Title	Indexing multi-dimensional uncertain data with arbitrary probability density functions
Author(s)	Taot, Y; Cheng, R; Xiao, X; Ngai, WK; Kao, B; Prabhakar, S
Citation	The 31st International Conference on Very Large Data Bases (VLDB 2005), Trondheim, Norway, 30 August-2 September 2005. In Proceedings of 31st VLDB, 2005, v. 3, p. 922-933
Issued Date	2005
URL	http://hdl.handle.net/10722/93064
Rights	Creative Commons: Attribution 3.0 Hong Kong License

Indexing Multi-Dimensional Uncertain Data with Arbitrary Probability Density Functions

Yufei Tao[†], Reynold Cheng[‡], Xiaokui Xiao[†], Wang Kay Ngai[§], Ben Kao[§], Sunil Prabhakar[♯]

[†]Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
{*taoyf, xkxiao*}*@cs.cityu.edu.hk*

[‡]Department of Computer Science
Hong Kong Polytechnic University
Hung Hom, Hong Kong
csckcheng@comp.polyu.edu.hk

[§]Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong
{*wkngai, kao*}*@csis.hku.hk*

[♯]Department of Computer Science
Purdue University
West Lafayette, USA
sunil@cs.purdue.edu

Abstract

In an "uncertain database", an object o is associated with a multi-dimensional probability density function (pdf), which describes the likelihood that o appears at each position in the data space. A fundamental operation is the "probabilistic range search" which, given a value p_q and a rectangular area r_q, retrieves the objects that appear in r_q with probabilities at least p_q. In this paper, we propose the U-tree, an access method designed to optimize both the I/O and CPU time of range retrieval on multi-dimensional imprecise data. The new structure is fully dynamic (i.e., objects can be incrementally inserted/deleted in any order), and does not place any constraints on the data pdfs. We verify the query and update efficiency of U-trees with extensive experiments.

1 Introduction

Uncertain databases are gaining considerable attention recently [13]. In such a system, tuples may not accurately capture the properties of real-world entities, which is an inherent property of numerous applications that manage "dynamic attributes" [14] with continuously changing values. To enable *location-based services* [15], for instance, a moving client informs a server about its coordinates, if its distance from the previously reported location exceeds a certain threshold. Hence, the database does not have clients' precise positions — an object can be anywhere in a circular *uncertainty region* (the grey area in Figure 1) that centers at its last update, and has a radius equal to the distance threshold.

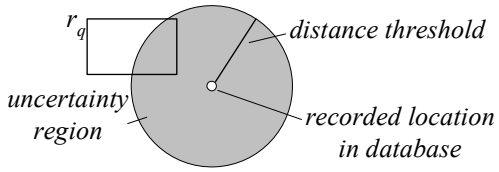


Figure 1: An uncertain object example

As another example, consider a *meteorology system* that monitors the temperatures, humidity, UV indexes (and etc.) in a large number of regions. The corresponding readings are taken by sensors in local areas, and transmitted to a central database periodically (e.g., every 30 minutes). The database content may not exactly reflect the current atmospheric status, e.g., the actual temperature in a region may have changed since it was last measured.

The uncertainty in the above examples is caused by delayed data updates while, in general, sources of imprecision include data randomness, limitation of measuring equipments, and so on. Information retrieval directly based on uncertain data is meaningless, since the result does not have any quality guarantees. Consider, for example, the query "find the clients currently in the downtown area". Returning the objects whose last updates satisfy the query is inadequate, because many objects may have entered or left the search region since they contacted the server last time.

To avoid this problem, the "precise" values need to be estimated using a probability density function (pdf). For example, if the location of a moving client o is considered uniformly distributed in its uncertainty region ur, the object pdf can be represented as $pdf(x) = 1/AREA(ur)$ if the parameter x (any point in the 2D data space) belongs to ur, or 0 otherwise. Thus, the *appearance probability* that

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

o lies in a given region r_q (e.g., the rectangle in Figure 1) equals:

$$\int_{r_q \cap ur} pdf(x)dx = \frac{AREA(r_q \cap ur)}{AREA(ur)} \quad (1)$$

where the integral area $r_q \cap ur$ is the intersection between r_q and ur. For simplicity, we used the *uniform distribution* in the above example, while in practice an appropriate pdf depends on the characteristics of the underlying application. For instance, an actual temperature may follow a *Gaussian* distribution with an appropriate variance and a mean calculated based on the last measured value (e.g., in the daytime, when the temperature is expected to rise, the mean may be set to some number larger than the measured one). Other common stochastic models include *Zipf*, *Poisson* (for describing the happening frequency of some event), etc.

In general, an "uncertain object" is a multi-dimensional point whose location can appear at any position x in the data space, subject to a probability density function $pdf(x)$. Given a value p_q and a rectangular query region r_q, a *probabilistic range search* (prob-range) returns the objects that appear in r_q with probabilities *at least* p_q. In location-based systems, such a query q would "retrieve the objects that are currently in the downtown area (r_q) with a probability no less than 80%". A similar inquiry in a meteorology system may "identify the regions whose temperatures are in range [75F, 80F], humidity in [40%, 60%], and UV indexes [4.5, 6] with at least 70% likelihood", where the search area r_q is a 3D box with projections on the temperature-, humidity-, UV-index dimensions described by the corresponding ranges, respectively.

Although conventional range search (on a "precise" dataset) has been very well studied [1, 3], its solutions are not applicable to uncertain data, since they do not consider the probabilistic requirements [6]. As explained later, the key of optimizing a prob-range query is to avoid, as much as possible, computing the appearance probability that an object satisfies a query. Such computation is expensive (especially when the dimensionality is high), since it requires costly numerical evaluation of a complex integral.

In this paper, we present the U-tree, a multi-dimensional access method on uncertain data with arbitrary pdfs. This structure minimizes the amount of appearance probability computation in prob-range search. Intuitively, it achieves this by pre-computing some "auxiliary information" for each object, which can be used to disqualify the object (in executing a query) or to validate it as a result without having to obtain its appearance probability. Such information is maintained at all levels of the tree to avoid accessing the subtrees that do not contain any results. Furthermore, U-trees are fully dynamic, i.e., objects can be inserted/deleted in any order.

The rest of the paper is organized as follows. Section 2 reviews the previous work that is directly related to ours. Section 3 formally defines the problem, and Section 4 explains techniques for efficiently pruning objects in prob-range search. Section 5 presents U-trees and clarifies the concrete update and query algorithms, while Section 6 evaluates the proposed methods with extensive experiments. Section 7 concludes the paper with directions for future work.

2 Related Work

In the next section, we first review the existing results on query processing in uncertain databases. Then, Section 2.2 describes the R*-tree, which is an effective multi-dimensional access method for range queries on precise data, and is fundamental to the subsequent discussion.

