5\}$.

2.2 Half-plane range reporting

We begin by describing a structure for the special case when all segments in S are full lines and we want to report the lines of S lying below a query point. This problem is dual to the well-known half-plane range reporting problem, for which there is an $O(n)$-size structure with $O(\log n + k)$-time [11]. We briefly describe a variant of this structure (in the dual setting), denoted by $\mathcal{H}(S)$, which we will use as a building block.

If we view each line ℓ in S as a linear function $\ell : \mathbb{R} \to \mathbb{R}$, then the *lower envelope* of S is the graph of the function $E_S(x) = \min_{\ell \in S} \ell(x)$, i.e., it is the boundary of the unbounded region in the planar map induced by S that lies below all the lines of S (see Figure 2). We represent the lower envelope as a sequence $x_0 = -\infty, \ell_1, x_1, \ell_2, \ldots, \ell_r, x_r = +\infty$, where the x_i's are the x-coordinates of the vertices of the lower envelope, and ℓ_i is the line that appears on the lower envelope in the interval $[x_{i-1}, x_i]$. Note that the lines appear along the envelope in decreasing order of their slopes. We partition S into a sequence $L_1(S), L_2(S), \ldots$, of subsets, called *layers*. $L_1(S) \subseteq S$ consists of the lines that appear on the lower envelope of S. For $i > 1$, $L_i(S)$ is the set of lines that appear on the lower envelope of $S \setminus \bigcup_{j=1}^{i-1} L_j(S)$; see Figure 2. For each i, we store the aforementioned representation of layer $L_i(S)$ in a list. To answer a query $q = (q_x, q_y)$, we start from $L_1(S)$ and locate the interval $[x_{i-1}, x_i]$ that contains q_x, using binary search. Next we walk along the envelope of $L_1(S)$ in both directions, starting from ℓ_i, to report the lines lying below q, in time linear to the output size. Then we query the rest of the layers $L_2(S), L_3(S), \ldots$ in order until no lines have been reported at a certain layer. By using *fractional cascading* [10] on the x-coordinates of the envelopes of these layers, the total query time can be improved to $O(k)$ plus the initial binary search in $L_1(S)$. Fractional cascading augments these lists with copies of elements from other lists, but the size of the structure remains linear, and it can be constructed in $O(n \log n)$ time [10, 11]. The following statement is slightly more general than what appeared in [11].

Lemma 2.2 *Let S be a set of n lines in the plane. S can be preprocessed in $O(n \log n)$ time into a data structure of linear size, so that given a query point $q \in \mathbb{R}^2$ and any line in $L_1(S)$ below q, all k lines of S lying below q can be reported in $O(k)$ time.*

2.3 Segment-tree based structure

This subsection describes a structure for the segments-below-point problem, based on the segment tree, that uses linear space and answers a query in $O(\sqrt{n} + k)$ time, or $O(n \log n)$ space and answers a query in $O(\log n + k)$ time. We later (cf. Section 2.4) bootstrap this structure to improve the query time to $O(\log n + k)$ while keeping the size linear.

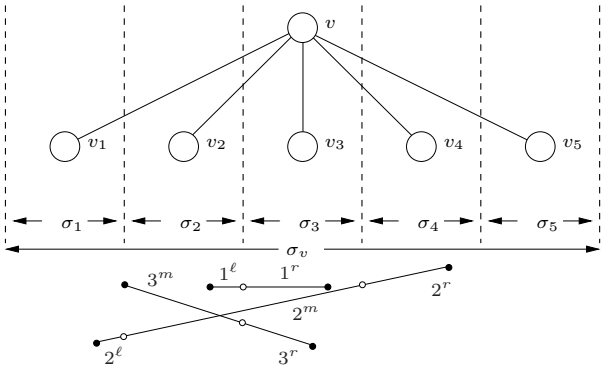

Figure 3: A segment tree node with fanout $r = 5$.

We fix a parameter r and construct a segment tree $\mathcal{T}$ of fanout r — an r-ary tree that defines an r-way hierarchical decomposition of the plane into vertical slabs, each associated with a node of $\mathcal{T}$. Let σ_v denote the slab corresponding to a node v. The slabs associated with the children of v are defined as follows. We partition σ_v into r vertical sub-slabs $\sigma_1, \ldots, \sigma_r$, each containing roughly the same number of endpoints of segments in S. We create r children $v_1, \ldots, v_r$ of v and associate σ_i with v_i; see Figure 3. A node v is a leaf if σ_v does not contain any endpoint of S in its interior.

We call any number of contiguous sub-slabs a *multi-slab* at v. Let $\sigma_v[i : j] = \sigma_i \cup \cdots \cup \sigma_j$ denote the multi-slab at v spanned by sub-slabs $\sigma_i, \ldots, \sigma_j$. Obviously there are $O(r^2)$ multi-slabs at any v. For any segment s, consider the highest node v where it intersects two or more sub-slabs; i.e., s intersects the boundary of a slab. Let $S_v \subseteq S$ be the set of segments for which v is the highest node at which they are split. At v we partition $s \in S_v$ into up to three pieces: a middle piece s^m that spans the maximal multi-slab at v, a left piece s^l, and a right piece s^r. More precisely, if s spans slabs $\sigma_i, \ldots, \sigma_j$, then $s^m = s \cap \sigma_v[i : j]$, and it is associated with the multi-slab $\sigma_v[i : j]$. If the left endpoint of s lies in the interior of σ_{i-1}, then $s^l = s \cap \sigma_{v_{i-1}}$, and if the right endpoint of s lies in the interior of $\sigma_{v_{j+1}}$, then we set $s^r = s \cap \sigma_{v_{j+1}}$. See Figure 3. Next, we recursively partition the left and right pieces of s following the r-ary tree. A segment is thus partitioned into at most three pieces at any level of the tree, resulting in a total of $O(\log_r n)$ pieces. Note that each piece with spans a multi-slab at some node.

