Range Queries on Uncertain Data

Jian Li[1] and Haitao Wang[2(✉)]

[1] Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing 100084, China
lijian83@mail.tsinghua.edu.cn
[2] Department of Computer Science, Utah State University, Logan, UT 84322, USA
haitao.wang@usu.edu

Abstract. Given a set P of n uncertain points on the real line, each represented by its one-dimensional probability density function, we consider the problem of building data structures on P to answer range queries of the following three types: (1) top-1 query: find the point in P that lies in I with the highest probability, (2) top-k query: given any integer $k \leq n$ as part of the query, return the k points in P that lie in I with the highest probabilities, and (3) threshold query: given any threshold τ as part of the query, return all points of P that lie in I with probabilities at least τ. We present data structures for these range queries with linear or near linear space and efficient query time.

1 Introduction

In this paper, we study range queries on uncertain data. Let $\mathbb{R}$ be any real line (e.g., the x-axis). In the (traditional) deterministic version of this problem, we are given a set P of n deterministic points on $\mathbb{R}$, and the goal is to build a data structure (also called "index" in database) such that given a range, specified by an interval $I \subseteq \mathbb{R}$, one point (or all points) in I can be retrieved efficiently. It is well known that a simple solution for this problem is a binary search tree over all points which is of linear size and can support logarithmic (plus output size) query time. However, in many applications, the location of each point may be uncertain and the uncertainty is represented in the form of probability distributions [3,5,11,19,20]. In particular, an *uncertain point* p is specified by its probability density function (pdf) $f_p : \mathbb{R} \rightarrow \mathbb{R}^+ \cup \{0\}$.

Let P be the set of n uncertain points in $\mathbb{R}$ (with pdfs specified as input). Our goal is to build data structures to quickly answer range queries on P. In this paper, we consider the following three types of range queries, each of which involves a query interval $I = [x_l, x_r]$. For any point $p \in P$, we use $\Pr[p \in I]$ to denote the probability that p is contained in I.

J. Li was supported in part by the National Basic Research Program of China Grant 2011CBA00300,2011CBA00301, the National Natural Science Foundation of China Grant 61202009,61033001, 61361136003. H. Wang was supported in part by NSF under Grant CCF-1317143. Part of the work on this paper by H. Wang was carried out when he was visiting IIIS at Tsinghua University.

H.-K. Ahn and C.-S. Shin (Eds.): ISAAC 2014, LNCS 8889, pp. 326–337, 2014.
DOI: 10.1007/978-3-319-13075-0_26

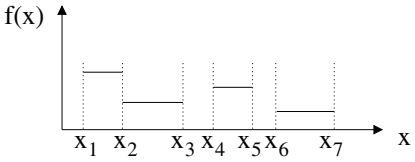

Fig. 1. The pdf of an uncertain point

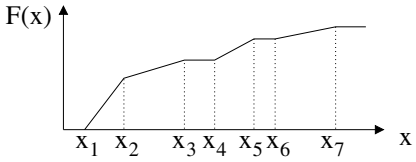

Fig. 2. The cdf of the uncertain point in Fig. 1

Top-1 query: Return the point p of P such that $\Pr[p \in I]$ is the largest.
Top-k query: Given any integer k, $1 \leq k \leq n$, as part of the query, return the k points p of P such that $\Pr[p \in I]$ are the largest.
Threshold query: Given a threshold τ, as part of the query, return all points p of P such that $\Pr[p \in I] \geq \tau$.

We assume f_p is a step function, i.e., a *histogram* consisting of at most c pieces for some integer $c \geq 1$ (e.g., see Fig. 1). More specifically, $f_p(x) = y_i$ for $x_{i-1} \leq x < x_i$, $i = 1, \ldots, c$, with $x_0 = -\infty$, $x_c = \infty$, and $y_1 = y_c = 0$. We assume c is a constant. The cumulative distribution function (cdf) $F_p(x) = \int_{-\infty}^{x} f_p(t)dt$ is a monotone piecewise-linear function consisting of c pieces (e.g., see Fig. 2). Note that $F_p(+\infty) = 1$, and for any interval $I = [x_l, x_r]$ the probability $\Pr[p \in I]$ is $F_p(x_r) - F_p(x_l)$. As discussed in [2], the histogram model can be used to approximate most pdfs with arbitrary precision in practice, including the *discrete* pdf where each uncertain point can only appear in a finite number of locations.

We also study an important special case where the pdf f_p is a uniform distribution function, i.e., f is associated with an interval $[x_l(p), x_r(p)]$ such that $f_p(x) = 1/(x_r(p) - x_l(p))$ if $x \in [x_l(p), x_r(p)]$ and $f_p(x) = 0$ otherwise. Clearly, the cdf $F_p(x) = (x - x_l(p))/(x_r(p) - x_l(p))$ if $x \in [x_l(p), x_r(p)]$, $F_p(x) = 0$ if $x \in (-\infty, x_l(p))$, and $F_p(x) = 1$ if $x \in (x_r(p), +\infty)$. Uniform distributions have been used as a major representation of uncertainty in some previous work (e.g., [10,11,16]). We refer to this special case the *uniform case* and the more general case where f_p is a histogram distribution function as the *histogram* case.