2.1 Query Processing on Imprecise Data

Early research [10, 14, 15] primarily focuses on various data models for accurately capturing the locations of moving objects. In this context, query algorithms aim at minimizing the amount of data transmission (for updating the central server) to ensure the precision of database values. Cheng et al. [4] are the first to formulate uncertain retrieval in general domains. They present an interesting taxonomy of novel query types, together with the corresponding processing strategies. An I/O efficient algorithm for nearest neighbor search is proposed in [5]. None of the above works considers prob-range retrieval.

Cheng et al. [6] develop several solutions for prob-range queries which, however, target 1D space only. They argue that range search in uncertain databases is inherently more difficult than that on traditional precise objects, and support their claims by providing two theoretical approaches that achieve (almost) asymptotically optimal performance. Nevertheless, the practicability of these methods is limited since (i) they cannot support objects with arbitrary pdfs (e.g., one method targets only uniform pdfs), and (ii) they may incur large actual execution overhead due to the hidden constants in their complexity guarantees.

Dalvi and Suciu [8] discuss "probabilistic databases", where each record is the same as a tuple in a conventional database, except that it is associated with an "existential" probability. For example, a 60% existential probability means that a tuple may not exist in the database with a 40% chance; if it does, however, its values are precise. Hence, probabilistic databases are different from uncertain databases (the topic of this paper), where each object definitely exists but its concrete values follow a probabilistic distribution.

2.2 R*-trees

The R*-tree [1] can be regarded as an extension of the B-tree for multi-dimensional rectangular objects. Figure 2 shows a two-dimensional example where 10 rectangles (a, b, ..., j) are clustered according to their spatial proximity into 4 leaf nodes N_1, ..., N_4, which are then recursively grouped into nodes N_5, N_6 that become the children of the root. Each intermediate entry is represented as a *minimum bounding rectangle* (MBR), which tightly bounds all the data in its subtree (e.g., N_1 is the MBR enclosing a, b, c).

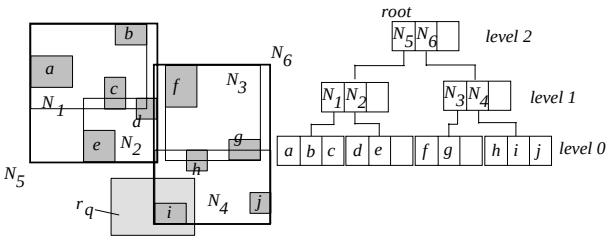


Figure 2: An R*-tree

To find the objects intersecting the search region r_q in Figure 2, for example, the algorithm visits the nodes (root of the R-tree, N_6, N_4) whose MBRs intersect r_q, and returns the only qualifying object i.

The R*-tree construction algorithm aims at minimizing the following penalty metrics: (i) the area, (ii) the margin (i.e., perimeter) of each MBR, (iii) the overlap between two MBRs in the same node, and (iv) the distance between the centroid of an MBR and that of the node containing it. As discussed in [9], minimization of these metrics decreases the probability that an MBR intersects a query region.

3 Problem Definition

Formally, an "uncertain object" o is associated with (i) a probability density function $o.pdf(x)$, where x is an arbitrary d-dimensional point, and (ii) a d-dimensional uncertainty region $o.ur$. Given a prob-range query with a (hyper-) rectangle r_q and a threshold $p_q \in [0, 1]$, the appearance probability $P_{app}(o, q)$ of an object o is calculated as:

$$P_{app}(o, q) = \int_{o.ur \cap r_q} o.pdf(x) \quad dx \tag{2}$$

where $o.ur \cap r_q$ denotes the intersection of $o.ur$ and r_q. Object o is a result if $P_{app}(o, q) \geq p_q$.

Our objective is to minimize the cost (including both I/O and CPU time) of prob-range search, without making any assumption about the "types" (e.g., uniform, Gaussian, Zipf, etc.) of objects' pdfs. Clearly, the problem would be much easier if all the pdfs were known to be of the same "type". For example, if only Gaussian functions were present, specialized methods could be developed based on their means and variances [6]. These methods, however, are not useful for other types of pdfs, which in turn require "dedicated" solutions based on their own characteristics. Instead, we aim at developing a "unified" solution that can support a database where objects can have arbitrary pdfs.

One difficulty in handling multi-dimensional data is that the integral in Equation 2 cannot be solved accurately even for a "regular" pdf such as Gaussian. To see this, assume that in Figure 1, the object's actual location is described using a Gaussian pdf whose mean falls at the center of the circle (i.e., the uncertainty region $o.ur$). Given an arbitrary query area r_q, the intersection between r_q and $o.ur$ has a shape that is *not* symmetric with respect to the mean. In this case, Equation 2 cannot be derived into a formula without any integrals, and hence, must be evaluated *numerically* through, for example, the following "monte-carlo" approach[1] [2].

First, a number n_1 of points x_1, x_2, ..., x_{n_1} are randomly generated in the uncertainty region $o.ur$ of an object o. Without loss of generality, assume that n_2 of these points fall into the search region r_q, and they are x_1, x_2, ..., x_{n_2}, respectively. For each point x_i ($1 \leq i \leq n_1$), we pass it into the object's pdf, and calculates the resulting value $pdf(x_i)$. Then, P_{app} (in Equation 2) can be approximated as:

$$P_{app}(o, q) \approx \sum_{i=1}^{n_2} o.pdf(x_i) / \sum_{i=1}^{n_1} o.pdf(x_i) \tag{3}$$

As a special case, when the entire uncertainty region $o.ur$ falls inside the query area r_q, the above equation returns the correct value 1 of $P_{app}(o, q)$ (since $n_2 = n_1$). In general, however, monte-carlo is accurate only if n_1 *is sufficiently large* (at the order of 10^6, as tested in our experiments). Even worse, the appropriate n_1 increases with the dimensionality. *Therefore, computing P_{app} incurs expensive costs, especially when the dimensionality d is high.* In the next section, we present techniques that can eliminate a majority of the non-qualifying data without calculating their appearance probabilities.

4 Filtering Multi-Dimensional Uncertain Data

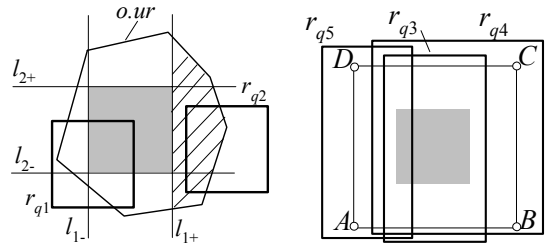
Section 4.1 first introduces "probabilistically constrained regions" (PCR) and explain the heuristics of applying PCRs to assist prob-range search, while Section 4.2 discusses "practical" versions of these heuristics. Section 4.3 presents "conservative functional boxes" (CFB) as a space-efficient method to capture PCRs. Section 4.4 provides an algorithm for computing CFBs based on linear programming.