Let $S_v^{i:j}$ denote the set of segments associated with the multi-slab $\sigma_v[i : j]$ at v, and let $H_v^{i:j}$ denote the full lines containing these segments. For a point $q \in \sigma_v[i : j]$, a segment $s \in S_v^{i:j}$ lies below q if and only if the line containing s lies below q. We therefore build the halfplane structure on $H_v^{i:j}$ described in Section 2.2. Since $\sum |S_v^{i:j}| = O(n \log_r n)$ and the structure built for each multi-slab has linear size, the size of the overall structure is also $O(n \log_r n)$, and it can be constructed in $O(n \log_r n \log n)$ time.

To report the segments of S lying below a query point q, we visit all the nodes v of $\mathcal{T}$ such that $q \in \sigma_v$. At each v, we query the halfplane structures corresponding to all the multi-slabs that contain q. Overall we query a total of $O(r^2 \log_r n)$ multi-slabs, so the total query time is $O(r^2 \log_r n \log n + k)$. Choosing $r = n^{1/4}/\sqrt{\log n}$ gives us the following.

Lemma 2.3 *Let S be a set of n segments in the plane. S can be preprocessed in $O(n \log n)$ time into a linear-size structure so that all segments of S lying below a query point can be reported in $O(\sqrt{n} + k)$ time.*

Remarks. The query time of the above scheme can be improved to $O(n^\varepsilon + k)$ for any small constant ε, but

a query time of $O(\sqrt{n}+k)$ is all we need for the bootstrapping later.

If we choose $r = 2$, we can construct a standard binary segment tree. At each node σ_v is split into two slabs by a vertical line $x = x_v$. We no longer need multislabs — each of the two subslabs of σ_v is associated with a child of v and the only multislab is σ_v itself. The size of the data structure is now $O(n \log n)$, and the query time is $O(\log^2 n + k)$. The query time can be improved to $O(\log n + k)$ by using fractional cascading: We need to query $O(\log n)$ halfplane structures associated with the nodes along a root-to-leaf path of the segment tree with the same query point $q = (q_x, q_y)$. By the discussion in Section 2.2, it is sufficient to locate the interval $[x_{i-1}, x_i]$ containing q_x in the first layer of each of these structures, and the rest of the cost will be linear in the output size. Since the first layer of each halfplane structure is a linear list, this is exactly the standard situation where fractional cascading [10] can be applied. Again, fractional cascading augments each list with auxiliary information, which increases the size of the structure by a constant factor. The time of locating all these intervals will be the time to search in the first list, plus $O(1)$ per succeeding list, i.e., $O(\log n + \log n) = O(\log n)$. We thus obtain the following.

Lemma 2.4 *Let S be a set of n segments in the plane. S can be preprocessed in $O(n \log^2 n)$ time into a structure of $O(n \log n)$ size so that all segments of S lying below a query point can be reported in $O(\log n + k)$ time.*

2.4 Optimal structure

We now describe an optimal structure for answering segments-below-point queries. We start with the binary segment-tree structure from the previous subsection. We stop the top-down construction of the segment tree $\mathcal{T}$ as soon as there are $\Theta(\log^2 n)$ endpoints of S left in the slab, that is, the "atomic slab" σ_z for each leaf z of $\mathcal{T}$ contains $\Theta(\log^2 n)$ endpoints of S. Since we have "fat" leaves, not all segments will be split — those with both endpoints lying in the same atomic slab will not be split. For an internal node v, let S_v be the subset of segments for which v is the highest node at which they are split, as in the previous data structure. For a leaf z, let S_z be the set of segments that lie completely in σ_z. We build a segment tree with large fan-out (Lemma 2.3) on S_z. Since $|S_z| = O(\log^2 n)$ and $\sum_z |S_z| \le n$, a segments-below-point query on S_z can be answered in $O(\log n + |S_z[q]|)$ time and the total size and construction time of the structure, summed over all leaves, are linear and $O(n \log n)$, respectively.

Next we describe the structure for the segments that are split at an internal node of $\mathcal{T}$. At each node u, the segment $s \in S_u$ is split by $x = x_u$ into a left segment s^- and a right segment s^+. Let $S_u^- = \{s^- \mid s \in S_u\}$, $S_u^+ = \{s^+ \mid s \in S_u\}$, $S^- = \bigcup_u S_u^-$, and $S^+ = \bigcup_u S_u^+$. Note that for any segment $s \in S_u$ and for any point $q \neq x_u$ lying above s, either s^- or s^+ lies below q, but not both. Therefore it suffices to build separate structures for S^- and S^+ and report $S^-[q]$ and $S^+[q]$ for a query point q. We describe the structure for S^-; a similar scheme works for S^+.

We now introduce some notation. Let $\mathbb{V}$ denote the set of nodes $v \in \mathcal{T}$ such that v is the right child of $p(v)$, where $p(v)$ denotes the parent of v. Let Λ be the set of pairs (u, z) such that z is a leaf and u is a proper ancestor of z;

$$|\Lambda| = O\left(\log n \frac{n}{\log^2 n}\right) = O\left(\frac{n}{\log n}\right).$$

For a node $v \in \mathbb{V}$, let $\Lambda(v) \subseteq \Lambda$ be the set of pairs (u, z) such that z is a descendant leaf of the left sibling of v and u is a proper ancestor of $p(v)$. For a pair $(u, z) \in \Lambda$, let $S_{uz} \subseteq S_u^-$ be the set of segments whose left endpoints lie in σ_z; set $n_{uz} = |S_{uz}|$. For two different pairs $(u, z), (u', z') \in \Lambda$, S_{uz} and $S_{u'z'}$ are disjoint because the left endpoints of all segments in S_{uz} lie in σ_z and the right endpoints lie on the splitting line $x = x_u$. Hence, $\sum_{(u,z)\in\Lambda} n_{uz} \le n$. Set $L_z = \bigcup_u S_{uz}$, where the union is taken over all proper ancestors of

z; it is clear that $\sum_z |L_z| \leq n$. Next, for a node $v \in \mathcal{T}$, let $\Phi_v \subseteq S^-$ be the set of segments that completely span σ_v but their left endpoints lie in $\sigma_{p(v)}$. In particular, if v is the left child of $p(v)$, i.e., $v \notin \mathbb{V}$, then $\Phi_v = \emptyset$; otherwise (i.e., $v \in \mathbb{V}$), each segment $s \in \Phi_v$ belongs to some S_{uz} where u is a proper ancestor of $p(v)$ and z is a descendant of the left sibling of v, that is,

$$\Phi_v = \bigcup_{(u,z)\in\Lambda(v)} S_{uz}. \tag{1}$$

Therefore a set S_{uz} is included in Φ_v at all nodes $v \in \mathbb{V}$ such that $p(v)$ lies on the path from z to the left child of u; see Figure 4.