Throughout the paper, we will always use $I = [x_l, x_r]$ to denote the query interval. The query interval I is *unbounded* if either $x_l = -\infty$ or $x_r = +\infty$. For the threshold query, we will always use m to denote the output size of the query, i.e., the number of points p of P such that $\Pr[p \in I] \geq \tau$.

Range reporting on uncertain data has many applications [2,11,15,18–20], As shown in [2], our problems are also useful even in some applications that involve only deterministic data. For example, consider the movie rating system in IMDB where each reviewer gives a rating from 1 to 10. A top-k query on $I = [7, +\infty)$ would find "the k movies such that the percentages of the ratings they receive at least 7 are the largest"; a threshold query on $I = [7, +\infty)$ and $\tau = 0.85$ would find "all the movies such that at least 85% of the ratings they receive are larger than or equal to 7". Note that in the above examples the interval I is unbounded, and thus, it would also be interesting to have data structures particularly for quickly answering queries with unbounded query intervals.

1.1 Previous Work

The threshold query was first introduced by Cheng *et al.* [11]. Using R-trees, they [11] gave heuristic algorithms for the histogram case, without any theoretical performance guarantees. For the uniform case, if τ is fixed for any query, they proposed a data structure of $O(n\tau^{-1})$ size with $O(\tau^{-1}\log n + m)$ query time [11]. These bounds depend on τ^{-1}, which can be arbitrarily large.

Agarwal *et al.* [2] made a significant theoretical step on solving the threshold queries for the histogram case. If τ is fixed, their approach can build an $O(n)$ size data structure in $O(n \log n)$ time, with $O(m+\log n)$ query time. If the threshold τ is not fixed, they built an $O(n \log^2 n)$ size data structure in expected $O(n \log^3 n)$ time that can answer each query in $O(m + \log^3 n)$ time. Tao *et al.* [19,20] considered the threshold queries in two and higher dimensions. They provided heuristic results and a query takes $O(n)$ time in the worst case. Recently, Abdullah *et al.* [1] extended the notion of *geometric coresets* to uncertain data for range queries in order to obtain efficient approximate solutions.

As discussed in [2], our uncertain model is an analogue of the *attribute-level uncertainty model* in the probabilistic database literature. Another popular model is the *tuple-level uncertainty model* [5,12,21], where a tuple has fixed attribute values but its existence is uncertain. The range query under the latter model is much easier since a d-dimensional range searching over uncertain data can be transformed to a $(d+1)$-dimensional range searching problem over certain data [2,21]. In contrast, the problem under the former model is more challenging, partly because it is unclear how to transform it to an instance on certain data.

1.2 Our Results

We say the *complexity* of a data structure is $O(A, B)$ if it is of size $O(B)$ and can be built in $O(A)$ time. For the histogram case, we build data structures on P for answering queries with unbounded query intervals, and the complexities for the three type of queries are all $O(n \log n, n)$. The top-1 query time is $O(\log n)$; the top-k query time is $O(k)$ if $k = \Omega(\log n \log\log n)$ and $O(\log n + k \log k)$ otherwise; the threshold query time is $O(\log n + m)$. Note that we consider c as a constant, otherwise all our results hold by replacing n by $c \cdot n$.

For the uniform case, we also present data structures for bounded query intervals. For the top-1 query, the complexity of our data structure is $O(n \log n, n)$, with query time $O(\log n)$. For other two queries, the data structure complexities are both $O(n \log^2 n, n \log n)$; the top-k query time is $O(k)$ if $k = \Omega(\log n \log\log n)$ and $O(\log n + k \log k)$ otherwise, and the threshold query time is $O(\log n + m)$.

For the histogram case with bounded query intervals, Agarwal *et al.* [2] built a data structure of size $O(n \log^2 n)$ in expected $O(n \log^3 n)$ time, which can answer each threshold query in $O(m+\log^3 n)$ time. Our results for the threshold queries are clearly better than the above solution for the uniform case and the histogram case with unbounded query intervals. Further, our algorithms are deterministic.

In Section 2, we give some observations. We discuss the uniform case and the histogram case in Sections 3 and 4, respectively. Due to the space limit, all lemma proofs are omitted and can be found in the full paper.

2 Preliminaries

For each uncertain point p, we call $\Pr[p \in I]$ the *I-probability* of p. Let $\mathcal{F}$ be the set of the cdfs of all points of P. Since each cdf is an increasing piecewise linear function, depending on the context, $\mathcal{F}$ may also refer to the set of the $O(n)$ line segments of all cdfs. Recall that $I = [x_l, x_r]$ is the query interval.

Lemma 1. *If $x_l = -\infty$, then for any uncertain point p, $\Pr[p \in I] = F_p(x_r)$.*

Let L be the vertical line with x-coordinate x_r. Since each cdf F_p is a monotonically increasing function, there is only one intersection between F_p and L. It is easy to know that for each cdf F_p of $\mathcal{F}$, the y-coordinate of the intersection of F_p and L is $F_p(x_r)$, which is the I-probability of p by Lemma 1. For each point in any cdf of $\mathcal{F}$, we call its y-coordinate the *height* of the point.

