Indexing 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 points in $\mathbb{R}$, each represented by its one-dimensional probability density function (pdf). The goal is to build an index 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 indexing schemes with linear or near-linear space and logarithmic query time. Our schemes support pdf's that are either histograms or more complex ones such as Gaussian or piecewise algebraic. They also extend to the external memory model in which the goal is to minimize the number of disk accesses when querying the index.

Categories and Subject Descriptors

F.2 [**Analysis of algorithms and problem complexity**]: Nonnumerical algorithms and problems; H.3.1 [**Information storage and retrieval**]: Content analysis and indexing—*indexing methods*

General Terms

Algorithms, theory

*P. K. Agawal is supported by NSF under grants CNS-05-40347, CFF-06-35000, and DEB-04-25465, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, by the NIH grant 1P50-GM-08183-01, and by a grant from the US-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).

PODS'09, June 29–July 2, 2009, Providence, Rhode Island, USA.

Keywords

Indexing, uncertain data, range query

1. INTRODUCTION

Indexing a set of points for answering range queries is one of the most widely studied topics in spatial databases [21] and computational geometry [3], with a wide range of applications. Most of the work to date deals with *certain* data, that is, the points have exact coordinates and the goal is to index them so that all points inside a query range can be reported efficiently. Recent years have witnessed a dramatically increasing amount of attention devoted to managing *uncertain* data, due to the observation that many real-world measurements are inherently accompanied with uncertainty [7, 14, 16, 23, 24].

Depending on the application, there are two models of data uncertainty, which we describe in the context of range searching. In the *tuple-level uncertainty model* (also known as *existentially uncertain data*) [7, 16, 25], a point in the data set still has a precise position but its existence is uncertain. Usually it is assumed that each point appears with some probability and all the points are independent. We could relax the independence assumption and introduce "rules" to model the correlation between the points. In the *attribute-level uncertainty model* [14, 23, 24], a point always exists but its location is uncertain, and is modeled as a probability distribution over space. Again we usually assume the points are independent but it is not necessary.

The generally agreed semantics for querying uncertain data is the *thresholding approach* [14, 16], i.e., for a particular threshold τ, retrieve all the tuples that appear in the query range with probability at least τ. Under the tuple-level uncertainty model, a d-dimensional range searching problem over uncertain data simply becomes a $(d+1)$-dimensional range searching problem over certain data, by treating the probability as another dimension. The same observation has also been made in [25]. The independence assumption is irrelevant since each point is considered individually. Of interests is the attribute-level uncertainty model, where each point has a pdf, and we wish to report all points whose probability of being inside the query range is at least τ. This problem is nontrivial, even in one dimension, as it cannot be easily transformed to an instance of range searching on certain data. The naïve approach of examining each point one by one and computing its proba-

bility of being inside the query is obviously very expensive. In this uncertainty model, the independence assumption is again irrelevant with respect to range queries.

Problem definition. 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 indexes also support more complicated pdf's (such as Gaussian or piecewise algebraic), and we will explicitly say so for these indexes.

Given the set P and the associated pdf's, the goal is to build an index 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 indexes 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. [14] and has numerous applications in practice. For example, in a sensor network, a certain measurement, say temperature, of a location may be taken by multiple sensors. Due to various imprecision factors, the readings of these sensors may not be necessarily identical, in which case the temperature of the 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. Cheng et al. [14] 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 an index 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 index also 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. [23, 24] considered the problem in two and higher dimensions, and presented some index 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 [19], but it is still the same as a sequential scan in the worst case.

Cheng et al. [14] 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 $\Theta(\log n + k)$ [12], while the latter takes $\Omega(\sqrt{n})$ time [13]. 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. [22], who considered the problem of indexing uncertain data that are categorical, namely, each random object takes a value from a discrete, unordered domain. The index structures presented there are again heuristic solutions.

Our results. In this paper, we make a significant theoretical step towards understanding the complexity of indexing uncertain data for range queries. We present linear or near-linear size indexing schemes 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 index 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: indexing 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 index for the segments-below-point problem — a linear-size index 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 [14]. In Section 3 we present a simpler index of size $O(n\alpha(n) \log n)$ and query time $O(\log n + k)$. This index extends to more general pdf's, such as Gaussian distributions or other piecewise algebraic pdf's, and they can also be used to approximately count the number of uncertain points lying in the query interval.