4.1 Probabilistically Constrained Regions

The PCR $o.pcr(p)$ of an object o takes a parameter p whose value is in $[0, 0.5]$. Figure 3a illustrates a 2D example, where the polygon represents the uncertainty region $o.ur$ of o (our technique can be applied to uncertainty regions of any shapes). The $o.pcr(p)$ (the grey area) is decided by 4 lines l_{1+}, l_{1-}, l_{2+}, l_{2-}. Line l_{1+} divides $o.ur$ into two parts (on the left and right of l_{1+} respectively) and the appearance probability of o in the right part (i.e., the shadowed area) equals p. Similarly, l_{1-} is such that the appearance likelihood of o on the left of l_{1-} equals p. Clearly, the probability that o lies between l_{1-} and l_{1+} is $1 - 2p$. Lines l_{2+} and l_{2-} are obtained in the same way, except that they horizontally partition $o.ur$.

It is possible to use PCRs to *prune* a non-qualifying object, or to *validate* that an object indeed satisfies a query, without computing the accurate appearance probability. To illustrate pruning, assume that the grey box in Figure 3a is the $o.pcr(0.2)$ of o, and boxes r_{q1}, r_{q2} are the search areas

[1] We choose monte-carlo because it is a popular technique for solving complex equations in the database literature [2]. In the future work, we will investigate alternative numerical approaches for evaluating Equation 2, as well as their impacts on query performance.



(a) For pruning (b) For validating

Figure 3: A 2D probabilistically constrained region

of two prob-range queries q_1, q_2 with probability thresholds $p_{q1} = 0.8$, $p_{q2} = 0.2$, respectively. Object o cannot qualify q_1 because r_{q1} *does not fully contain* $o.pcr(0.2)$. To understand this, notice that r_{q1} is disjoint with the shadowed region. Hence, the appearance probability of o in r_{q1} must be smaller than $1-0.2 = 0.8$, where 0.2 is the probability of o falling on the right of l_{1+}. Rectangle $o.pcr(0.2)$ also indicates that o does not satisfy q_2, but for a different reason: r_{q2} *does not intersect* $o.pcr(0.2)$. In fact, since r_{q2} lies entirely on the right of l_{1+}, the appearance probability of o in r_{q2} is definitely smaller than 0.2 (the probability that o lies in the shadowed region, as mentioned earlier).

Figure 3b explains how a PCR is utilized to validate an object. Again, assume the grey box to be $o.pcr(0.2)$, and rectangle $ABCD$ is the MBR (denoted as $o.MBR$) of the uncertainty region (the polygon). Lines l_{1-} and l_{1+}, which pass the left and right boundaries of $o.pcr(0.2)$, are not shown (to avoid an excessively complex figure) but should be implicitly understood. Consider queries q_3, q_4, q_5 with search areas r_{q3}, r_{q4}, r_{q5}, and probability thresholds 0.6, 0.8, and 0.2 respectively. Object o must satisfy q_3 because r_{q3} *fully covers the part of* $o.pcr(0.2)$ *between* l_{1-} *and* l_{1+}, where the appearance probability of o equals $1-0.2-0.2 = 0.6$. It can also be asserted that o qualifies q_4 (and q_5) since r_{q4} (and r_{q5}) *completely contains the portion of* $o.pcr(0.2)$ *on the right (and left) of* l_{1-}, where o appears with a probability $1 - 0.2 = 0.8$ (and 0.2, respectively). It is important to note that different pruning/validating criteria were used for the 5 queries q_1, q_2, ..., q_5 in Figure 3.

Formally, in a general d-dimensional space, the PCR $o.pcr(p)$ ($p \leq 0.5$) of an object o is a hyper-rectangle decided by a $2d$-dimensional vector: $\{o.pcr_{1-}(p)$, $o.pcr_{1+}(p)$, ..., $o.pcr_{d-}(p)$, $o.pcr_{d+}(p)\}$. In particular, $[o.pcr_{i-}(p), o.pcr_{i+}(p)]$ is the projection of $o.pcr(p)$ on the i-th dimension. In the sequel, without ambiguity we also use $o.pcr_{i-}(p)$ to refer to a plane that is perpendicular to the i-th dimension, and intersects this axis at coordinate $o.pcr_{i-}(p)$. Value $o.pcr_{i+}(p)$ also defines a plane in a similar manner. Then, the probability that o appears on the left (right) of plane $o.pcr_{i-}(p)$ ($o.pcr_{i+}(p)$) equals p, where "left" refers to the negative direction of the i-th axis, and "right" to the positive direction. Notice that $o.pcr(p)$ of an object o continuously shrinks as p increases, and when $p = 0.5$, $o.pcr(p)$ degenerates into a point. The heuristics illustrated in Figure 3 are formalized as follows.

Observation 1. *For a prob-range query q with search region r_q and probability p_q:*

1. *For $p_q > 0.5$, an object o can be eliminated if r_q does not fully contain $o.pcr(1-p_q)$;*
2. *For $p_q \leq 0.5$, the pruning condition is that r_q does not intersect $o.pcr(p_q)$;*
3. *For any p_q, an object is guaranteed to satisfy q if r_q fully covers the part of $o.MBR$ between planes $o.pcr_{i-}(\frac{1-p_q}{2})$ and $o.pcr_{i+}(\frac{1-p_q}{2})$ for some $i \in [1,d]$;*
4. *For $p_q > 0.5$, the validating criterion is that r_q completely contains the part of $o.MBR$ on the right (or left) of plane $o.pcr_{i-}(1-p_q)$ (or $o.pcr_{i+}(1-p_q)$) for some $i \in [1,d]$.*
5. *For $p_q \leq 0.5$, the validating criterion is that r_q completely contains the part of $o.MBR$ on the left (or right) of $o.pcr_{i-}(p_q)$ (or $o.pcr_{i+}(p_q)$) for some $i \in [1,d]$.*

In many cases, an object can be asserted to violate or satisfy a query using the above rules directly, thus avoiding the expensive appearance probability computation (which is necessary *only if* these rules can neither prune nor validate the object).

Observation 1 requires a fast method to answer questions in the form: "does r_q cover the part of $o.MBR$ between two planes l_- and l_+ perpendicular to an axis (called *the interesting dimension*)?". For this purpose, we first examine if r_q completely encloses $o.MBR$ on all dimensions except the interesting axis. If not, then the answer to the original question is negative. Otherwise, we continue to check whether the projection of r_q on the interesting dimension includes the corresponding projections of l_- and l_+ (i.e., two points). The answer to the target question is "yes" only if the second check returns a positive result. Hence, the total examination time is $O(d)$ (the time of checking the intersection of d pairs of 1D intervals).

Equipped with the above method, Observation 1 can prune/validate an object with a small cost. Specifically, depending on the value of p_q, only 3 rules are applicable simultaneously. For example, for $p_q > 0.5$, only Rules 1, 4, 3 are useful, and we apply them in this order (e.g., if Rule 1 already achieves pruning, then no validation is necessary). Similarly, for $p_q \leq 0.5$, we apply Rule 2 first, followed by Rules 5, 3 respectively.