In the uniform case, each cdf F_p has three segments: the leftmost one is a horizontal segment with two endpoints $(-\infty, 0)$ and $(x_l(p), 0)$, the middle one, whose slope is $1/(x_r(p) - x_l(p))$, has two endpoints $(x_l(p), 0)$ and $(x_r(p), 1)$, and the rightmost one is a horizontal segment with two endpoints $(x_r(p), 1)$ and $(+\infty, 1)$. We transform each F_p to the line l_p containing the middle segment of F_p. Consider an unbounded interval I with $x_l = -\infty$. We can use l_p to compute $\Pr[p \in I]$ in the following way. Suppose the height of the intersection of L and l_p is y. Then, $\Pr[p \in I] = 0$ if $y < 0$, $\Pr[p \in I] = y$ if $0 \le y \le 1$, $\Pr[p \in I] = 1$ if $y > 1$. Therefore, once we know $l_p \cap L$, we can obtain $\Pr[p \in I]$ in constant time. Hence, we can use l_p instead of F_p to determine the I-probability of p. The advantage of using l_p is that lines are usually easier to deal with than line segments. Below, with a little abuse of notation, for the uniform case we simply use F_p to denote the line l_p for any $p \in P$ and now $\mathcal{F}$ is a set of lines.

Fix the query interval $I = [x_l, x_r]$. For each i, $1 \le i \le n$, denote by p_i the point of P whose I-probability is the i-th largest. Based on the above discussion, we obtain Lemma 2, which holds for both the histogram and uniform cases.

Lemma 2. *If $x_l = -\infty$, then for each $1 \le i \le n$, p_i is the point of P such that $L \cap F_{p_i}$ is the i-th highest among the intersections of L and all cdfs of $\mathcal{F}$.*

Suppose $x_l = -\infty$. Based on Lemma 2, to answer the top-1 query on I, it is sufficient to find the cdf of $\mathcal{F}$ whose intersection with L is the highest; to answer the top-k query, it is sufficient to find the k cdfs of $\mathcal{F}$ whose intersections with L are the highest; to answer the threshold query on I and τ, it is sufficient to find the cdfs of $\mathcal{F}$ whose intersections with L have y-coordinates $\ge \tau$.

Half-Plane Range Reporting. As the half-plane range reporting data structure [9] is important for our later developments, we briefly discuss it in the dual setting. Let S be a set of n lines. Given any point q, the goal is to report all lines of S that are above q. An $O(n)$-size data structure can be built in $O(n \log n)$ time that can answer each query in $O(\log n + m')$ time, where m' is the number of lines above the query point q [9]. The data structure can be built as follows.

Let U_S be the upper envelope of S (e.g., see Fig. 3). We represent U_S as an array of lines $l_1, l_2, \ldots, l_h$ ordered as they appear on U_S from left to right. For

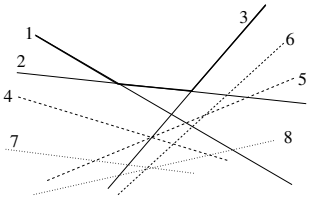

Fig. 3. Partitioning S into three layers: $L_1(S) = \{1, 2, 3\}$, $L_2(S) = \{4, 5, 6\}$, $L_3(S) = \{7, 8\}$. The thick polygonal chain is the upper envelope of S.

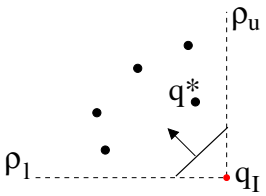

Fig. 4. Dragging a segment of slope 1 out of the corner at q_I: q^* is the first point that will be hit by the segment

each line l_i, l_{i-1} is its *left neighbor* and l_{i+1} is its *right neighbor*. We partition S into a sequence $L_1(S), L_2(S), \ldots$, of subsets, called *layers* (e.g., see Fig. 3). The first layer $L_1(S) \subseteq S$ consists of the lines that appear on U_S. For $i > 1$, $L_i(S)$ consists of the lines that appear on the upper envelope of the lines in $S \setminus \bigcup_{j=1}^{i-1} L_j(S)$. Each layer $L_i(S)$ is represented in the same way as U_S. To answer a half-plane range reporting query on a point q, let $l(q)$ be the vertical line through q. We first determine the line l_i of $L_1(S)$ whose intersection with $l(q)$ is on the upper envelope of $L_1(S)$, by doing binary search on the array of lines of $L_1(S)$. Then, starting from l_i, we walk on the upper envelope of $L_1(S)$ in both directions to report the lines of $L_1(S)$ above the point q, in linear time with respect to the output size. Next, we find the line of $L_2(S)$ whose intersection with $l(q)$ is on the upper envelope of $L_2(S)$. We use the same procedure as for $L_1(S)$ to report the lines of $L_2(S)$ above q. Similarly, we continue on the layers $L_3(S), L_4(S), \ldots$, until no line is reported in a certain layer. By using fractional cascading [7], after determining the line l_i of $L_1(S)$ in $O(\log n)$ time by binary search, the data structure [9] can report all lines above q in constant time each.