For the variable-threshold version, we use a different reduction and show that it can be solved by carefully indexing a number of points in $\mathbb{R}^3$ for answering halfspace range queries. Combining with the very recent result of Afshani and Chan [1] for 3D halfspace range reporting, we obtain an index 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 indexing schemes can be dynamized, supporting insertions and deletions of (uncertain) points with a slight increase in the query time. If the resulting index needs to be stored on disk, the query time is dominated by the number of I/Os (or page reads), where

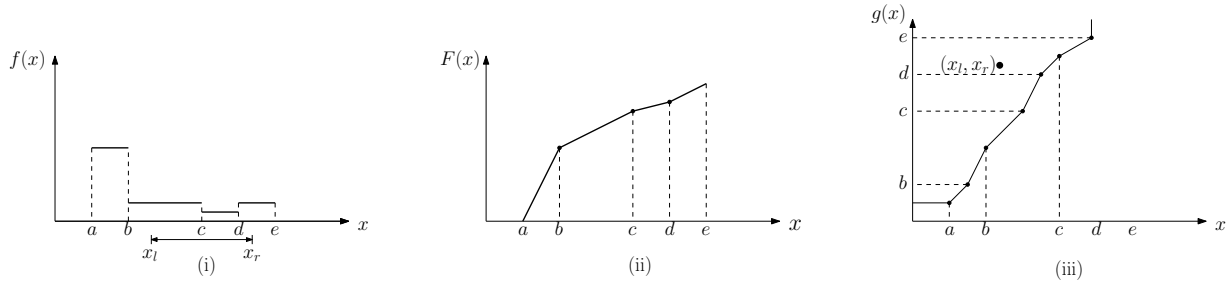

Figure 1. Reduction to the segments-below-point problem: (i) pdf, (ii) cdf, and (iii) threshold function.

each I/O reads a whole disk block of size B. We also show how to externalize our index so that the queries can be answered in an I/O-efficient manner.

2. FIXED-THRESHOLD RANGE QUERIES

We present an optimal index for answering range queries on uncertain data where the probability threshold τ is fixed. Our index 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 three indexing schemes for the latter problem, each of which reduces a segments-below-point query to answering halfplane range-reporting queries. The first index, based on the segment tree, uses linear space and answers a query in $O(\sqrt{n} + k)$ time (Section 2.3). The second one, based on the interval tree, uses $O(n \log n)$ space and answers a query in $O(\log n + k)$ time (Section 2.4). We then combine them to construct an index of linear size with $O(\log n + k)$ query time (Section 2.5). We conclude this section by describing how we make the index dynamic and extend it to the I/O model.

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. Suppose the histogram of f consists of s pieces, and let

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

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 $F(b) - F(a) \ge \tau$. If no such b exists, $g(a)$ is set to ∞; see Figure 1 (iii).

LEMMA 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 $\Pr[p \in [x_l, x_r]] \ge \tau$ simply becomes checking whether $x_r \ge g(x_l)$. 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.

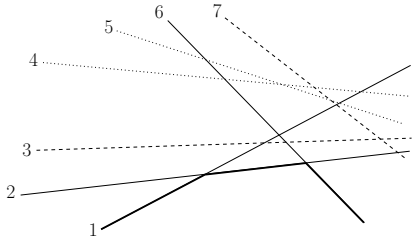

Figure 2. The indexing scheme for a set of lines: thick polygonal chain is the lower envelope of S; $L_1(S) = \{1, 2, 6\}$, $L_2(S) = \{3, 7\}$, $L_3(S) = \{4, 5\}$.

We thus have the following problem at hand: Let S be a set of n segments in $\mathbb{R}^2$. Construct an index 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.

2.2 Half-plane range reporting

We begin by describing an index 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 index with $O(\log n + k)$-time [12]. We briefly describe a variant of this index (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

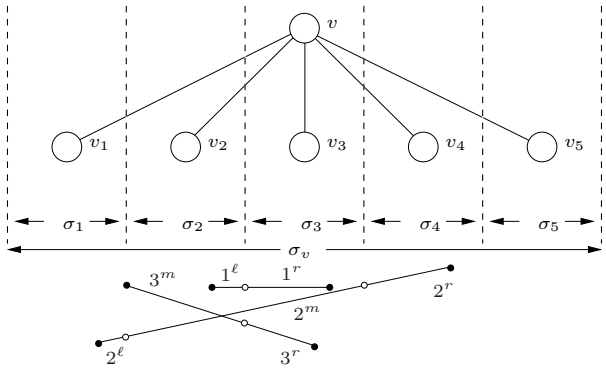

Figure 3. A segment tree node with fanout $r = 5$.

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* [11] 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 $L_1(S)$. The size of the index is linear, and it can be constructed in $O(n \log n)$ time [12]. The statement below is slightly more general than that appeared in [12].

LEMMA 2. *Let S be a set of n lines in the plane. We can construct in $O(n \log n)$ time an index on S 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 index

This subsection describes an index 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. We later (cf. Section 2.5) bootstrap this index to improve the query time to $O(\log n + k)$ while keeping the size linear.

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. At v we partition s 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 ends up with spanning 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. We build the index on $H_v^{i:j}$ described in Section 2.2. Since $\sum |S_v^{i:j}| = O(n \log_r n)$ and the index built for each multi-slab has linear size, the size of the overall index 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 index 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 3. *Let S be a set of segments in the plane. We can construct in $O(n \log n)$ time an index of linear size on S so that all segments of S lying below a query point can be reported in $O(\sqrt{n} + k)$ time.*

Remark. 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. By choosing $r = 2$, we can construct an indexing scheme of size $O(n \log n)$ with query time $O(\log^2 n + k)$. Using fractional cascading the query time can be improved to $O(\log n + k)$.