Let q be a query point, let z be the leaf of $\mathcal{T}$ such that $q \in \sigma_z$, and let Π_q be the path in $\mathcal{T}$ from the root to z. If a segment $s \in S^-$ lies below q, then either (i) $s \in L_z$, or (ii) $s^- \in \Phi_v$ for some $v \in \Pi_q$ and the line containing s^- lies below q. To handle (i), using Lemma 2.3, we build in $O(|L_z| \log n)$ time a data structure of linear size that returns $L_z[q]$ in time $O(\log n + |L_z[q]|)$, as $|L_z| = O(\log^2 n)$. It thus suffices to describe how we handle case (ii). The binary-segment-tree data structure in the previous subsection basically preprocesses each Φ_v separately, leading to an $O(n \log n)$ size data structure. We build a more global structure to reduce the size to linear.

For a pair $(u, z) \in \Lambda$, let H_{uz} be the set of lines containing the segments in S_{uz}. We preprocess H_{uz} into a linear-size halfplane range reporting data structure using Lemma 2.2. To report $\Phi_v[q]$ for a point $q \in \sigma_v$, we need a structure that returns one *representative* line of $L_1(H_{uz})$ lying below q (if there exists one), for each pair $(u, z) \in \Lambda(v)$. We can then use the structure built on H_{uz} to report the remaining lines in $H_{uz}[q]$ (see Lemma 2.2). One possibility is to build a structure on the set $\bigcup_{(u,z)\in\Lambda(v)} L_1(H_{uz})$ at each node v to find such a line, but $|L_1(H_{uz})| = |H_{uz}|$ in the worst case, so this will again lead to a structure of size $O(n \log n)$. The following observation will help us in reducing the size.

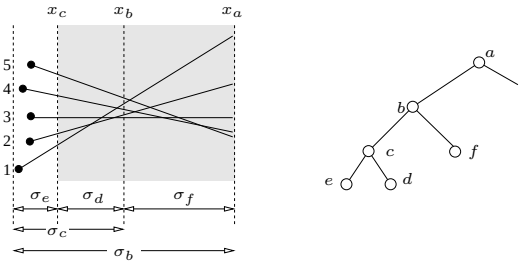

Figure 4: Set $S_{ae} = \{1, 2, 3, 4, 5\}$ and the strip Σ_{ae} (shaded); S_{ae} is included in Φ_e (queried in slab σ_d) and Φ_c (queried in slab σ_f); $H^e_{ae} = \{1, 2, 3\}$ and $H^c_{ae} = \{3, 4, 5\}$.

Fix a pair $(u, z) \in \Lambda$. Let Σ_{uz} be the strip formed by the splitting line at u and the right boundary of σ_z; see Figure 4. The right endpoint of each segment in S_{uz} lies on the right edge of Σ_{uz} and the left endpoint lies to the left of Σ_{uz}, so each segment of S_{uz} spans Σ_{uz}. Let $w_1, w_2, \ldots, w_r$ be the nodes such that each w_i is the right child of $p(w_i)$ and $p(w_i)$ lies on the path from the left child of u to z; the (left) sibling of each w_i also lies on this path. For the example in Figure 4, if $u = a, z = e$, then these u_i's are d and f. The slabs $\sigma_{w_1}, \ldots, \sigma_{w_r}$ induce a partitioning of Σ_{uz}. Moreover $(u, z) \in \Lambda(v)$ if and only if $v = w_i$ for some $1 \leq i \leq r$. For such a node v, let $H^v_{uz} \subseteq L_1(H_{uz})$ be the set of lines that appear on the lower envelope of H_{uz} within σ_v; set $n^v_{uz} = |H^v_{uz}|$. At most one line of H^v_{uz} will appear on the lower envelope of H_{uz} to the right of σ_v (for example, only line 3 of H^e_{ae} may appear on the lower envelope of H_{ae} to the right of σ_d).

Since $r = O(\log n)$, we have

$$\sum_{v \in \mathbb{V}:(u,z) \in \Lambda(v)} n_{uz}^v = n_{uz} + O(\log n).$$

For a pair (u, z), the sets H_{uz}^v can be computed in $O(n_{uz} \log n)$ time by constructing the lower envelope of H_{uz}. The total time spent over all pairs in Λ is $O(n \log n)$.

For a node $v \in \mathbb{V}$, let $\Gamma_v = \bigcup_{(u,z) \in \Lambda(v)} H_{uz}^v$; Γ_v is the set of *representative lines* stored at v. We build in $O(|\Gamma_v| \log n)$ time the halfplane-range-reporting structure of linear size on Γ_v. For each line ℓ in Γ_v, we store a pointer to its copy in H_{uz}. Finally, as in the structure of Lemma 2.4, we also use fractional cascading on these half-plane range-reporting structures. This completes the description of the structure we build. To analyze the size of the data structure and its preprocessing time, we note that

$$\begin{aligned}
\sum_{v \in \mathbb{V}} |\Gamma_v| &= \sum_{v \in \mathbb{V}} \sum_{(u,z) \in \Lambda(v)} n_{uz}^v \\
&= \sum_{(u,z) \in \Lambda} \sum_{v \in \mathbb{V}:(u,z) \in \Lambda(v)} n_{uz}^v \\
&= \sum_{(u,z) \in \Lambda} (n_{uz} + O(\log n)) \\
&= O(n) + O\left(\frac{n}{\log n} \log n\right) \\
&= O(n).
\end{aligned}$$

Hence, the total size of the structure is $O(n)$, and it can be constructed in $O(n \log n)$ time.