For any vertical line l, for each layer $L_i(S)$, denote by $l_i(l)$ the line of $L_i(S)$ whose intersection with l is on the upper envelope of $L_i(S)$. By fractional cascading [7], we have the following lemma for the data structure [9].

Lemma 3. [7,9] *For any vertical line l, after the line $l_1(l)$ is known, we can obtain the lines $l_2(l), l_3(l), \ldots$ in this order in $O(1)$ time each.*

3 The Uniform Distribution

Recall that $\mathcal{F}$ is a set of lines in the uniform distribution.

3.1 Queries with Unbounded Intervals

We first discuss the unbounded case where $I = [x_l, x_r]$ is unbounded and some techniques introduced here will also be used later for the bounded case. Without loss of generality, we assume $x_l = -\infty$, and the other case where $x_r = +\infty$ can be solved similarly. Recall that L is the vertical line with x-coordinate x_r.

For top-1 queries, by Lemma 2, we only need to maintain the upper envelope of $\mathcal{F}$, which can be computed in $O(n \log n)$ time and $O(n)$ space. For each query, it is sufficient to determine the intersection of L with the upper envelope of $\mathcal{F}$, which can be done in $O(\log n)$ time. Next, we consider top-k queries.

Given I and k, by Lemma 2, it suffices to find the k lines of $\mathcal{F}$ whose intersections with L are the highest, and we let $\mathcal{F}_k$ denote the set of the above k lines. As preprocessing, we build the half-plane range reporting data structure (see Section 2) on $\mathcal{F}$, in $O(n \log n)$ time and $O(n)$ space. Suppose the layers of $\mathcal{F}$ are $L_1(\mathcal{F}), L_2(\mathcal{F}), \ldots$. In the sequel, we compute the set $\mathcal{F}_k$. Let the lines in $\mathcal{F}_k$ be $l^1, l^2, \ldots, l^k$ ordered from top to bottom by their intersections with L.

Let $l_i(L)$ be the line of $L_i(\mathcal{F})$ which intersects L on the upper envelope of the layer $L_i(\mathcal{F})$, for $i = 1, 2, \ldots$. We first compute $l_1(L)$ in $O(\log n)$ time by binary search on the upper envelope of $L_1(\mathcal{F})$. Clearly, l^1 is $l_1(L)$. Next, we determine l^2. Let the set H consist of the following three lines: $l_2(L)$, the left neighbor (if any) of $l_1(L)$ in $L_1(\mathcal{F})$, and the right neighbor (if any) of $l_1(L)$ in $L_1(\mathcal{F})$.

Lemma 4. *l^2 is the line in H whose intersection with L is the highest.*

We refer to H as the *candidate set*. By Lemma 4, we find l^2 in H in $O(1)$ time. We remove l^2 from H, and below we insert at most three lines into H such that l^3 must be in H. Specifically, if l^2 is $l_2(L)$, we insert the following three lines into H: $l_3(L)$, the left neighbor of $l_2(L)$, and the right neighbor of $l_2(L)$. If l^2 is the left (resp., right) neighbor l of $l_1(L)$, we insert the left (resp., right) neighbor of l in $L_1(\mathcal{F})$ into H. By generalizing Lemma 4, we can show l^3 must be in H (the details are omitted). We repeat the same algorithm until we find l^k. To facilitate the implementation, we use a heap to store the lines of H whose "keys" in the heap are the heights of the intersections of L and the lines of H.

Lemma 5. *The set $\mathcal{F}_k$ can be found in $O(\log n + k \log k)$ time.*

We can improve the algorithm to $O(\log n + k)$ time by using the selection algorithm in [14] for sorted arrays. The key idea is that we can implicitly obtain $2k$ sorted arrays of $O(k)$ size each and $\mathcal{F}_k$ can be computed by finding the largest k elements in these arrays. The result is given in Lemma 6 with details omitted.

Lemma 6. *The set $\mathcal{F}_k$ can be found in $O(\log n + k)$ time.*

Remark: The above builds a data structure of $O(n \log n, n)$ complexity that can answer each top-k query in $O(\log n + k)$ time for the uniform unbounded case.

For the threshold query, we are given I and a threshold τ. We again build the half-plane range reporting data structure on $\mathcal{F}$. To answer the query, as discussed in Section 2, we only need to find all lines of $\mathcal{F}$ whose intersections with L have y-coordinates larger than or equal to τ. We first determine the line $l_1(L)$ by doing binary search on the upper envelope of $L_1(\mathcal{F})$. Then, by Lemma 3, we find all lines $l_2(L), l_3(L), \ldots, l_j(L)$ whose intersections have y-coordinates larger than or equal to τ. For each i with $1 \leq i \leq j$, we walk on the upper envelope of $L_i(\mathcal{F})$, starting from $l_i(L)$, on both directions in time linear to the output size to find the lines whose intersections have y-coordinates larger than or equal to τ. Hence, the running time for answering the query is $O(\log n + m)$.