Interestingly, although evaluating the appearance probability of an object is costly, $o.pcr(p)$ can actually be obtained with small overhead, since it can be computed by considering each individual dimension in turn. We illustrate this using the 2D example in Figure 3a but the idea extends to arbitrary dimensionality in a straightforward manner. To decide, for example, line l_{1-} (l_{1+}), we resort to the *cumulative density function* $o.cdf(x_1)$ of $o.pdf(x)$ on the horizontal dimension. Specifically, $o.cdf(x_1)$ is the probability that o appears on the left of a vertical line intersecting the axis at x_1. Thus, l_{1-} can be decided by solving x_1 from equation $o.cdf(x_1) = p$, and similarly, l_{l+} from $o.cdf(x_1) = 1 - p$. For "regular" pdfs (e.g., uniform), $o.cdf(x_1)$ can be derived into a simple formula (by integrating $o.pdf(x)$ along only one dimension), after which both equations mentioned earlier can be solved efficiently.

4.2 Heuristics with A Finite Number of PCRs

The effectiveness of Observation 1 would be maximized if we could pre-compute the PCRs for all $p \in [0, 0.5]$. Since this is impossible, for each object o, we obtain its $o.pcr(p)$ only at some *pre-determined values of* p, which are common to all objects and constitute the *U-catalog* (a system parameter). Denote the values in the U-catalog as p_1, p_2, ..., p_m sorted in ascending order, where m is the size of the catalog. A problem, however, arises. Given an arbitrary p_q, the corresponding PCR needed in Observation 1 for pruning/validating may not exist. For instance, in Figure 3a, as mentioned earlier disqualifying the object o for query q_1 requires its $o.pcr(0.2)$. Thus, the pruning cannot be performed if 0.2 is not a value in the U-catalog.

We solve this problem by applying Observation 1 in a "conservative" way. Assuming a U-catalog with $m = 3$ values $\{p_1 = 0.1, p_2 = 0.25, p_3 = 0.4\}$, Figure 4 shows an example where the dashed rectangle is the MBR of the polygonal uncertainty region of an object o, and the rectangle inside the polygon is the $o.pcr(0.25)$ of o (for clarity, the other PCRs are omitted). Let q_1 be a query with $p_{q1} = 0.8$ whose search region is the small grey box r_{q1}. Rectangle $o.pcr(0.25)$ is not contained in r_{q1}, which implies that (by Rule 1 of Observation 1) o does not qualify q_1 even if the query probability threshold *were* $1 - 0.25 = 0.75$, *let alone a larger value 0.8*.

The value 0.25 used in the above example is the *smallest number in the U-catalog no less than* $1 - p_{q1} = 0.2$. Any value in the U-catalog smaller than 0.25 cannot be applied for pruning based on Rule 1. For example, $p_1 = 0.1$ is useful only for a query with a probability threshold at least $1 - 0.1 = 0.9$. On the other hand, although a value (e.g., $p_3 = 0.4$) larger than 0.25 can be applied, it is less effective. To understand this, note that $o.pcr(0.4)$ is necessarily covered by $o.pcr(0.25)$; thus, if $o.pcr(0.25)$ is contained in the query region (i.e., $o.pcr(0.25)$ cannot disqualify the object), so is $o.pcr(0.4)$ (it cannot, either). The reverse, however, is not true (i.e., $o.pcr(0.25)$ may still succeed in pruning even if $o.pcr(0.4)$ fails).

Next let us consider a query q_2 with $p_{q2} = 0.7$ and a search region r_{q2} (the left grey rectangle in Figure 4). We can validate o for q_2 without calculating its appearance probability. In fact, since r_{q2} covers the part $o.MBR$ on the left of the line passing the right boundary of $o.pcr(0.25)$, we can assert (by Rule 4 of Observation 1) that o appears in r_{q2} with a probability at least 0.75, i.e., larger than p_{q2}. Observe that, here, the selected value 0.25 is the *largest number in the U-catalog no greater than* $1 - p_{q2} = 0.3$. 0.25 can be verified to be the best in the U-catalog to perform the validation, following an analysis similar to the case of query q_1.

The above examples show that, using only a finite number of PCRs, we can still prune or validate objects by identifying the "appropriate" PCR. To successfully prune an object o, the selected PCR should allow us to verify that o cannot appear in the query region r_q even with a probability *lower than or equal to* the query threshold p_q. To validate o, we aim at establishing that o falls in r_q with a chance *higher than or equal to* p_q. The PCR selection reduces to picking the suitable value in the U-catalog. A special picking method is required for each of the 5 cases in Observation 1. The examples in Figure 4 explain the methods for the representative Rules 1 and 4, respectively. Since the rationale of the other cases is similar, we do not analyze them in detail, but simply list the resulting rules in a separate observation:

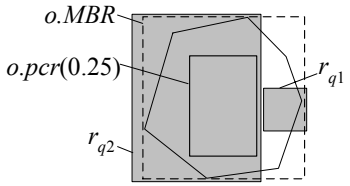


Figure 4: Illustration of Observation 2

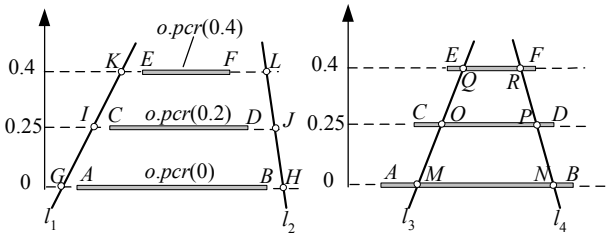
Observation 2. *For a prob-range query q with search region r_q and probability p_q:*

1. *For $p_q > 1 - p_m$, an object o can be eliminated if r_q does not fully contain $o.pcr(p_j)$, where p_j ($1 \le j \le m$) is the smallest value in the U-catalog not less than $1 - p_q$;*
2. *For $p_q \le 1 - p_m$, o can be pruned if r_q does not intersect $o.pcr(p_j)$, where p_j is the largest value in the U-catalog not greater than p_q.*
3. *An object is guaranteed to satisfy q if r_q fully covers the part of $o.MBR$ between planes $o.pcr_{i-}(p_j)$ and $o.pcr_{i+}(p_j)$ for some $i \in [1, d]$, where p_j is the largest value in the U-catalog not greater than $(1 - p_q)/2$;*
4. *For $p_q > 0.5$, the validating criterion is that r_q completely contains the part of $o.MBR$ on the right (or left) of $o.pcr_{i-}(p_j)$ (or $o.pcr_{i+}(p_j)$) for some $i \in [1, d]$, where p_j is the largest value in the U-catalog not greater than $1 - p_q$.*
5. *For $p_q \le 0.5$, the validating criterion is that r_q completely contains the part of $o.MBR$ on the left (or right) of $o.pcr_{i-}(p_j)$ (or $o.pcr_{i+}(p_j)$) for some $i \in [1, d]$, where p_j is the smallest value in the U-catalog not less than p_q.*

4.3 Conservative Functional Boxes

Although PCRs provide an efficient way for pruning objects, they are not suitable for indexing, since each entry in the resulting structure would need to record m PCRs, where m is the size of the U-catalog. Storing a PCR requires $2d$ values, and thus, each entry contains at least $2d \cdot m$ values, which renders the node fanout to decrease quickly as d increases, and compromises query performance. In the sequel, we present an approach that avoids this problem by storing the PCRs of an object in a compact manner.