For a query point q, the set $S[q]$ is reported as follows. We first find in $O(\log n)$ time the leaf z whose slab contains q. Next, we report in $O(\log n + |S_z[q]|)$ time the set $S_z[q]$. Then, we report in $O(\log n + |L_z[q]|)$ time the set $L_z[q]$. Next, for each node $v \in \Pi_q$, if v is the right child of $p(v)$, we report the set $\Phi_v[q]$. More precisely, we visit the nodes of Π_q in a top-down manner. For each node $v \in \Pi_q \cap \mathbb{V}$, we first query the structure on Γ_v and report the set $\Gamma_v[q]$, i.e., the set of representative lines of H_v that lie below q. This takes time $O(\log n + |\Gamma_v[q]|)$ time for the first node v; for each successive node, the time spent is only linear in the number of representative lines we return by fractional cascading. Consider each $\ell \in \Gamma_v[q]$ in turn. Suppose ℓ is a representative line from some $H_{uz'}$. We then report the set $H_{uz'}[q]$ in time proportional to its size by querying the structure built on $H_{uz'}$. Thus the total time spent at all the nodes on Π_q is $O(\log n)$ plus a term linear to the output size. Finally, we report $S^+[q]$ in a similar manner. Putting everything together, the total query time is $O(\log n + |S[q]|)$.

Theorem 2.5 *Let S be a set of n segments in $\mathbb{R}^2$. S can be preprocessed in $O(n \log n)$ time into a linear-size structure so that all k segments of S lying below a query point can be reported in $O(\log n + k)$ time.*

Since each uncertain point produces $O(s)$ segments in the segments-below-point problem, we immediately have the following.

Corollary 2.6 *Let P be a set of n uncertain points in $\mathbb{R}$, each associated with a histogram having s pieces, and let $0 < \tau \leq 1$ be a threshold parameter. P can be preprocessed in $O(n \log n)$ time into a linear-size structure so that a range query on P with probability threshold τ can be answered in $O(\log n + k)$ time, where k is the output size.*

2.5 Dynamization

Finally we briefly discuss how to make our structure dynamic, i.e., supporting insertions and deletions of uncertain points in the uncertain data set. When an uncertain point is being inserted or deleted, we need to insert or delete the $2s$ segments in the graph of its threshold function (cf. Section 2.1) in our segments-below-point structure. If only insertions are to be supported, we can apply the *logarithmic method* [6] to Theorem 2.5. Then standard analysis gives us a linear-size semi-dynamic structure that answers a query in $O(\log^2 n + k)$ time and supports insertion of a point in amortized $O(\log^2 n)$ time. Unfortunately, it is hard to support deletions in our optimal structure, since it crucially relies on the halfplane searching structure of [11], which is inherently static. The best known dynamic structure for halfplane range reporting uses $O(n \log n)$ space, supports insertions and deletions in $O(\text{polylog}\, n)$ time amortized, and answers queries in $O(\log n + k)$ time [8, 9]. Currently, it is unknown if one can obtain a linear-size dynamic structure with $O(\text{polylog}\, n)$ update times. Since super-linear space is unavoidable, we can simply plug this dynamic halfplane structure into the segment-tree based structure with fanout 2 (see the remark following Lemma 2.3) and obtain the following.

Theorem 2.7 *Given a set P of n uncertain points in $\mathbb{R}$ and their pdf's, each of which is a histogram of constant size, and a parameter $0 < \tau \leq 1$, P can be maintained in a dynamic structure of size $O(n \log^2 n)$ that answers a range query with probability threshold τ in $O(\log^2 n + k)$ time, and supports insertions and deletions of an uncertain point in $O(\text{polylog}\, n)$ amortized time.*

Remark. If s is not a constant, all our space and query bounds in this section still hold by simply replacing n by sn. Note that since the input has size $\Theta(sn)$, a structure with size $O(sn)$ is still linear in the input. The update time in Theorem 2.7 becomes $O(s \,\text{polylog}\, n)$ since s segments need to be inserted or deleted.

3 Handling More General Pdf's

In Section 2.1, we converted the uncertain range searching problem to the problem of storing a set of x-monotone polygonal chains in a data structure so that all the chains below a query point can be reported efficiently. In this section, we follow a more direct approach to solve this problem. It results in a structure with $O(n\alpha(n) \log n)$ size and $O(\log n + k)$ query time, where $\alpha(n)$ is the *inverse Ackermann function*, an extremely slow-growing function. Although the space bound is not as good as the structure in Theorem 2.5, the new structure we present below is simpler and easily extends to the case where the polygonal chains are replaced by more general curves, such as piecewise-algebraic curves. This will allow us to handle pdf's that are more general than histograms, which we will elaborate later.

The framework of our structure is similar to that of the 3D halfspace searching structure of Chan [7]. Let C be the set of n polygonal chains representing the n threshold functions; each of them consists of at most $2s$ segments. We first randomly sample a subset R_1 of $n/2$ chains from C. Then for $i = 2, \ldots, \log n$, we randomly sample a subset R_i of $n/2^i$ chains from R_{i-1}. For each i, we compute its lower envelope E_i of R_i. According to [19], E_i consists of at most $O(|R_i| \cdot \alpha(|R_i|))$ segments. From the boundary points of these segments we shoot a ray downwards, yielding a set Ξ_i of trapezoids (see Figure 5). For each trapezoid $t \in \Xi_i$, we find the set of all chains in C that intersect the trapezoid, denoted C_t. We store C_t simply as a list associated with t, and call C_t the *conflict list* of t. Following the random-sampling framework of Clarkson

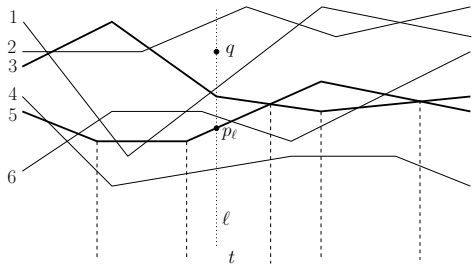

Figure 5: The thick chains are in the random sample R_i. The dashed lines divide the lower envelope of R_i into trapezoids. For the trapezoid t, its conflict list C_t consists of chains 4 and 6.

and Shor [14], the expected size of C_t is at most $O(2^i)$. Therefore, the expected total size of our structure is

$$O\left(\sum_i^{\log n} 2^i \cdot |R_i|\alpha(|R_i|)\right) \quad = \quad O\left(\sum_i^{\log n} n\alpha(|R_i|)\right) = O(n\alpha(n)\log n).$$

For each R_i, we can compute its lower envelope and all the conflict lists in expected $O(n \log n)$ time, so we can build the structure in a total of $O(n \log^2 n)$ time in expectation.