3.2 Queries with Bounded Intervals

Now we assume $I = [x_l, x_r]$ is bounded. Consider any point $p \in P$. Recall that p is associated with an interval $[x_l(p), x_r(p)]$ in the uniform case. Depending on the positions of $I = [x_l, x_r]$ and $[x_l(p), x_r(p)]$, we classify $[x_l(p), x_r(p)]$ and the point p into the following three types with respect to I.

L-type: $[x_l(p), x_r(p)]$ and p are L-type if $x_l \leq x_l(p)$.
R-type: $[x_l(p), x_r(p)]$ and p are R-type if $x_r \geq x_r(p)$.
M-type: $[x_l(p), x_r(p)]$ and p are M-type if $I \subset (x_l(p), x_r(p))$.

Denote by P_L, P_R, and P_M the sets of all L-type, R-type, and M-type of points of P, respectively. In the following, for each kind of query, we will build an data structure such that the different types of points will be searched separately (note that we will not explicitly compute the three subsets P_L, P_R, and P_M). For each point $p \in P$, we refer to $x_l(p)$ as the *left endpoint* of the interval $[x_l(p), x_r(p)]$ and refer to $x_r(p)$ as the *right endpoint*. For simplicity of discussion, we assume no two interval endpoints of the points of P have the same value.

The Top-1 Queries. For any point $p \in P$, denote by $\mathcal{F}_r(p)$ the set of the cdfs of the points of P whose intervals have left endpoints larger than or equal to $x_l(p)$. Again, as discussed in Section 2 we transform each cdf of $\mathcal{F}_r(p)$ to a line. We aim to maintain the upper envelope of $\mathcal{F}_r(p)$ for each $p \in P$. If we compute the n upper envelopes explicitly, we would have an data structure of size $\Omega(n^2)$. To reduce the space, we choose to use the persistent data structure [13] to maintain them implicitly such that data structure size is $O(n)$. The details are given below.

We sort the points of P by the left endpoints of their intervals from left to right, and let the sorted list be $p'_1, p'_2, \ldots, p'_n$. For each i with $2 \leq i \leq n$, observe that the set $\mathcal{F}_r(p'_{i-1})$ has exactly one more line than $\mathcal{F}_r(p'_i)$. If we maintain the upper envelope of $\mathcal{F}_r(p'_i)$ by a balanced binary search tree (e.g., a red-black tree), then by updating it we can obtain the upper envelope of $\mathcal{F}_r(p'_{i-1})$ by an insertion and a number of deletions on the tree, and each tree operation takes $O(\log n)$ time. An easy observation is that there are $O(n)$ tree operations in total to compute the upper envelopes of all sets $\mathcal{F}_r(p'_1), \mathcal{F}_r(p'_2), \ldots, \mathcal{F}_r(p'_n)$. Further, by making the red-black tree persistent [13], we can maintain all upper envelopes in $O(n \log n)$ time and $O(n)$ space. We use $\mathcal{L}$ to denote the above data structure.

We can use $\mathcal{L}$ to find the point of P_L with the largest I-probability in $O(\log n)$ time, as follows. First, we find the point p'_i such that $x_l(p'_{i-1}) < x_l \leq x_l(p'_i)$. It is easy to see that $\mathcal{F}_r(p'_i) = P_L$. Consider the unbounded interval $I' = (-\infty, x_r]$. Consider any point p whose cdf is in $\mathcal{F}_r(p'_i)$. Due to $x_l(p) \geq x_l$, we can obtain that $\Pr[p \in I] = \Pr[p \in I']$. Hence, the point p of $\mathcal{F}_r(p'_i)$ with the largest value $\Pr[p \in I]$ also has the largest value $\Pr[p \in I']$. This implies that we can instead use the unbounded interval I' as the query interval on the upper envelope of $\mathcal{F}_r(p'_i)$, in the same way as in Section 3.1. The persistent data structure $\mathcal{L}$ maintains the upper envelope of $\mathcal{F}_r(p'_i)$ such that we can find in $O(\log n)$ time the point p of $\mathcal{F}_r(p'_i)$ with the largest value $\Pr[p \in I']$.

Similarly, we can build a data structure $\mathcal{R}$ of $O(n)$ space in $O(n \log n)$ time that can find the point of P_R with the largest I-probability in $O(\log n)$ time.

To find the point of P_M with the largest I-probability, the approach for P_L and P_R does not work because we cannot reduce the query to another query with an unbounded interval. Instead, we reduce the problem to a "segment dragging query" by dragging a line segment out of a corner in the plane, as follows.

For each point p of P, we define a point $q = (x_l(p), x_r(p))$ in the plane, and we say that p *corresponds to* q. Similar transformation was also used in [11]. Let Q be the set of the n points defined by the points of P. For the query interval $I = [x_l, x_r]$, we also define a point $q_I = (x_l, x_r)$ (this is different from [11], where I defines a point (x_r, x_l)). If we partition the plane into four quadrants with respect to q_I, then we have the following lemma.