Consider an object o whose pre-computed PCRs (at the values in the U-catalog) are $o.pcr(p_1)$, ..., $o.pcr(p_m)$. We aim at capturing these m rectangles using two functions $o.cfb_{out}$ and $o.cfb_{in}$, which are called the *outer* and *inner* conservative functional boxes, respectively. For each value p_j ($1 \le j \le m$), $o.cfb_{out}(p_j)$ returns a d-dimensional box that *contains* $o.pcr(p_j)$. The subscript *out* indicates that $o.cfb_{out}(p_j)$ bounds $o.pcr(p_j)$ from the "outside". Similarly, $o.cfb_{in}(p_j)$ produces a d-dimensional rectangle



(a) $o.cfb_{out}$ (b) $o.cfb_{in}$

Figure 5: Visualizing conservative functional boxes

that *is contained* in $o.pcr(p_j)$. Specifically, $o.cfb_{out}$ and $o.cfb_{in}$ have linear forms:

$$o.cfb_{out}(p) = \alpha_{out} - \beta_{out} \cdot p \quad (4)$$

$$o.cfb_{in}(p) = \alpha_{in} - \beta_{in} \cdot p \quad (5)$$

where α_{out}, β_{out}, α_{in}, and β_{in} are $2d$-dimensional vectors independent of p. In particular, α_{out} is essentially the d-dimensional rectangle $o.cfb_{out}(0)$, while β_{out} describes "how fast" $o.cfb_{out}(p)$ shrinks as p grows. The semantics of α_{in} and β_{in} are similar but with respect to $o.cfb_{in}$. Representing both $o.cfb_{out}$ and $o.cfb_{in}$ requires only $8d$ values (independent of the U-catalog size m), as opposed to $2d \cdot m$ mentioned earlier.

Conservative functional boxes can be best visualized for $d = 1$, where an object's uncertainty region is an interval. Assume that the U-catalog includes $m = 3$ values[2] $\{p_1 = 0, p_2 = 0.25, p_3 = 0.4\}$. Figure 5a shows the pre-computed 1D PCRs of an object o in a special two-dimensional space, where the horizontal axis captures the coordinates of uncertainty intervals, and the vertical axis indicates the probability values where these intervals are obtained. For example, the $pcr(p_1)$ is a segment AB whose projection on the vertical dimension equals $p_1 = 0$. Intervals CD and EF represent the PCRs at p_2 and p_3, respectively.

Function $o.cfb_{out}$ is illustrated using lines l_1 and l_2. Specifically, $o.cfb_{out}(p_1)$ corresponds to segment GH (i.e., a 1D rectangle), where G (H) is the intersection between l_1 (l_2) and the horizontal dimension. Similarly, $o.cfb_{out}(p_2)$ and $o.cfb_{out}(p_3)$ are intervals IJ and KL, respectively. Notice that $o.cfb_{out}(p_j)$ indeed contains $o.pcr(p_j)$, for $j = 1, 2, 3$. For $d = 1$, α_{out} and β_{out} in Equation 4 are 2D vectors; let us denote them as $\{\alpha^{1-}_{out}, \alpha^{1+}_{out}\}$ and $\{\beta^{1-}_{out}, \beta^{1+}_{out}\}$, respectively. Since $\alpha_{out} = o.cfb_{out}(0)$, interval $[\alpha^{1-}_{out}, \alpha^{1+}_{out}]$ is equivalent to segment GH. On the other hand, β^{1-}_{out} (β^{1+}_{out}) is determined by the slope of line l_1 (l_2)[3]. Figure 5b demonstrates $o.cfb_{in}$ using lines l_3 and l_4, such that $o.cfb_{in}(p_1)$, $o.cfb_{in}(p_2)$, $o.cfb_{in}(p_3)$ are segments MN, OP and QR, respectively. Notice that $o.cfb_{in}(p_j)$ is contained in $o.pox(p_j)$ for each $1 \leq j \leq 3$.

Both $cfb_{out}(p_j)$ and $cfb_{in}(p_j)$ of an object can be represented as $2d$-dimensional vectors. For example, the vector of $cfb_{out}(p_j)$ is:

$$\{cfb^{1-}_{out}(p_j), cfb^{1+}_{out}(p_j), ..., cfb^{d-}_{out}(p_j), cfb^{d+}_{out}(p_j)\} \quad (6)$$

where interval $[cfb^{i-}_{out}(p_j), cfb^{i+}_{out}(p_j)]$ is the projection of (rectangle) $cfb_{out}(p_j)$ on the i-th dimension ($1 \leq i \leq d$). When it is not ambiguous, we use $cfb^{i-}_{out}(p_j)$ to denote the plane passing the left boundary of $cfb_{out}(p_j)$ on the j-axis (likewise, $cfb^{i+}_{out}(p_j)$ also captures a plane). The above definitions also apply to cfb_{in} in a straightforward manner.

To enable CFBs in query processing, we need to adapt the heuristics in Observation 2. The reason for introducing both $o.cfb_{out}$ and $o.cfb_{in}$ is that they are required for appropriately adapting different rules. We first present the resulting rules before providing explanation.

Observation 3. *Observation 2 is still correct with the following changes:*

- *In Rule 1, replace $o.pcr(p_j)$ with $o.cfb_{in}(p_j)$;*
- *In Rule 2, replace $o.pcr(p_j)$ with $o.cfb_{out}(p_j)$;*
- *In Rules 3 and 4, replace $o.pcr_{i-}(p_j)$ and $o.pcr_{i+}(p_j)$ with $o.cfb^{i-}_{out}(p_j)$ and $o.cfb^{i+}_{out}(p_j)$, respectively;*
- *In Rule 5, replace $o.pcr_{i-}(p_j)$ and $o.pcr_{i+}(p_j)$ with $o.cfb^{i-}_{in}(p_j)$ and $o.cfb^{i+}_{in}(p_j)$, respectively;*

The observation is a natural extension of Observation 2. To illustrate this, let us first focus on Rule 1. If $o.cfb_{in}(p_j)$ is not covered by the query region r_q, neither is $o.pcr(p_j)$ — recall that $o.cfb_{in}(p_j)$ is contained $o.pcr(p_j)$. According to Observation 2, this indicates that o is not a qualifying object, thus justifying the first bullet of Observation 3. For Rule 2, when $o.cfb_{out}(p_j)$ does not intersect r_q, $o.pcr(p_j)$ must be disjoint with r_q too (since $o.cfb_{out}(p_j)$ encloses $o.pcr(p_j)$). In this case, by Observation 2, o can also be eliminated, confirming the second case in Observation 3. The modifications to Rules 3-5 (for validating a qualifying object) follow the same idea.