2.4 Interval-tree based index

We now describe a different index to store S, based on the interval tree, that uses $O(n \log n)$ space and answers a query in time $O(\log n + k)$. An interval tree $\mathcal{T}$ for S is built as follows. Let E be the set of endpoints of the segments of S. We first choose the median of E, and vertically divide the plane into two halves. We store this splitting line at the root of $\mathcal{T}$ and then build its two subtrees for the two halves recursively. Similar to the segment tree, the interval tree $\mathcal{T}$ also hierarchically partitions the plane into $O(n)$ canonical vertical slabs. For a node $u \in \mathcal{T}$, let σ_u be the *slab* associated with u and x_u the x-coordinate of the *splitting line* at u. For the root u of $\mathcal{T}$, σ_u is the entire plane. For each u with children v and w, x_u is the median of the x-coordinates of $E \cap \sigma_v$, and the line $x = x_u$ partitions σ_u into σ_v and σ_w. When $|E \cap \sigma_u| = 1$, we stop the partitioning and make u a leaf. For a node u, let $p(u)$, $\mathsf{Sb}(u)$, $l(u)$, and $r(u)$ denote the parent, sibling, left child, and right child of u, respectively.

A segment $s \in S$ is now stored at the highest node u such that the splitting line $x = x_u$ intersects s. Let $S_u \subseteq S$ be the set of segments stored at u. Since each segment only appears

in one S_u, $\sum_u |S_u| = O(n)$. Each 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 indexing schemes for S^- and S^+ and report $S^-[q]$ and $S^+[q]$ for a query point q.

We describe the indexing scheme for S^-; a similar scheme works for S^+. It is tempting to construct an indexing scheme on S_u^- at each node as in a standard interval tree. However, the segments in S_u^- do not span the slab $\sigma_{l(u)}$, so we cannot regard them as a set of lines and build the indexing scheme described of Section 2.2. Instead we refine the sets S_u^- and proceed as follows.

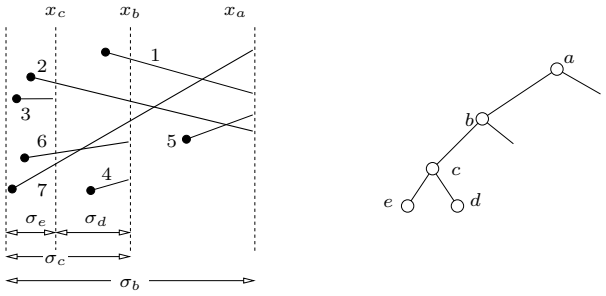

Figure 4. The Φ_v and S_{uv}'s for a set of 7 segments for (a portion of) the interval tree on the right: $S_{ab} = \{1, 2, 5, 7\}$, $S_{ac} = \{1, 2, 7\}$, $S_{ad} = \{1\}$, $S_{ae} = \{2, 7\}$, $S_{bd} = \{4\}$, $S_{be} = \{6\}$; $\Phi_c = \{1, 2, 7\}$, $\Phi_e = \{2, 6, 7\}$.

For a proper descendant v of a node u, we define $S_{uv} \subseteq S_u$ to be

$$S_{uv} = \{s \in S_u^- \mid s\text{'s left endpoint} \in \sigma_v\}. \tag{1}$$

Note that by definition, S_{uv} is empty if v is in the right subtree of u. For a node v, let

$$\Phi_v = \bigcup_u S_{uv}, \tag{2}$$

where the union is taken over all proper ancestors of $p(v)$. See Figure 4 for an example. For a fixed v, the sets S_{uv} are pairwise disjoint; while for a fixed u, the sets S_{uv} induce a binary hierarchical partitioning of S_u. Hence, $\sum_{u,v} |S_{uv}| = O(n \log n)$. A crucial observation is that if v is the left child of its parent, then each segment in Φ_v spans the slab $\sigma_{\mathsf{Sb}(v)}$. Hence, a point $q \in \sigma_{\mathsf{Sb}(v)}$ lies above a segment e of Φ_v if and only if it lies above the line containing e. Let H_v be the set of lines containing the segments in Φ_v. At each node v, we construct the index on H_v described in Section 2.2.

The total size of the overall indexing scheme is $O(n \log n)$, and it can be constructed in time $O(n \log^2 n)$. The following lemma suggests how to report the segments in $S^-[q]$ for a query point q. Let Π_q denote the path from the root to the leaf z of $\mathcal{T}$ such that $q \in \sigma_z$.