Lemma 7. *The points of P_M correspond to the points of Q that strictly lie in the second quadrant (i.e., the northwest quadrant) of q_I.*

Let ρ_u be the upwards ray originating from q_I and let ρ_l be the leftwards ray originating from q_I. Imagine that starting from the point q_I and towards northwest, we drag a segment of slope 1 with two endpoints on ρ_u and ρ_l respectively, and let q^* be the point of Q hit first by the segment (e.g., see Fig. 4).

Lemma 8. *The point of P that defines q^* is in P_M and has the largest I-probability among all points in P_M.*

Based on Lemma 8, to determine the point of P_M with the largest I-probability, we only need to solve the above query on Q by dragging a segment out of a corner. More specifically, we need to build a data structure on Q to answer the following *out-of-corner segment-dragging queries*: Given a point q, find the first point of Q hit by dragging a segment of slope 1 from q and towards the northwest direction with the two endpoints on the two rays $\rho_u(q)$ and $\rho_l(q)$, respectively, where $\rho_u(q)$ is the upwards ray originating from q and $\rho_l(q)$ is the leftwards ray originating from q. By using Mitchell's result in [17] (reducing the problem to a point location problem), we can build an $O(n)$ size data structure on Q in $O(n \log n)$ time that can answer each such query in $O(\log n)$ time.

Hence, for the uniform case, we can build in $O(n \log n)$ time an $O(n)$ size data structure on P that can answer each top-1 query in $O(\log n)$ time.

The Top-k Queries. To answer a top-k query, we will do the following. First, we find the top-k points in P_L (i.e., the k points of P_L whose I-probabilities are the largest), the top-k points in P_R, and the top-k points in P_M. Then, we find the top-k points of P from the above $3k$ points. Below we build three data structures for computing the top-k points in P_L, P_R, and P_M, respectively.

We first build the data structure for P_L. Again, let $p'_1, p'_2, \ldots, p'_n$ be the list of the points of P sorted by the left endpoints of their intervals from left to right. We construct a complete binary search tree T_L whose leaves from left to right store the n intervals of the points $p'_1, p'_2, \ldots, p'_n$. For each internal node v, let P_v denote

the set of points whose intervals are stored in the leaves of the subtree rooted at v. We build the half-plane range reporting data structure discussed in Section 2 on P_v, denoted by D_v. Since the size of D_v is $|P_v|$, the total size of the data structure T_L is $O(n \log n)$, and T_L can be built in $O(n \log^2 n)$ time.

We use T_L to compute the top-k points in P_L as follows. By the standard approach and using x_l, we find in $O(\log n)$ time a set V of $O(\log n)$ nodes of T_L such that $P_L = \bigcup_{v \in V} P_v$ and no node of V is an ancestor of another node. Then, we can determine the top-k points of P_L in similarly as in Section 3.1. However, since we now have $O(\log n)$ data structures D_v, we need to maintain the candidate sets for all such D_v's. Specifically, after we find the top-1 point in D_v for each $v \in V$, we use a heap H to maintain them where the "keys" are the I-probabilities of the points. Let p be the point of H with the largest key. Clearly, p is the top-1 point of P_L; assume p is from D_v for some $v \in V$. We remove p from H and insert at most three new points from D_v into H, in a similar way as in Section 3.1. We repeat the same procedure until we find all top-k points of P_L.

To analyze the running time, for each node $v \in V$, we can determine in $O(\log n)$ time the line in the first layer of D_v whose intersection with L is on the upper envelope of the first layer, and subsequent operations on D_v each takes $O(1)$ time due to fractional cascading. Hence, the total time for this step in the entire algorithm is $O(\log^2 n)$. However, we can do better by building a fractional cascading structure [7] on the first layers of D_v for all nodes v of the tree T_L. In this way, the above step only takes $O(\log n)$ time in the entire algorithm, i.e., do binary search only at the root of T_L. In addition, building the heap H initially takes $O(\log n)$ time. Note that the additional fractional cascading structure on T_L does not change the size and construction time of T_L asymptotically [7]. The entire query algorithm has $O(k)$ operations on H in total and the size of H is $O(\log n + k)$. Hence, the total time for finding the top-k points of P_L is $O(\log n + k \log(k + \log n))$, which is $O(\log n + k \log k)$ by Lemma 9.

Lemma 9. $\log n + k \log(k + \log n) = O(\log n + k \log k)$.

If $k = \Omega(\log n \log \log n)$, we have a better result in Lemma 10. Note that comparing with Lemma 6, we need to use other techniques to obtain Lemma 10 since the problem here involves $O(\log n)$ half-plane range reporting data structures D_v while Lemma 6 only needs to deal with one such data structure.

Lemma 10. *If $k = \Omega(\log n \log \log n)$, we can compute the top-k points in P_L in $O(k)$ time.*

To compute the top-k points of P_R, we build a similar data structure T_R, in a symmetric way as T_L, and we omit the details.