Although (compared to Observation 2) Observation 3 has weaker pruning/validating power, it requires only the CFBs of an object o (instead of the m rectangles $o.pcr(p_1)$, ..., $o.pcr(p_m)$), and thus reduces space consumption. The space saving increases node fanout in the corresponding index structure (the topic of Section 5), which in turn improves query performance. It is worth mentioning that a more sophisticated function could also be selected for CFBs, as long as the conservative properties are preserved (e.g., $o.cfb_{out}(p_j)$ should always enclose $o.pcr(p_j)$ for $l = 1, ..., m$). For example, instead of using a linear form, one could represent $o.cfb_{out}(p)$ using a quadratic function of p so that $o.cfb_{out}(p_j)$ bounds $o.pcr(p_j)$ more tightly. While this approach enhances the pruning effect of Observation 3, it also increases the storage space of CFBs, and adversely affects query efficiency. Furthermore, as will be elaborated in Section 5, a linear CFB offers considerable convenience in updating the index of uncertain data, which would be difficult for other representations.

[2]We use a small value $m = 3$ here for simplicity, but a practically suitable m is around 10, as shown in the experiments.

[3]Precisely, β^{1-}_{out} is the inverse of the slope of l_1, and β^{1+}_{out} is the negative inverse of the slope of l_2.

4.4 Finding Conservative Functional Boxes

Next, we elaborate the computation of CFBs. A good cfb_{out} should be such that $cfb_{out}(p_j)$ (the rectangle output by cfb_{out} with parameter p_j) is similar to the $pcr(p_j)$ of the corresponding object, for each p_j ($1 \leq j \leq m$) in the U-catalog. Since $cfb_{out}(p_j)$ always covers $pcr(p_j)$, a natural goal in optimizing cfb_{out} is to minimize[4]:

$$\sum_{j=1}^{m} MARGIN(cfb_{out}(p_j)) \tag{7}$$

where $MARGIN$ is the perimeter of a d-dimensional rectangle. Using the vector representation of cfb_{out} in Formula 4, we can derive Equation 7 as:

$$\sum_{j=1}^{m}\left(\sum_{i=1}^{d}\left(cfb_{out}^{i+}(p_j) - cfb_{out}^{i-}(p_j)\right)\right) \tag{8}$$

Recall that, as in Equation 4, cfb_{out} is decided by two $2d$-dimensional vectors α_{out} and β_{out}, whose components are listed as: $\{\alpha_{out}^{1-}, \alpha_{out}^{1+}, ..., \alpha_{out}^{d-}, \alpha_{out}^{d+}\}$ and $\{\beta_{out}^{1-}, \beta_{out}^{1+}, ..., \beta_{out}^{d-}, \beta_{out}^{d+}\}$, respectively. According to Equation 4, for each $i \in [1, d]$, we have:

$$cfb_{out}^{i-}(p) = \alpha_{out}^{i-} - \beta_{out}^{i-} \cdot p \tag{9}$$
$$cfb_{out}^{i+}(p) = \alpha_{out}^{i+} - \beta_{out}^{i+} \cdot p \tag{10}$$

Thus, Formula 8 becomes:

$$\sum_{i=1}^{d}\left(\sum_{j=1}^{m}\left(\alpha_{out}^{i+} - \beta_{out}^{i+} \cdot p_j - \alpha_{out}^{i-} + \beta_{out}^{i-} \cdot p_j\right)\right)$$
$$= \sum_{i=1}^{d}\left(\sum_{j=1}^{m}\left(\alpha_{out}^{i+} - \beta_{out}^{i+} \cdot p_j\right) - \sum_{j=1}^{m}\left(\alpha_{out}^{i-} - \beta_{out}^{i-} \cdot p_j\right)\right)$$
$$= \sum_{i=1}^{d}\left(m \cdot \alpha_{out}^{i+} - \beta_{out}^{i+} \cdot P - m \cdot \alpha_{out}^{i-} + \beta_{out}^{i-} \cdot P\right)$$

where P is a constant equal to $\sum_{j=1}^{m} p_j$ (i.e., the sum of all the values in the U-catalog). The above equation is minimized when

$$m \cdot \alpha_{out}^{i+} - \beta_{out}^{i+} \cdot P - m \cdot \alpha_{out}^{i-} + \beta_{out}^{i-} \cdot P \tag{11}$$

takes the smallest value for each $i \in [1, d]$. Without loss of generality, next we consider $i = 1$, and discuss the computation of α_{out}^{1-}, α_{out}^{1+}, β_{out}^{1-}, β_{out}^{1+} that minimize Formula 11. The solution can be applied to find the best α_{out}^{i-}, α_{out}^{i+}, β_{out}^{i-}, β_{out}^{i+}. Combining the solutions for all $i = 1, ..., d$, the resulting vectors α_{out} and β_{out} achieve the minimum for Formula 7, and therefore, produce the best cfb_{out}.

The 4 variables α_{out}^{1-}, α_{out}^{1+}, β_{out}^{1-}, β_{out}^{1+} are not arbitrary, but confined by several linear constraints. First, for each p_j ($1 \leq j \leq m$), interval $[cfb_{out}^{1-}(p_j), cfb_{out}^{1+}(p_j)]$ should always cover the projection of the $pcr(p_j)$ of the corresponding object on the first dimension. Denoting this projection as $[pcr_{1-}(p_j), pcr_{1+}(p_j)]$, we have:

$$cfb_{out}^{1-}(p_j) = \alpha_{out}^{1-} - \beta_{out}^{1-} \cdot p_j \leq pcr^{1-}(p_j) \tag{12}$$
$$cfb_{out}^{1+}(p_j) = \alpha_{out}^{1+} - \beta_{out}^{1+} \cdot p_j \geq pcr^{1+}(p_j) \tag{13}$$

for each $j \in [1, m]$. Therefore, discovering α_{out}^{1-}, α_{out}^{1+}, β_{out}^{1-}, β_{out}^{1+} that minimize Formula 11 can be cast as a *linear programming* problem, subject to the $2m$ linear constraints shown in inequalities 12 and 13. Linear programming has been very well studied and numerous efficient solutions exist. In our implementation, we adopt the well-known Simplex [7] method.

So far we have focused on computing cfb_{out}, while a similar approach can be utilized to obtain cfb_{in}. Since $cfb_{in}(p_j)$ is always enclosed by $pcr(p_j)$, we aim at *maximizing* a metric identical to Formula 7, replacing the subscript "out" with "in". This problem is also an instance of linear programming, where the objective is to maximize Formula 11 based on constraints in inequalities 12 and 13 with the following modifications: (i) all subscripts are changed to "in", (ii) in inequality 12, sign "$\leq$" should be "$\geq$", and (iii) in inequality 13, "$\geq$" should be "$\leq$". Unlike discovering cfb_{out}, finding cfb_{in} requires another type of constraints capturing $cfb_{in}^{1-}(p_j) \leq cfb_{in}^{1+}(p_j)$ for $1 \leq j \leq m$:

$$\alpha_{in}^{1-} - \beta_{in}^{1-} \cdot p_j \leq \alpha_{in}^{1+} - \beta_{in}^{1+} \cdot p_j \tag{14}$$

In fact, when an object's pdf (e.g., Gaussian) is symmetric with respect to the center of the uncertainty region, $o.cfb_{in}(p_j)$ and $o.cfb_{out}(p_j)$ are also symmetric (by the center) for all $j \in [1, m]$. Hence, the size of representing a CFB can be cut by half (e.g., in Figure 5a, if l_1 and l_2 are symmetric, then only one line needs to be kept). Note that the time for computing the conservative functional boxes of an object is a one-time cost, since the CFBs *need to be computed only once* (at the time the object is inserted into the database). The resulting CFBs are then managed by an efficient structure introduced in the next section.