LEMMA 4. *Let q be a query point, let z be the leaf of $\mathcal{T}$ such that $q \in \sigma_z$, and let $e \in S^-$ be a segment that lies below q. If the left endpoint of e does not lie in σ_z, i.e., $e \notin \Phi_z$, then there is a node $v \in \Pi_q$ such that $q \in \sigma_{r(v)}$ and $e \in \Phi_{l(v)}$.*

PROOF. Suppose $e \in S_u^-$, then $e \subseteq \sigma_{l(u)}$ and the right endpoint of e lies on the line $x = x_u$, the splitting line of u. Let w be the leaf of $\mathcal{T}$ such that p, the left endpoint of e lies in σ_w. If $w = z$, then $e \in \Phi_z$ and there is nothing to prove. So assume that $w \neq z$, and let v be the lowest common ancestor of w and z. Then $q \in \sigma_{r(v)}$, and $p \in \sigma_{l(v)}$. Since e lies below q, e intersects $\sigma_{r(v)}$ and thus crosses the line $x = x_v$. Therefore u is a proper ancestor of v, implying that $e \in S_{ul(v)}$ and $e \in \Phi_{l(v)}$. This completes the proof of the lemma. □

In view of the above lemma, we can answer a query for a point q as follows: We visit Π_q in a top-down manner. Suppose we are at a node v. If v is leaf, we report the (at most one) segment in Φ_v provided it lies below q. If v is an internal node and $r(v) \in \Pi_q$, then we query the index on $H_{l(v)}$ with q and report all segments of $\Phi_{l(v)}[q]$. Since we query the indexing scheme at $O(\log n)$ nodes, the overall query time is $O(\log^2 n + k)$ time. Again, using the fractional cascading on the indexing schemes built at each node of $\mathcal{T}$, the query time can be reduced to $O(\log n + k)$. Hence, we obtain the following.

LEMMA 5. *Let S be a set of segments in the plane. We can construct in $O(n \log^2 n)$ time an index of $O(n \log n)$ size on S so that all segments of S lying below a query point can be reported in $O(\log n + k)$ time.*

2.5 Hybrid index

We now describe an optimal index for answering segments-below-point queries, by combining the previous two schemes along with additional ideas. We start with the interval-tree based index from the previous subsection. We stop the top-down construction of the interval 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. For each internal node u, we define S_u^- and S_u^+ as above, and let S^-, S^+ be as defined earlier. Since we have "fat" leaves, not all segments will be split — those with both endpoints lying in the same atomic slab will not. For a leaf z, let S_z be the set of segments whose both endpoints lie in σ_z. We build a segment-tree based index (Lemma 3) on S_z. Since $|S_z| = O(\log^2 n)$ and $\sum_z |S_z| \leq n$, a segments-below-point query on S_z can be answered in $O(\log n + k)$ time and the total size and construction time of the index, summed over all leaves, are linear and $O(n \log n)$, respectively.

Next we describe the index we build on S^-; a similar index is built on S^+. Let S_{uv}, Φ_v, and H_v be defined as earlier. By Lemma 4, if a segment $e \in S^-$ lies below a point $q \in \sigma_z$ for a leaf $z \in \mathcal{T}$, then either (i) $e \in \Phi_z$, or (ii) there is an internal node u such that $e \in \Phi_{l(u)}$ and $q \in \sigma_{r(u)}$; in this case a segment of $\Phi_{l(u)}$ lies below q if and only if the line containing it lies below q. To handle (i), we build in $O(|\Phi_z| \log n)$ time the segment-tree based linear-size index on Φ_z for each leaf z. Since the left endpoints of all segments in Φ_z lie in σ_z, $\sum_z |\Phi_z| \leq n$. Therefore the total size of the index over all leaves is $O(n)$, and they can be constructed in $O(n \log n)$ time. It thus suffices to describe how we handle case (ii), which is the more interesting case.

Let Λ be the set of pairs (u, z) such that z is a leaf and u is proper ancestor of z. For a node v, let $\Lambda(v) \subseteq \Lambda$ be the set of pairs (u, z) such that z is a descendant of v and u is a proper ancestor of $p(v)$. Let $n_{uz} = |S_{uz}|$, and let H_{uz} be the set of lines of containing the segments in S_{uz}. For two 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, $\sum_{(u,z)\in\Lambda} n_{uz} \leq n$. By (2), $\Phi_v = \bigcup_{(u,z)\in\Lambda(v)} S_{uz}$. Therefore a set S_{uz} is included in Φ_v at all nodes v that lie on the path from $l(u)$ to z; see Figure 5. To reduce the size of the index to linear, instead of building a separate index for each Φ_v, we build a more global index. Namely, we build in $O(n_{uz} \log n)$ time a linear-size halfplane-range-reporting index (cf. Lemma 2) on H_{uz}, where z is a leaf. By Lemma 2, if we know a line in $L_1(H_{uz})$ lying below q, we can report $H_{uz}[q]$ in time proportional to its size.

Returning to the problem of reporting $\Phi_v[q]$ for a point $q \in \sigma_{\mathsf{Sb}(v)}$, we now need an index 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 index built on H_{uz} to report the remaining lines in $H_{uz}[q]$. One possibility is to build an index on the set $\bigcup_{(u,z)\in\Lambda(v)} L_1(H_{uz})$ at each node to find such a line, but $|L_1(H_{uz})| = |H_{uz}|$ in the worst case, so this will again lead to an index of size $O(n \log n)$. The following observation will help us in reducing the size.