Finally, we add a fractional cascading structure on the E_i's, so that given a vertical line ℓ, we can find all the trapezoids intersected by ℓ, one from each E_i, in $O(\log n)$ time. The size of this structure is only $O(n\alpha(n))$, and it can be built in the same amount of time.

Now we describe how a query for a point q is answered. First, we find all the $\log n$ trapezoids in $O(\log n)$ time that intersect the vertical line ℓ passing through q, one from each E_i; let $t_i \in \Xi_i$ be the trapezoid that intersects ℓ. We describe a procedure below that for a given r, finds in $O(r)$ expected time the r lowest chains of C along ℓ, i.e., those chains whose intersections with ℓ have the r smallest y-coordinates. Then we can try successively larger and larger values of $r = 1, 2, 4, 8, \ldots$, and halt as soon as at least one of the r lowest chains is above q. When we stop we have $r/2 \leq k < r$, and we just report the k chains that are actually below q. The total time spent will be $O(\log n + 1 + 2 + 4 + \cdots + r) = O(\log n + k)$.

Let $0 < \delta < 1$ be a parameter. We first give a Monte Carlo algorithm with running time $O(r/\delta^2)$ that fails with probability $O(\delta^3)$; then we show how to convert it to a Las Vegas algorithm that never fails and runs in expected time $O(r)$. We will only consider the case $r/\delta < n$; otherwise the problem is trivial since we can simply scan all the n chains. Set $\rho = \lfloor \log_2(r/\delta) \rfloor$ and let t_ρ be the trapezoid of Ξ_ρ that intersects ℓ. We first check whether $|C_{t_\rho}| > r/\delta^2$. If so the algorithm immediately aborts with a failure. Otherwise we scan the entire list C_{t_ρ}. Let p_ℓ be the intersection point of ℓ and the upper boundary of the trapezoid t_ρ (Figure 5). While scanning C_{t_ρ} we check whether there are at least r chains below p_ℓ. If so the algorithm succeeds in finding the r lowest chains along ℓ; else the algorithm fails.

This Monte Carlo algorithm clearly runs in time $O(r/\delta^2)$. Now we analyze its failure probability. There are two cases that the algorithm may fail: (a) $|C_{t_\rho}| > r/\delta^2$; or (b) there are fewer than r chains in C_{t_ρ} below p_ℓ. Since $\mathsf{E}[|C_{t_\rho}|] = O(2^\rho) = O(r/\delta)$, by Markov inequality,

$$\Pr[|C_{t_\rho}| > r/\delta^2] = O(\delta).$$

For (b) to happen, the chain corresponding to the upper boundary of t_ρ must be one of the r lowest chains along ℓ. Since R_ρ is a random sample of size $n/2^\rho$, this occurs with probability $O(r/2^\rho) = O(\delta)$. The algorithm thus fails with probability $O(\delta)$. Finally, keeping three independent data structures will bring down the failure probability to $O(\delta^3)$, with only a constant-factor blowup in the space and query costs.

Finally, we show how to convert the Monte Carlo algorithm into a Las Vegas algorithm with expected running time $O(r)$, which will complete the description of the query algorithm. We invoke the algorithm above with $\delta = 2^{-1}, 2^{-2}, 2^{-3}, \ldots$, stopping as soon as some invocation succeeds. Let X_i be the indicator random variable whose value is 1 if the algorithm is invoked with $\delta = 2^{-i}$, and 0 otherwise. Note that

$$\begin{aligned} \mathsf{E}[X_i] &= \Pr[X_i = 1] = \Pr[\text{all first } i-1 \text{ invocations fail}] \\ &\leq \Pr[\text{the } (i-1)\text{-th invocation fails}] \\ &= O(1/2^{3(i-1)}). \end{aligned}$$

Then the total expected running time is

$$\sum_{i \geq 1} \mathsf{E}[X_i] \cdot O(r \cdot 2^{2i}) = \sum_{i \geq 1} O\left(\frac{1}{2^{3(i-1)}}\right) \cdot O(r \cdot 2^{2i}) = O(r).$$

Theorem 3.1 *Given a set P of n uncertain points in $\mathbb{R}$, their pdf's, each of which is a histogram of constant size, and a parameter $0 < \tau \leq 1$, P can be preprocessed in $O(n \log^2 n)$ time into a structure of size $O(n\alpha(n) \log n)$, where $\alpha(n)$ is the inverse Ackermann function, so that a range query with probability threshold τ can be answered in expected $O(\log n + k)$ time.*

As commented earlier, this structure easily extends to more general pdf's. The algorithm remains the same, except that the complexity of E_i depends on the input pdf's. In the analysis above, the threshold functions are a collection of piecewise linear functions, and the complexity of the lower envelope of any n such functions is $O(n\alpha(n))$ [19]. With other families of pdf's, the threshold functions will have different forms. Note that Lemma 2.1 easily extends to other piecewise functions. Recall that the threshold function $g(x)$ satisfies $F(g(x)) - F(x) = \tau$ (unless $g(x) = \infty$), where $F(x)$ is the cdf. For instance, if the pdf is a piecewise linear function, then the $F(x)$ will be piecewise quadratic, and the threshold function will have the form $g(x) = -c_1 + \sqrt{c_2 x^2 + c_3 x + c_4}$ with different constants c_1, c_2, c_3, c_4 in each of the at most $2s$ pieces, by solving $F(g(x)) - F(x) = \tau$.