5 The U-Tree

Based on the discussion in the previous section, we can precompute the $o.cfb_{out}$ and $o.cfb_{in}$ of all objects o, and process a prob-range query based on sequential scan. Specifically, given a query q, the filter step inspects each object in turn, and attempts to prune it using Observation 3. Objects that cannot be eliminated this way are candidates, whose appearance probabilities must be computed from their pdfs in the refinement step. In this section, we present the U-tree, an index structure designed to accelerate the filter step. Section 5.1 explains the structure of a U-tree and its properties, and Section 5.2 elaborates the algorithm for prob-range search. Section 5.3 presents the incremental update algorithms.

[4] An alternative choice is to minimize the sum of *areas* of $cfb_{out}(p_j)$ for $1 \leq j \leq m$. We choose margin because a rectangle with a low margin also has a small area, but not the vice versa.

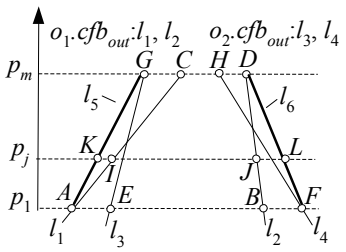


Figure 6: Representation of a non-leaf entry

5.1 Structure Overview and Properties

A U-tree is designed for pruning subtrees that do not contain any results (the structure does not accelerate the validating process, which requires the detailed information of individual objects stored in leaf nodes). A leaf entry contains the $o.cfb_{out}$ and $o.cfb_{in}$ of an object o, the MBR of its uncertainty region $o.ur$, together with a disk address where the the details of $o.ur$ and the parameters of $o.pdf$ are stored.

An intermediate entry e_I carries a pointer referencing its child node, and two d-dimensional rectangles $e.MBR_{\perp}$ and $e.MBR_{\top}$. Specifically, $e.MBR_{\perp}$ is the MBR (minimum bounding rectangle) of $o.cfb_{out}(p_1)$ of all the objects o in the subtree of e, where p_1 is the smallest value in the U-catalog. The $e.MBR_{\top}$ is similarly defined but with respect to $o.cfb_{out}(p_m)$, where p_m is the largest in the U-catalog.

Figure 6 shows a 1D example that illustrates the cfb_{out} of two objects o_1 and o_2 (in a way similar to Figure 5). Specifically, lines l_1 and l_2 represent function $o_1.cfb_{out}(p)$, segment AB corresponds to $o_1.cfb_{out}(p_1)$, and segment CD to $o_1.cfb_{out}(p_m)$. Likewise, lines l_3 and l_4 capture $o_2.cfb_{out}(p)$, $o_2.cfb_{out}(p_1) = EF$, and $o_2.cfb_{out}(p_m) = GH$. Assume that o_1 and o_2 are the only two objects in the subtree of an intermediate entry e. Then, $e.MBR_{\perp}$ is interval AF (i.e., the MBR of AB and EF), while $e.MBR_{\top}$ is GD (the MBR of CD and GH).

Based on $e.MBR_{\perp}$ and $e.MBR_{\top}$, we define a linear function of p for e:

$$e.MBR(p) = \alpha - \beta \cdot p \tag{15}$$

where α and β are $2d$-dimensional vectors resulting in $e.MBR(p_1) = e.MBR_{\perp}$ and $e.MBR(p_m) = e.MBR_{\top}$. The two vectors can be uniquely solved as (considering $p_1 = 0$): $\alpha = e.MBR_{\perp}$, and $\beta = (e.MBR_{\perp} - e.MBR_{\top})/p_m$. It is important to note that α and β are *not* physically stored; instead, they are derived from $e.MBR_{\perp}$ and $e.MBR_{\top}$ whenever necessary. In Figure 6 (where $e.MBR_{\perp}$ is segment AF and $e.MBR_{\top}$ is GD), function $e.MBR(p)$ is decided by two segments l_5, l_6, where l_5 connects points A, G, and l_6 links D, F. For the p_j shown in the example, $e.MBR(p_j)$ returns a segment KL, where point K (L) is the intersection of line l_5 (l_6) with the horizontal line $p = p_j$.

Without loss of generality, let e be an intermediate entry in the U-tree, and o be any object in its subtree. *Then, for any value p_j ($1 \leq j \leq m$) in the U-catalog, $e.MBR(p_j)$ always covers $o.cfb_{out}(p_j)$.* In Figure 6, for instance, the $o_1.cfb_{out}(p_j)$ of o_1 equals segment IJ, which is indeed enclosed in $e.MBR(p_j) = IL$. This property leads to an efficient algorithm for prob-range queries, as discussed shortly.

We point out that an intermediate entry does not contain any information about the cfb_{in} of the objects in its subtree. Indeed, a U-tree is constructed solely based on cfb_{out}. As will be elaborated in Section **??**, although preserving cfb_{in} in non-leaf levels may reduce query costs, it significantly complicates the resulting structure, as well as its update algorithms.

5.2 Prob-Range Query Algorithm

We provide an observation for pruning subtrees that do not contain qualifying objects.

Observation 4. *For a prob-range query q with search region r_q and probability p_q, the subtree of an intermediate entry e can be pruned if r_q does not intersect $e.MBR(p_j)$ (for some $j \in [1, m]$), where $e.MBR(.)$ is a function as in Equation 15, and p_j is the largest value in the U-catalog satisfying $p_j \leq p_q$.*

To establish the correctness of this heuristic (i.e., it does not generate any false negatives), we will show: no object o in the subtree of e can satisfy query q if the search region r_q is disjoint with $e.MBR(p_j)$ (p_j is selected as above). This is relatively obvious if the probability threshold p_q does not exceed $1 - p_m$. Specifically, as mentioned earlier, $o.cfb_{out}(p_j)$ is totally contained in $e.MBR(p_j)$. Since $e.MBR(p_j)$ does not intersect r_q, $o.cfb_{out}(p_j)$ must also be disjoint with r_q. Notice that the value of p_j here is identical to that in Rule 2 of Observation 3, which asserts that o is not a query result.

When $p_q > 1 - p_m$, we utilize the fact that all values in the U-catalog do not exceed 0.5, leading to: $p_q > 1 - p_m \geq 0.5 \geq p_m$. Therefore, the p_j in Observation 4 is necessarily p_m (the largest value in the U-catalog). Consider an alternative query q' whose probability threshold $p_{q'}$ equals p_m, and its search region is that of q (i.e., disjoint with $e.MBR(p_m)$). Since $p_m \leq 1 - p_m$, the analysis earlier shows that no object o in the subtree of e can possibly satisfy q'. Since any qualifying object for q must at least satisfy q' (due to $p_q > p_{q'}$), we guarantee that the subtree of e does not have any result for q either.