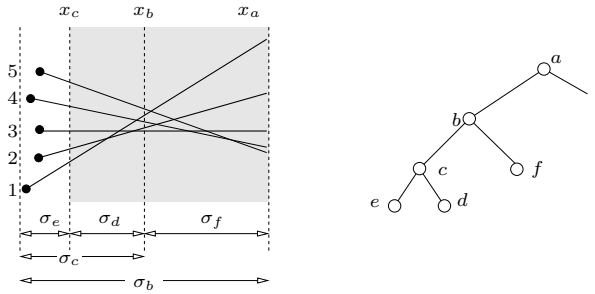

Figure 5. Set $S_{ae} = \{1, 2, 3, 4, 5\}$ and the strip Σ_{ae} (shaded); S_{ae} is included in Φ_e (queried in slab σ_d) and Φ_c (queried in Φ_f); $H^e_{ae} = \{1, 2, 3\}$ and $H^c_{ae} = \{3, 4, 5\}$.

For a node v, Φ_v is queried by a point q only when v is the left child of $p(v)$ and $q \in \sigma_{\mathsf{Sb}(v)}$. Let $\mathbb{V}$ denote the set of nodes $v \in \mathcal{T}$ such that v is left child of $p(v)$. Let Σ_{uz} be the strip formed by the splitting line at u and the right boundary of σ_z; see Figure 5. 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 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 $l(u)$ to z; the (left) sibling of each w_i also lies on this path. The slabs $\sigma_{w_1}, \ldots, \sigma_{w_r}$ induce a partitioning of Σ_{uz}. S_{uz} (or rather H_{uz}) is queried only by the points lying in σ_{w_i}, for $1 \leq i \leq r$. For a query point $q \in \sigma_{w_i}$, we need to report one (representative) line of H_{uz} lying below q. We choose the representative lines of H_{uz} as follows. For a node v on the path from $l(u)$ to z such that $v \in \mathbb{V}$ ($(u, z) \in \Lambda(v)$ and $\mathsf{Sb}(v)$ is one of the w_i's), we define $H^v_{uz} \subseteq H_{uz}$ to be the set of lines that appear on the lower envelope of H_{uz} within $\sigma_{\mathsf{Sb}(v)}$; set $n^v_{uz} = |H^v_{uz}|$. If $n^v_{uz} \geq 2$, then at most one line of H^v_{uz} will appear on the lower envelope of H_{uz} on the right side of $\sigma_{\mathsf{Sb}(v)}$ (for example, only line 3 of H^e_{ae} may appear on the lower envelope of H_{ae} on the right side of σ_d). Since $r = O(\log n)$, we have

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

For a pair u, z, the sets H^v_{uz} can be computed in $O(n_{uz} \log n)$ time by constructing the lower envelope of H_{uz}. H^v_{uz} is the set of representative lines that are stored at v.

For a node $v \in \mathbb{V}$, let $\Gamma_v = \bigcup_{(u,z)\in\Lambda(v)} H^v_{uz}$ be all the representative lines stored at v. We build in $O(|\Gamma_v| \log n)$ time the halfplane-range-reporting index of linear size on Γ_v. For each line ℓ in Γ_v, we store a pointer to its copy in H_{uz}. Finally, we also build a fractional-cascading structure on these indexing schemes. Summing over all nodes $v \in \mathcal{T}$,

$$\begin{aligned}
\sum_{v\in\mathbb{V}} |\Gamma_v| &= \sum_{v\in\mathbb{V}} \sum_{(u,z)\in\Lambda(v)} n^v_{uz} \\
&= \sum_{(u,z)\in\Lambda} \sum_{v\in\mathbb{V},(u,z)\in\Lambda(v)} n^v_{uz} \\
&= \sum_{(u,z)\in\Lambda} (n_{uz} + O(\log n)) \\
&= O(n) + O\left(\frac{n}{\log^2 n} \log n \log n\right) \\
&= O(n).
\end{aligned}$$

This completes the description of the index we build. Putting all the pieces together, the total size of the index 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 $S^-[q]$ using Lemma 4: We first query the index on Φ_z and report in $O(\log n + |\Phi_z[q]|)$ time the set $\Phi_z[q]$. Next, for each node $v \in \Pi_q$, if v is the right child of $p(v)$, we report the set $H_{\mathsf{Sb}(v)}[q]$. We accomplish this in two stages: We first query the index on $\Gamma_{\mathsf{Sb}(v)}$ and report in $O(|\Gamma_{\mathsf{Sb}(v)}[q]|)$ time the set $\Gamma_{\mathsf{Sb}(v)}[q]$, i.e., the set of representative lines of $H_{\mathsf{Sb}(v)}$ that lie below q. Let $\ell \in \Gamma_{\mathsf{Sb}(v)}[q]$ be the line that belongs to H_{uz}. We then report the set $H_{uz}[q]$ in time proportional to its size by querying the index built on H_{uz}. The total time spent at v is $O(|H_{\mathsf{Sb}(v)}[q]|)$. We spend an additional $O(\log n)$ time at the root of $\mathcal{T}$ to search in the fractional cascading structure built on top of Γ_v's. Finally, we report $S^+[q]$ in a similar manner. Putting everything together, the total query time is $O(\log n + |S[q]|)$.