Finally, to compute the top-k points in P_M, we do the following transformation. For each point $p \in P$, we define a point $q = (x_l(p), x_r(p), 1/(x_r(p) - x_l(p))$ in the 3-D space with x-, y-, and z-axes. Let Q be the set of all points in the 3-D space thus defined. Let the query interval I define an unbounded query box (or 3D rectangle) $B_I = (-\infty, x_l) \times (x_r, +\infty) \times (-\infty, +\infty)$. Similar to Lemma 7 in Section 3.1, the points of P_M correspond exactly to the points of $Q \cap B_I$. Further,

the top-k points of P_M correspond to the k points of $Q \cap B_I$ whose z-coordinates are the largest. Denote by Q_I the k points of $Q \cap B_I$ whose z-coordinates are the largest. Below we build a data structure on Q for computing the set Q_I for any query interval I and thus finding the top-k points of P_M.

We build a complete binary search tree T_M whose leaves from left to right store all points of Q ordered by the increasing x-coordinate. For each internal node v of T_M, we build an auxiliary data structure D_v as follows. Let Q_v be the set of the points of Q stored in the leaves of the subtree of T_M rooted at v. Suppose all points of Q_v have x-coordinates less than x_l. Let Q'_v be the points of Q_v whose y-coordinates are larger than x_r. The purpose of the auxiliary data structure D_v is to report the points of Q'_v in the decreasing z-coordinate order in constant time each after the point of q_v is found, where q_v is the point of Q'_v with the largest z-coordinate. To achieve this goal, we use the data structure given by Chazelle [8] (the one for Subproblem P1 in Section 5), and the data structure is a *hive graph* [6], which can be viewed as the preliminary version of the fractional cascading techniques [7]. By using the result in [8], we can build such a data structure D_v of size $O(|Q_v|)$ in $O(|Q_v| \log |Q_v|)$ time that can first compute q_v in $O(\log |Q_v|)$ time and then report other points of Q'_v in the decreasing z-coordinate order in constant time each. Since the size of D_v is $|Q_v|$, the size of the tree T_M is $O(n \log n)$, and T_M can be built in $O(n \log^2 n)$ time.

Using T_M, we find the set Q_I as follows. We first determine the set V of $O(\log n)$ nodes of T_M such that $\bigcup_{v \in V} Q_v$ consists of all points of Q whose x-coordinates less than x_l and no point of V is an ancestor of another point of V. Then, for each node $v \in V$, by using D_v, we find q_v, i.e., the point of Q_v with the largest z-coordinate, and insert q_v into a heap H, where the key of each point is its z-coordinate. We find the point in H with the largest key and remove it from H; denote the above point by q'_1. Clearly, q'_1 is the point of Q_I with the largest z-coordinate. Suppose q'_1 is in a node $v \in V$. We proceed on D_v to find the point of Q_v with the second largest z-coordinate and insert it into H. Now the point of H with the largest key is the point of Q_I with the second largest z-coordinate. We repeat the above procedure until we find all k points of Q_I.

To analyze the query time, finding the set V takes $O(\log n)$ time. For each node $v \in V$, the search for q_v on D_v takes $O(\log n)$ time plus the time linear to the number of points of D_v in Q_I. Hence, the total time for searching q_v for all vertices $v \in V$ is $O(\log^2 n)$ time. Similarly as before, we can remove a logarithmic factor by building a fractional cascading structure on the nodes of T_M for searching such points q_v's, in exactly the same way as in [6]. With the help of the fractional cascading structure, all these q_v's for $v \in V$ can be found in $O(\log n)$ time. Note that building the fractional cascading structure does not change the construction time and the size of T_M asymptotically [6]. In addition, building the heap H initially takes $O(\log n)$ time. In the entire algorithm there are $O(k)$ operations on H in total and the size of H is always bounded by $O(k + \log n)$. Therefore, the running time of the query algorithm is $O(\log n + k \log(k + \log n))$, which is $O(\log n + k \log k)$ by Lemma 9.

Using similar techniques as in Lemma 10, we obtain the following result.

Lemma 11. *If $k = \Omega(\log n \log\log n)$, we can compute the top-k points in P_M in $O(k)$ time.*

In summary, for the uniform case, we can build in $O(n \log^2 n)$ time an $O(n \log n)$ size data structure on P that can answer each top-k query in $O(k)$ time if $k = \Omega(\log n \log\log n)$ and $O(k \log k + \log n)$ time otherwise.

For the threshold queries, we build the same data structure as for the top-k queries, i.e., the three trees T_L, T_M, and T_R. The query algorithmic scheme is also similar. We omit the details.

4 The Histogram Distribution

In this section, we present our data structures for the histogram case, where $I = [x_l, x_r]$ is unbounded. Again, we assume w.l.o.g. that $x_l = -\infty$. Recall that L is the vertical line with x-coordinate x_r. In the histogram case, the cdf of each point $p \in P$ has c pieces; recall that we assumed c is a constant, and thus $\mathcal{F}$ is still a set of $O(n)$ line segments. Note that Lemmas 1 and 2 are still applicable.

For the top-1 queries, as in Section 3.1 it is sufficient to maintain the upper envelope of $\mathcal{F}$. Although $\mathcal{F}$ now is a set of line segments, its upper envelope is still of size $O(n)$ and can be computed in $O(n \log n)$ time [4]. Given the query interval I, we can compute in $O(\log n)$ time the cdf of $\mathcal{F}$ whose intersection with L is on the upper envelope of $\mathcal{F}$.