Interestingly, the complexity of the lower envelope of these threshold functions only depends on how many times two pieces from two different threshold functions could intersect. If two pieces intersect at no more than c points, then the complexity of the lower envelope of n such functions is $\lambda_{c+2}(n)$, the maximum length of any $(n, c+2)$ *Davenport-Schinzel sequence* [17]. If each threshold function consists of a single unbounded curve (e.g., when the pdf's are Gaussian distributions), then the complexity of the envelope is $\lambda_c(n)$. Thus Theorem 3.1 still holds, with the space bound changing to $O(\lambda_{c+2}(n) \log n)$ and $O(\lambda_c(n) \log n)$, respectively.

Theorem 3.2 *Let P be a set of n uncertain points in $\mathbb{R}$. Suppose the threshold function of each $p \in P$ has s pieces and any two pieces from two different threshold functions intersect at no more than c points. P can be preprocessed in $O(\lambda_{c+2}(n) \log^2 n)$ time into a structure of size $O(\lambda_{c+2}(n) \log n)$, where $\lambda_t(n)$ is the maximum length of an (n, t) Davenport-Schinzel sequence, so that a range query with a fixed probability threshold τ can be answered in expected $O(\log n + k)$ time. If each threshold function consists of a single unbounded curve, and any two such functions intersect at no more than c points, then the space and preprocessing time become $O(\lambda_c(n) \log n)$ and $O(\lambda_c(n) \log^2 n)$, respectively.*

Sharp bounds for $\lambda_c(n)$ are known for any fixed c: $\lambda_1(n) = n, \lambda_2(n) = 2n - 1, \lambda_3(n) = \Theta(n\alpha(n))$, $\lambda_4(n) = \Theta(n2^{\alpha(n)})$ and $\lambda_{2t+2}(n) = n2^{(1/t!)\alpha^t(n) + \Theta(\alpha^{t-1}(n))}$. These bounds are very close to linear due

to the extremely slow growth of $\alpha(n)$; see the survey by Agarwal and Sharir [4] for a complete treatment of Davenport-Schinzel sequences and their applications, and the recent paper [23] for slightly improved bounds.

For most common pdf's, c is a small constant. For instance if the pdf's are histograms, then $c = 1$ as obtained in Theorem 3.1. If the pdf's are piecewise linear, then the threshold functions are piecewise with each piece having the form $g(x) = -c_1 + \sqrt{c_2 x^2 + c_3 x + c_4}$. Two such functions intersect at no more than two points, so the size of our structure is $O(\lambda_4(n) \log n)$. Note that for some pdf's, such as Gaussian, the threshold function may not have a closed form, so we will have to use numerical methods to compute their intersections. Nevertheless, even when the $g(x)$'s do not have closed forms, we can often show that they do not intersect too many times.

Lemma 3.3 *For two Gaussian distributions, their threshold functions intersect at most twice.*

Proof: Let the pdf's of the two Gaussians be $f_1(x)$ and $f_2(x)$. Let their cdf's be $F_1(x)$ and $F_2(x)$, respectively. The threshold function is $g_i(x) = F_i^{-1}(F_i(x) + \tau), i = 1, 2$. We only need to consider the domain in which $F_i(x) + \tau < 1$. Let (a, b) be an intersection point of g_1 and g_2, i.e., $g_1(a) = g_2(a) = b$. Then we have $F_i(b) = F_i(a) + \tau, i = 1, 2$. Subtracting the two equations, we have

$$F_1(b) - F_2(b) = F_1(a) - F_2(a).$$

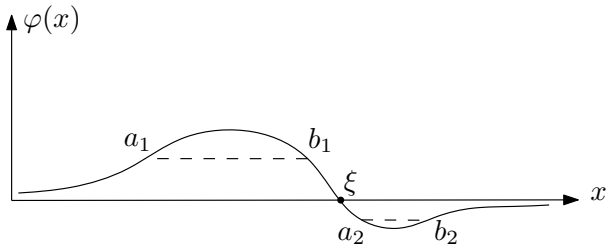

Figure 6: The function $\varphi(x)$ admits at most 2 intersections of $g_1(x)$ and $g_2(x)$.

Define the function $\varphi(x) = F_1(x) - F_2(x)$. Consider its derivative

$$\varphi'(x) = f_1(x) - f_2(x) = \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) - \frac{1}{\sqrt{2\pi}\sigma_2} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right)$$

for some $\mu_1, \mu_2, \sigma_1, \sigma_2$. We observe that $\varphi'(x)$ has at most two roots. If $\varphi'(x)$ has one root, $\varphi(x)$ is unimodal or inverse-unimodal; if $\varphi'(x)$ has two roots, by combining with the fact that $\varphi(-\infty) = \varphi(+\infty) = 0$, we can conclude that $\varphi(x)$ must have exactly one root, and that $\varphi(x)$ is unimodal before the root and inverse-unimodal after it, or vice-versa; see Figure 6.

On the other hand, the intersection points of g_1 and g_2 satisfy the following two conditions:

(C1) For any intersection point (a, b), $\varphi(a) = \varphi(b)$.

(C2) For any two different intersection points (a_1, b_1), (a_2, b_2), we must have $b_1 < b_2$ if $a_1 < a_2$ since $g_i(x)$ is non-decreasing.

If $\varphi'(x)$ has only one root, i.e., $\varphi(x)$ is unimodal (or inverse-unimodal), then there is at most one intersection point of g_1 and g_2 that satisfies both (C1) and (C2), so let us assume that $\varphi'(x)$ has two roots and $\varphi(x)$ has one root, say, ξ. Let (a_1, b_1) be an intersection point of g_1, g_2. Since $\varphi(x) > 0$ for $x \in (-\infty, \xi)$ and $\varphi(x) < 0$ for $x \in (\xi, \infty)$, $\xi \notin [a_1, b_1]$. Suppose $a_1 < b_1 < \xi$. Let (a_2, b_2) be another intersection point, with $a_2 > a_1$. We claim that $a_2 > \xi$. Indeed, if $a_2 < \xi$, then the unimodality of φ in the range $[-\infty, \xi]$ implies that $b_2 < b_1$. But by (C2), $b_2 > b_1$, a contradiction. Hence, $a_2 > \xi$. The same argument implies that there is at most one intersection point of g_1 and g_2 that lies after ξ, implying that they have at most two intersection points. □

Invoking Theorem 3.2, our structure for Gaussian distributions has size $O(\lambda_2(n) \log n) = O(n \log n)$.