THEOREM 6. *Let S be a set of n segments in $\mathbb{R}^2$. We can build in $O(n \log n)$ time a linear-size index on S 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 7. *Given a set P of n uncertain points in $\mathbb{R}^1$, each associated with a histogram having s pieces, and a threshold parameter $0 < \tau \leq 1$, we can build in $O(n \log n)$ time a linear-size index on P so that a range query on S with probability threshold τ can be answered in $O(\log n + k)$ time, where k is the output size.*

2.6 Extensions

Finally, we briefly describe two useful extensions of our indexing scheme.

Externalization. It is not difficult to externalize our index under the standard two-level I/O model [6], so that the query can be answered I/O-efficiently. Agarwal et al. [2]

developed an external index for half-plane range reporting. Although not explicitly mentioned in [2], their structure is amenable to fractional cascading, so that we can extend Lemma 2 to the I/O model, namely, given any line in $L_1(S)$ below a query point q, all the k lines in S lying below q can be reported in $O(k/B)$ I/Os using an index occupying $O(n/B)$ disk blocks.

Next we externalize the segment-tree based index. The fanout of the segment tree is set to $r = (n/B)^{1/4}/\sqrt{\log(n/B)}$, and the leaf size of the tree is set to $\Theta(B)$. For each multi-slab, we build the external half-plane range-reporting index for its segments, as mentioned above. The height of the tree is still $O(1)$, and it takes $O(r^2+k/B)$ I/Os to answer a query. So we obtain a linear-size index for S so that all segments below a query point can be reported using $O(\sqrt{n/B}+k/B)$ I/Os. Equipped with these schemes, we can go through the construction of the hybrid index, while only changing the leaf size from $\Theta(\log^2 n)$ to $\Theta(B\log^2 n)$. Without repeating the tedious details, we conclude with the following.

THEOREM 8. *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 $0 < \tau \le 1$, we can build a linear-size external-memory index on S such that a range query with probability threshold τ can be answered in $O(\log n + k/B)$ I/Os.*

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 index. If only insertions are to be supported, we can apply the *logarithmic method* [9] to Theorem 6. Then standard analysis gives us a semi-dynamic structure that answers a query in $O(\log^2 n + k)$ time and supports insertion/deletion of a point in amortized $O(\log^2 n)$ time. Unfortunately, it is hard to support deletions, since our index crucially relies on the halfplane searching structure of [12], which is inherently static. The best known dynamic index for halfplane range reporting uses $O(n^{1+\varepsilon})$ space, supports insertions and deletions in $O(n^{\varepsilon})$ time amortized, and answers queries in $O(\log n + k)$ time [4], where ε is any small constant. Since super-linear space is unavoidable, the hybrid index is not needed any more. Instead, we can simply plug this dynamic halfplane structure into the segment-tree based index with fanout 2 (see the remark following Lemma 3). Omitting the details, we conclude the following:

THEOREM 9. *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 $0 < \tau \le 1$, we can build a fully dynamic index on P of size $O(n^{1+\varepsilon})$, for any constant $\varepsilon > 0$, that answers a range query with probability threshold τ in $O(\log^2 n + k)$ time, and supports insertions and deletions of uncertain points in $O(n^{\varepsilon})$ 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 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 9 becomes $O(sn^{\varepsilon})$ 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 indexing a set of x-monotone polygonal chains so that all the chains below a query point can be reported efficiently. In this section, we follow a completely different approach to solve this problem. It results in an index 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[1]. Although the space bound is not as good as the structure in Theorem 6, the new index we present below is simpler and easily extends to the case where the polygonal chains are replaced by more general curves, such as piecewise-quadratic functions or other algebraic curves. This will allow us to handle pdf's that are more general than histograms. For instance, if the pdf is a piecewise linear function, then the same reduction of Section 2.1 will produce threshold functions that are piecewise quadratic.

The framework of our index is similar to that of the *3D halfspace searching* structure of Chan [10]. Let C be the set of n polygonal chains representing the n threshold functions; each of them consists of $2s$ 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 R_i, we compute its lower envelope E_i. According to [18], 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 series of trapezoids (see Figure 6). For each trapezoid t, 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. Invoking the random-sampling framework of Clarkson and Shor [15], we can prove that the expected size of C_t is $O(2^i)$. Therefore, the expected total size of our structure is

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

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 index in a total of $O(n\log^2 n)$ time in expectation.

The final touch-up to our structure is a fractional cascade on the E_i's, such 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 fractional cascade is only $O(n\alpha(n))$, and it can be built in the same amount of time.

Now we describe how a query q is answered using the index constructed above. First we find all the $\log n$ trapezoids in $O(\log n)$ time that intersect the vertical line ℓ passing q, one from each E_i. Denote by t_i the trapezoid from E_i. Below we show that for a given r, how to use our structure to find in $O(r)$ expected time the r lowest chains 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 $r/2 \le k < r$, and we just report the k chains that are actually below q. The total time spent will be $O(\log n + 1 + 2 + 4 + \cdots + r) = O(\log n + k)$.