For the threshold query, as discussed in Section 2 we only need to find the cdfs of $\mathcal{F}$ whose intersections with L have y-coordinates at least τ. Let q_I be the point (x_r, τ) on L. A line segment is *vertically above* q_I if the segment intersects L and the intersection is at least as high as q_I. Hence, to answer the threshold query on I, it is sufficient to find the segments of $\mathcal{F}$ that are vertically above q_I. Agarwal *et al.* [2] gave the following result on the *segment-below-point queries*. For a set S of $O(n)$ line segments in the plane, a data structure of $O(n)$ size can be computed in $O(n \log n)$ time that can report the segments of S vertically *below* a query point q in $O(m' + \log n)$ time, where m' is the output size. In our problem, we need a data structure on $\mathcal{F}$ to solve the *segments-above-point queries*, which can be solved by using the similar approach as [2]. Therefore, we can build in $O(n \log n)$ time an $O(n)$ data structure on P that can answer each threshold query with an unbounded query interval in $O(m + \log n)$ time.

For the top-k queries, we only need to find the k segments of $\mathcal{F}$ whose intersections with L are the highest. To this end, we can slightly modify the data structure for the segment-below-point queries in [2]. The details are omitted.

References

1. Abdullah, A., Daruki, S., Phillips, J.: Range counting coresets for uncertain data. In: The 29th Symposium on Computational Geometry, SoCG, pp. 223–232 (2013)
2. Agarwal, P., Cheng, S.W., Tao, Y., Yi, K.: Indexing uncertain data. In: Proc. of the 28th Symposium on Principles of Database Systems, PODS, pp. 137–146 (2009)

3. Agarwal, P., Efrat, A., Sankararaman, S., Zhang, W.: Nearest-neighbor searching under uncertainty. In: Proc. of the 31st Symposium on Principles of Database Systems, PODS, pp. 225–236 (2012)
4. Agarwal, P., Sharir, M.: Red-blue intersection detection algorithms, with applications to motion planning and collision detection. SIAM Journal on Computing **19**, 297–321 (1990)
5. Agrawal, P., Benjelloun, O., Sarma, A.D., Hayworth, C., Nabar, S., Sugihara, T., Widom, J.: Trio: a system for data, uncertainty, and lineage. In: Proc. of the 32nd International Conference on Very Large Data Bases, VLDB, pp. 1151–1154 (2006)
6. Chazelle, B.: Filtering search: A new approach to query-answering. SIAM Journal on Computing **15**(3), 703–724 (1986)
7. Chazelle, B., Guibas, L.: Fractional cascading: I. A data structuring technique. Algorithmica **1**(1), 133–162 (1986)
8. Chazelle, B., Guibas, L.: Fractional cascading: II. Applications. Algorithmica **1**(1), 163–191 (1986)
9. Chazelle, B., Guibas, L., Lee, D.: The power of geometric duality. BIT **25**, 76–90 (1985)
10. Cheng, R., Chen, J., Mokbel, M., Chow, C.: Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In: IEEE International Conference on Data Engineering, ICDE, pp. 973–982 (2008)
11. Cheng, R., Xia, Y., Prabhakar, S., Shah, R., Vitter, J.: Efficient indexing methods for probabilistic threshold queries over uncertain data. In: Proc. of the 30th International Conference on Very Large Data Bases, VLDB, pp. 876–887 (2004)
12. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. The VLDB Journal, 523–544 (2006)
13. Driscoll, J., Sarnak, N., Sleator, D., Tarjan, R.: Making data structures persistent. Journal of Computer and System Sciences **38**(1), 86–124 (1989)
14. Frederickson, G., Johnson, D.: The complexity of selection and ranking in $X + Y$ and matrices with sorted columns. Journal of Computer and System Sciences **24**(2), 197–208 (1982)
15. Knight, A., Yu, Q., Rege, M.: Efficient range query processing on uncertain data. In: Proce. of IEEE International Conference on Information Reuse and Integration, pp. 263–268 (2011)
16. Li, J., Deshpande, A.: Ranking continuous probabilistic datasets. Proceedings of the VLDB Endowment **3**(1), 638–649
17. Mitchell, J.: L_1 shortest paths among polygonal obstacles in the plane. Algorithmica **8**(1), 55–88 (1992)
18. Qi, Y., Jain, R., Singh, S., Prabhakar, S.: Threshold query optimization for uncertain data. In: ACM SIGMOD International Conference on Management of Data (SIGMOD), pp. 315–326 (2010)
19. Tao, Y., Cheng, R., Xiao, X., Ngai, W., Kao, B., Prabhakar, S.: Indexing multidimensional uncertain data with arbitrary probability density functions. In: Proc. of the 31st International Conference on Very Large Data Bases, VLDB, pp. 922–933 (2005)
20. Tao, Y., Xiao, X., Cheng, R.: Range search on multidimensional uncertain data. ACM Transactions on Database Systems (TODS) **32** (2007)
21. Yiu, M., Mamoulis, N., Dai, X., Tao, Y., Vaitis, M.: Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data. IEEE Transactions of Knowledge Data Engineering (TKDE) **21**, 108–122 (2009)