Remark. If s is not a constant, all our space and query bounds in this section still hold by simply replacing n by sn.

4 Variable-Threshold Queries

The geometric reduction in Section 2.1 does not work if τ, the probability threshold parameter, is part of a query. This section shows how to decompose the variable-threshold version of the problem into answering a few *3D halfspace range-reporting* queries, which yields a structure of $O(n \cdot \text{polylog}(n))$ size with $O(\text{polylog}(n) + k)$ query time.

In the 3D halfspace searching problem, we want to store a set of points in $\mathbb{R}^3$ in a data structure such that all points below a given a query plane can be reported efficiently. By duality, this is equivalent to storing a set of planes in $\mathbb{R}^3$ in a structure such that for a query point p, all planes below p are reported. As in the 2D case, this problem can also be solved in linear space and $O(\log n + k)$ query time [1].

Consider a particular point p and its pdf $f_p(x)$. As in Section 2 suppose that the histogram $f_p(x)$ consists of s pieces:

$$f_p(x) = y_i, \qquad \text{for } x_{i-1} \le x < x_i,\ i = 1, \ldots, s,$$

where $x_0 = -\infty$, $x_s = \infty$ and $y_1 = y_s = 0$. For a query range $I = [x_l, x_r]$, let us consider $\Pr[p \in [x_l, x_r]]$ as a *threshold function* of x_l and x_r, denoted by $g_p(x_l, x_r)$. If $x_l \in [x_{i-1}, x_i]$ and $x_r \in [x_{j-1}, x_j]$ for some $i \le j$, then $g_p(x_l, x_r)$ increases linearly in x_l, with y_i as the slope, and also increases linearly in x_r, with y_j as the slope, implying that $g_p(x_l, x_r)$ is a bivariate linear function in the rectangle $[x_{i-1}, x_i] \times [x_{j-1}, x_j]$. Thus $g_p(x_l, x_r)$ is a bivariate piecewise-linear function consisting of s^2 pieces; each piece spans a rectangle of the form $[x_{i-1}, x_i] \times [x_{j-1}, x_j]$, for some $i \le j$, in the $x_l x_r$-plane; see Figure 7. Given the function f_p, the threshold function g_p can be computed easily. Given a query $[x_l, x_r]$ and a threshold parameter $\tau > 0$, a point p is reported, i.e., p lies in $[x_l, x_r]$ with probability at least τ, if $g_p(x_l, x_r) \ge \tau$.

Let $U = \langle b_1 < \ldots < b_u \rangle$, $u \le sn$, be the set of breakpoints in the pdf's of the point set P. Let $\mathcal{R} = \{r_1, \ldots, r_t\}$, $t = O(s^2 n)$, be the set of rectangles in the xy-projections of the threshold functions g_p, for $p \in P$; vertices of $\mathcal{R}$ belong to the set $U \times U$. For each rectangle $r_i \in \mathcal{R}$, which is the projection of a rectangular piece of g_p, let φ_i be the plane that contains that rectangular piece of g_p; we associate the point p with the rectangle r_i and the plane φ_i. Given a query interval $[x_l, x_r]$ and a probability τ, among all the rectangles r_i of $\mathcal{R}$ that contain the point (x_l, x_r), we wish to report those for which the plane φ_i lies above the point $(x_l, x_r, \tau) \in \mathbb{R}^3$. If a rectangle r_i is reported, then the point of P associated with r_i is returned. We build a structure on $\mathcal{R}$ as follows.

We cover the interval $[b_1, b_u]$ by a family $\mathcal{I}$ of $O(n)$ canonical intervals, by building a minimum-height binary search tree on U, so that any interval $[b_i, b_j]$ can be partitioned in $O(\log n)$ canonical intervals; a

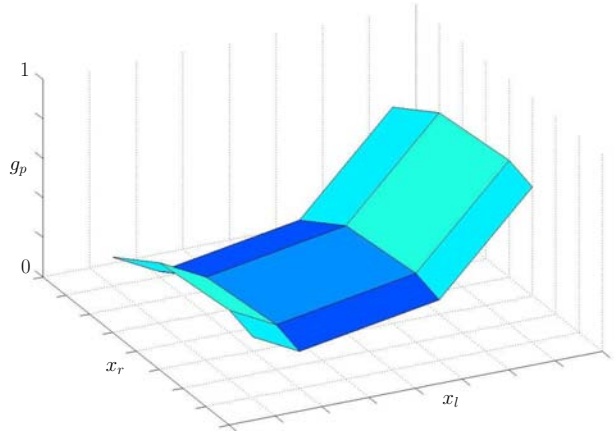

Figure 7: $\Pr[p \in [x_l, x_r]]$ is a bivariate piecewise linear function in x_l and x_r. It consists of s^2 pieces and each piece covers a rectangular region in the $x_l x_r$-plane.

point $b \in \mathbb{R}$ lies in $O(\log n)$ canonical intervals. Set $\mathcal{C} = \mathcal{I} \times \mathcal{I}$ to be a set of $O(n^2)$ *canonical rectangles* in the $x_l x_r$-plane; $\mathcal{C}$ is not constructed explicitly. A rectangle $r_i \in \mathcal{R}$ can be partitioned into a set $\mathcal{C}[r_i]$ of $O(\log^2 n)$ canonical rectangles. For each rectangle $C \in \mathcal{C}$, let $\Phi_C = \{\varphi_i \mid C \in \mathcal{C}[r_i], r_i \in \mathcal{R}\}$. By construction, $\sum_C |\Phi_C| = O(n \log^2 n)$. For each $C \in \mathcal{C}$ such that $\Phi_C \neq \emptyset$, we build the structure by Afshani and Chan [1] on Φ_C for answering halfspace range-reporting queries. Since this structure has linear size, the total size over all of the canonical rectangles is $O(n \log^2 n)$, and it takes $O(n \log^3 n)$ expected time to build them.