[1] If $n \le 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\Big\}$ 65536 twos, then $\alpha(n) \le 4$.

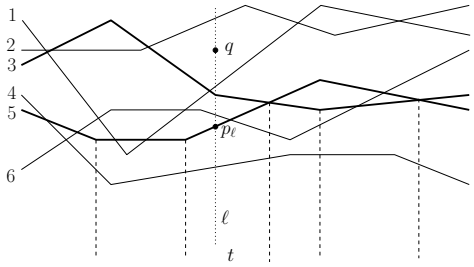

Figure 6. The thick chains are in the random sample R_i. The dashed lines divide the lower envelope of R_i into trapezoids. For the trapezoid t, its conflict list C_t consists of chains 4 and 6.

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. We will examine C_{t_ρ} where $\rho = \lfloor \log(r/\delta) \rfloor$. We first check if $|C_{t_\rho}| > r/\delta^2$. 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 (Figure 6). While scanning C_t we check if 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) $|C_{t_\rho}| > r/\delta^2$; and (b) there are less 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 the probability that $|C_{t_\rho}|$ exceeds r/δ^2 is $O(\delta)$. 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 $O(r/2^\rho) = O(\delta)$. Thus by union bound the algorithm 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},..., stopping as soon as some invocation succeeds. Let X_i be the indicator random variable whose value is 1 if the i-th invocation succeeds, and 0 otherwise. Then the total expected running time is

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

THEOREM 10. *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 $0 < \tau \leq 1$, we can build in $O(n \log^2 n)$ time an index of size $O(n\alpha(n) \log n)$, where $\alpha(n)$ is the inverse Ackermann function, such that a range query with probability threshold τ can be answered in expected $O(\log n + k)$ time.*

As commented earlier, this index easily extends to more general pdf's. All the algorithms remain the same, except that the complexity of E_i may vary. 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 $O(n\alpha(n))$ [18]. With other families of pdf's, the threshold functions will have different forms. Interestingly, the complexity of their lower envelope only depends on how many times two pieces of two different threshold functions could intersect. If two pieces intersect at no more than c points ($c = 1$ in the case of histogram pdfs), 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., in the case of pdfs being Gaussian distribution), then the complexity of the envelope is $\lambda_c(n)$. Thus Theorem 10 still holds, with the space bound changing to $O(\lambda_{c+2}(n) \log n)$ and $O(\lambda_c(n) \log n)$, 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 [5] for a complete treatment of Davenport-Schinzel sequences and their applications and the recent paper [20] for slightly improved bounds.

For most common pdf's, c is a small constant. In general, if the pdf is a piecewise polynomial function with degree d, its threshold function is a piecewise polynomial with degree $d+1$, and the space bound becomes $O(\lambda_{d+3}(n) \log n)$ correspondingly. If the pdf is a Gaussian distribution, then $c = 2$ and the size of our index is $O(n \log n)$.

THEOREM 11. *Let P be a set of n uncertain points in $\mathbb{R}$, their pdf's, each having s pieces and any two pieces intersecting in at most c points, and let $0 < \tau \leq 1$ be a parameter. We can build an index on P 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 probability threshold τ can be answered in expected $O(\log n + k)$ time.*

Remarks. We first remark that this index can be extended to external memory, following a similar framework described in [2]. We omit the technical details from this abstract. Secondly, the index can be easily made to support approximate counting queries with relative error at most ε following a similar procedure as in [8]. The query time is $O(\log n)$ and the size of the index is $O(1/\varepsilon \cdot \lambda_{c+2}(n) \log n)$. We omit the details.

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 an index with $O(n \cdot \text{polylog}(n))$ size and $O(\text{polylog}(n) + k)$ query time.

In the 3D halfspace searching problem, we want to index a set of points in $\mathbb{R}^3$ such that all points below a given a query plane can be reported efficiently. By duality, this is equivalent to indexing a set of planes in $\mathbb{R}^3$, such that for a query point p, all planes below p are reported. It had been a long open problem whether 3D halfspace searching can be solved in linear space and $O(\log n + k)$ query time, but recently Afshani and Chan presented such a solution [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.

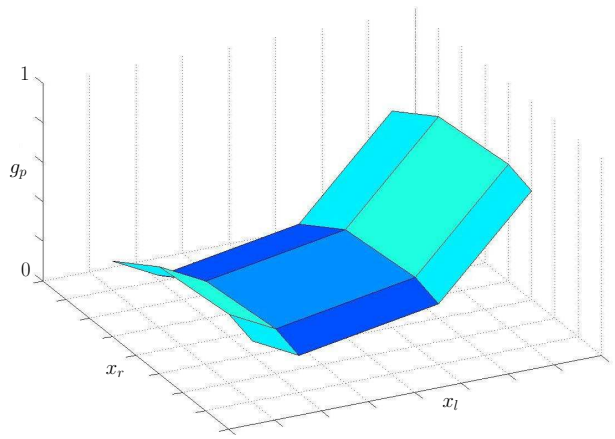

Figure 7. $\Pr[p \in [x_l, x_r]]$ is a bivariate piecewise linear function in x_l and x_r. It consists of s^2 pieces and each piece covers a rectangular region in the $x_l x_r$-plane.

The point p lies in an interval $[x_l, x_r]$ 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 pdfs 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 an index 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 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 index by Afshani and Chan [1] on Φ_C for answering halfspace range-reporting queries. Since this index 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 index 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 12. *Given a set P of n uncertain points in $\mathbb{R}$, each associated with a histogram having at most s pieces, we can build in expected $O(n \log^3 n)$ time an index of size $O(n \log^2 n)$ on P, 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.*

Externalization. To extend our index to external memory, we employ the external structure of Agarwal et al. [2] for 3D halfspace searching structure. This structure has an expected size of $O(\frac{n}{B} \log \frac{n}{B})$ blocks and answers a query using $O(\log_B n + k/B)$ expected I/Os. Using this structure for each nonempty canonical vertical strip, we obtain the following result.

THEOREM 13. *Given a set P of n uncertain points in $\mathbb{R}$, each associated with a histogram having s pieces, we can build an external memory index on disk occupying expected $O(\frac{n}{B} \log^3 \frac{n}{B})$ blocks, so that for a query interval I and a probability τ, it can report using expected $O(\log^2 \frac{n}{B} \log_B \frac{n}{B} + k/B)$ I/Os all k points of P that lie in I with probability at least τ.*

Dynamization. Similar to Section 2.3, we can make this indexing scheme semi-dynamic by applying the logarithmic method. Insertions can be supported in expected $O(\log^3 n)$ amortized time. The query bound increases to $O(\log^4 n + k)$. We omit the rather uninteresting details.

To make our index fully dynamic, we replace the static 3D halfspace searching structure of [1] with the dynamic structure of [4]. The same as the 2D case, this structure uses $O(n^{1+\varepsilon})$ space, supports insertions and deletions in $O(n^{\varepsilon})$ time amortized, and answers queries in $O(\log n + k)$ time [4], where ε is any small constant. By using this structure for each nonempty canonical vertical strip, we obtain the following:

THEOREM 14. *Given a set of n uncertain point, each associated with a histogram having s pieces, we can build a fully dynamic index of size $O(n^{1+\varepsilon})$ such that a range query with any probability threshold can be answered in $O(\log^3 n + k)$ time. This index supports insertions and deletions of uncertain points in $O(n^{\varepsilon})$ 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 14 becomes $O(s^2 n^{\varepsilon})$.

5. CONCLUSION

In this paper we have studied the problem of indexing uncertain data to support range queries efficiently. Our indexing schemes 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 indexes, 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 indexes.

6. ADDITIONAL AUTHORS

References

[1] P. Afshani and T. M. Chan, Optimal halfspace range reporting in three dimensions, *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2009, pp. 180–186.

[2] P. K. Agarwal, L. Arge, J. Erickson, P. Franciosa, and J. Vitter, Efficient searching with linear constraints, *Journal of Computer and System Sciences*, 61 (2000), 194–216.

[3] P. K. Agarwal and J. Erickson, Geometric range searching and its relatives, in: *Advances in Discrete and Computational Geometry* (B. Chazelle, J. E. Goodman, and R. Pollack, eds.), American Mathematical Society, Providence, RI, 1999, pp. 1–56.

[4] P. K. Agarwal and J. Matoušek, Dynamic half-space range reporting and its applications, *Algorithmica*, 13 (1995), 325–345.

[5] P. K. Agarwal and M. Sharir, Davenport-Schinzel sequences and their geometric applications, in: *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, eds.), Elsevier Science Publishers, Amsterdam, 2000, pp. 1–47.

[6] A. Aggarwal and J. S. Vitter, The input/output complexity of sorting and related problems, *Communications of the ACM*, 31 (1988), 1116–1127.

[7] P. Agrawal, O. Benjelloun, A. Das Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom, Trio: A system for data, uncertainty, and lineage, *Proc. International Conference on Very Large Databases*, 2006, pp. 1151–1154.

[8] B. Aronov, S. Har-Peled, and M. Sharir, On approximate halfspace range counting and relative epsilon-approximations, *Proc. 23rd Annual Symposium on Computational Geometry*, 2007, pp. 327–336.

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

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

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

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

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

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

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

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

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

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

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

[20] G. Nivasch, Improved bounds and new techniques for Davenport-Schinzel sequences and their generalizations, *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2009, pp. 1–10.

[21] H. Samet, *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann, 2006.

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

[23] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar, Indexing multi-dimensional uncertain data with arbitrary probability density functions, *Proc. International Conference on Very Large Databases*, 2005, pp. 922–933.

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

[25] M. L. Yiu, N. Mamoulis, X. Dai, Y. Tao, and M. Vaitis, Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data, *IEEE Transactions on Knowledge and Data Engineering*, 21 (2009), 108–122.