Given a query interval $[a, b]$ and a probability threshold τ, we first find the sets $\mathcal{I}_a, \mathcal{I}_b \subset \mathcal{I}$, $O(\log n)$ canonical intervals each, that contain a, b, respectively. $\mathcal{I}_a \times \mathcal{I}_b \subset \mathcal{C}$ is the set of canonical rectangles that contain the point $(a, b) \in \mathbb{R}^2$. For each such canonical rectangle C, we query the structure built on C to report all planes of Φ_C that lies above the point (a, b, τ). If a plane is reported, then we return the point of P associated with it. By construction, each point is reported only once. The total time spent in reporting all k points is $O(\log^3 n + k)$. Hence, we conclude the following.

Theorem 4.1 *Given a set P of n uncertain points in $\mathbb{R}$, each associated with a histogram having at most s pieces, P can be preprocess in expected $O(n \log^3 n)$ time into a structure of size $O(n \log^2 n)$, so that for a query interval I and a probability τ, it can report in $O(\log^3 n + k)$ time all k points of P that lie in I with probability at least τ.*

Dynamization. To make our structure dynamic, we replace the static 3D halfspace searching structure of [1] with the dynamic version [8, 9], which uses $O(n \log n)$ space, supports updates in $O(\operatorname{polylog} n)$ time amortized, and answers queries in $O(\log^2 n + k)$ time. By using this structure for each nonempty canonical rectangle, we obtain the following:

Theorem 4.2 *Given a set P of n uncertain points in $\mathbb{R}$, each associated with a histogram having at most s pieces, P can be maintained in a fully dynamic data structure of size $O(n \operatorname{polylog} n)$ such that a range query with any probability threshold can be answered in $O(\operatorname{polylog} n + k)$ time. This structure supports insertions and deletions of uncertain points in $O(\operatorname{polylog} n)$ time amortized.*

Remark. If s is not a constant, all our space and query bounds in this section still hold by simply replacing n by $s^2 n$, since each uncertain point generates a piecewise linear function with s^2 pieces. The update time of Theorem 4.2 becomes $O(s^2 \operatorname{polylog} n)$.

5 Conclusion

In this paper we have studied the problem of range searching on uncertain data. Our data structures have linear or near-linear sizes and support range queries in logarithmic (or polylogarithmic) time. These results significantly improve upon the previous ones on this problem. Although our results are mostly theoretical in nature, we believe that some of the structures, such as the one in Section 3, are simple enough to be of practical interests. For the other more complicated ones, some of the ideas (such as the geometric reductions) could be borrowed to devise more practical data structures.

We conclude by mentioning two open problems:

(i) How fast can a range query on uncertain points in $\mathbb{R}^2$ be answered? We can extend our approach in this paper to this case but the problem reduces to the so-called *semialgebraic range searching* in 4D. A query can be answered in sublinear time using linear space [3], but more efficient data structures remain elusive.

(ii) How fast can a nearest neighbor query on uncertain points be answered? Here we may wish to report all points whose probability of being the nearest neighbor to the query point is larger than a threshold τ, or return the point with the largest probability of being the nearest neighbor. A few heuristics based on R-trees have been proposed in [21], but no provably good solutions are known. Unlike range searching, we need to consider the interplay between the uncertain points when answering a nearest neighbor query, which seems to make the problem considerably more difficult.

References

[1] P. Afshani and T. M. Chan. Optimal halfspace range reporting in three dimensions. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 180–186, 2009.

[2] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

[3] P. K. Agarwal and J. Matoušek. On range searching with semialgebraic sets. *Discrete and Computational Geometry*, 11:393–418, 1994.

[4] P. K. Agarwal and M. Sharir. Davenport-Schinzel sequences and their geometric applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 1–47. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[5] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2004.

[6] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *Journal of Algorithms*, 1:301–358, 1980.

[7] T. M. Chan. Random sampling, halfspace range reporting, and construction of $(\leq k)$-levels in three dimensions. *SIAM Journal on Computing*, 30(2):561–575, 2000.

[8] T. M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 1196–1202, 2006.

[9] T. M. Chan. Personal communication, 2009.

[10] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.

[11] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25(1):76–90, 1985.

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

[13] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proc. International Conference on Very Large Data Bases*, pages 876–887, 2004.

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

[15] N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: Diamonds in the dirt. *Communications of the ACM*, 52(7):86–96, 2009.

[16] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. International Conference on Very Large Data Bases*, pages 864–875, 2004.

[17] H. Davenport and A. Schinzel. A combinatorial problem connected with differential equations. *American Journal of Mathematics*, 87:684–689, 1965.

[18] J. Y. Halpern. *Reasoning about Uncertainty*. The MIT Press, 2003.

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

[20] L. N. Kanal and J. F. Lemmer. *Uncertainty in Artificial Intelligence*. Elsevier Science Pub. Co., Inc., New York, 1986.

[21] X. Lian and L. Chen. Probabilistic ranked queries in uncertain databases. In *Proc. Conference on Extending Database Technology*, 2008.

[22] V. Ljosa and A. K. Singh. APLA: Indexing arbitrary probability distributions. In *Proc. IEEE International Conference on Data Engineering*, pages 946–955, 2007.

[23] G. Nivasch. Improved bounds and new techniques for Davenport-Schinzel sequences and their generalizations. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 1–10, 2009.

[24] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch. Indexing uncertain categorical data. In *Proc. IEEE International Conference on Data Engineering*, pages 616–625, 2007.

[25] Y. Tao, X. Xiao, and R. Cheng. Range search on multidimensional uncertain data. *ACM Transactions on Database Systems*, 32(3):15, 2007.