We are ready to discuss the prob-range algorithm. The search starts from the root, and eliminates its entries according to Observation 4. For each remaining entry, we retrieve its child node, and perform the above process recursively until a leaf node is reached. For an object o encountered, we first attempt to prune or validate it using Observation 3. In case o can neither be eliminated or asserted as a result, it is added to a candidate set S_{can} together with the disk address storing its $o.ur$ and $o.pdf$. After the necessary nodes in the U-tree have been visited, we start the refinement step for processing S_{can}. In this phase, elements in S_{can} are first grouped by their associated disk addresses. For each address, one I/O is performed to load the detailed information of all relevant candidates, whose appearance probabilities are then computed.

5.3 Dynamic Update Algorithms

The U-tree shares a common rationale with the R-tree: an intermediate entry always "bounds" the entries in its subtree. Specifically, let e be an intermediate entry, whose child node is a non-leaf node. Then, $e.MBR_{\perp}$ ($e.MBR_{\top}$) is the MBR of those of all entries in its child node. Hence, the information of a non-leaf entry can be decided directly from its child node, without accessing any object further down its subtree. Furthermore, if we want to insert a new object o into the subtree of an entry e, the new $e.MBR_{\perp}$ (after incorporating o) equals the union of the old $e.MBR_{\perp}$ with $o.cfb_{out}(p_1)$, and similarly, $e.MBR_{\top}$ should be updated to the union of itself with $o.cfb_{out}(p_m)$.

Observation 4 implies that, to maximize query effectiveness, we should minimize the rectangles returned by $e.MBR(p_j)$ for all $j = 1, ..., N$. To achieve this, we adapt the update algorithms of the R*-tree (introduced in Section 2.2) to U-trees. The core of adaptation is the optimization metric. Recall that the R* algorithms require four metrics: (i-ii) the margin (or area) of a MBR, (iii) the overlap between two MBRs, and (iv) the distance between the cetroids of two MBRs. These metrics are no longer suitable for U-trees because each entry has a more complex form.

We replace these metrics with their *summed counterparts*[5]. Given an intermediate entry e, its summed margin equals $\sum_{j=1}^{m} MARGIN(e.MBR(p_j))$, where function $e.MBR(.)$ is defined in Equation 15, p_j is the j-th value in the U-catalog, and $MARGIN$ gives the margin of rectangle $e.MBR(p_j)$. Similarly, a summed area is $\sum_{j=1}^{m} AREA(e.MBR(p_j))$. Given two non-leaf entries e_1, e_2, we compute their summed overlap and summed centroid distance as $\sum_{j=1}^{m} OVERLAP(e_1.MBR(p_j), e_2.MBR(p_j))$ and $\sum_{j=1}^{m} CDIST(e_1.MBR(p_j), e_2.MBR(p_j))$ respectively, where $OVERLAP$ ($CDIST$) calculates the overlapping area (centroid distance) between two rectangles. The U-tree aims at minimizing these "summed metrics" with the reasoning that, a good intermediate entry e should lead to a small rectangle $e.MBR(p_j)$ for all $j = 1, ..., m$.

Each insertion/deletion in a U-tree is performed in exactly the same way as the R*-tree, except that each metric is replaced with its summed counterpart. The only exception lies in the split algorithm (handling a node overflow). Recall that, in the R*-tree [1], a node split is performed in two steps, which select a split axis, and decide the actual entry distribution, respectively. Each step relies on sorting the coordinates of the MBRs in the node. In the U-tree, the sorting must be performed in an alternative manner due to the difference in entry representation.

Intuitively, a good split should be such that, the parent entries e_1 and e_2 of the resulting nodes have small $MBR(p)$ for all values $p = p_1, ..., p_m$ in the U-catalog. Therefore, ideally, the best split should be obtained by performing a sorting at each p_j ($1 \leq j \leq m$) which, unfortunately, incurs expensive overhead. We avoid so many sorting operations using a simple heuristic that examines only the median value $p_{\lceil m/2 \rceil}$ in the U-catalog. Specifically, given an overflowing leaf (or intermediate) node, we first compute the $e.MBR(p_{\lceil m/2 \rceil})$ of all entries e contained. Then, the entry distribution after splitting is decided using the R*-split, passing all the rectangles obtained in the previous step. Note that considering all values in the U-catalog in the other update procedures is feasible, because the cost of calculating a summed metric is trivial, and no sorting is required. Finally, although the above discussion uses intermediate levels as examples, it also applies to leaf nodes, by replacing function $e.MBR(.)$ with $o.cfb_{out}(.)$ of an object.

[5] A similar technique was applied in [11] to convert R*-trees to a spatio-temporal index.

6 Experiments

This section experimentally evaluates the efficiency of the proposed techniques. We create uncertain data to simulate location-based service environments (Figure 1). For this purpose, we select two real spatial datasets *LB* and *CA*, which contain 53k and 62k points representing locations in the Long Beach county and California, respectively [6]. All dimensions are normalized to have domains [0, 10000].

Each data point p generates an uncertain object o. The uncertainty region $o.ur$ is a circle centering at p with radius 250 (i.e., 2.5% of the length of an axis). The pdf of o is *Uniform* or *Constrained-Gaussian* (*Con-Gau* for short). Specifically, for *Uniform*, o falls at each position in $o.ur$ with equal probability. The definition of *Con-Gau* is based on the traditional Gaussian distribution which, however, has an infinite input universe (in our case, o must be limited to $o.ur$). Hence, given the pdf $pdf_G(x)$ of Gaussian[7], we first calculate the value $\lambda = \int_{x \in o.ur} pdf_G(x)dx$, and then formulate pdf_{CG} as:

$$pdf_{CG} = \begin{cases} pdf_G(x)/\lambda & \text{if } x \in o.ur \\ 0 & \text{otherwise} \end{cases} \quad (16)$$

The variance of $pdf_G(x)$ used to define pdf_{CG} is set to 125^2 (i.e., the standard deviation 125 is half the uncertainty region's radius). We convert *LB* and *CA* into uncertain datasets by applying *Uniform* and *Con-Gau* on their objects, respectively. Note that the λ in Equation 16 is identical for all the data items in *CA*, and needs to be calculated only once.

In order to investigate our solutions in 3D space, we generate another *Aircraft* dataset as follows. First, 2000 points are sampled from *LB* to serve as "airports". The "reported location" of an "airplane" consists of 3 values a, b, c, corresponding to its spatial location (a, b) and altitude c. To obtain the first two numbers, we randomly choose two airports as the aircraft's source and destination; then (a, b) is set to a random point on the segment connecting the two

[6] Available at http://www.census.gov/geo/www/tiger/.

[7] If x is a 2D point with coordinates (a, b), then $pdf_G(x) = \frac{1}{2\pi\sigma^2} e^{-[(a-\mu_a)^2+(b-\mu_b)^2])/2\sigma^2}$, where σ^2 is the variance, and μ_a, μ_b are the means of a, b, respectively.

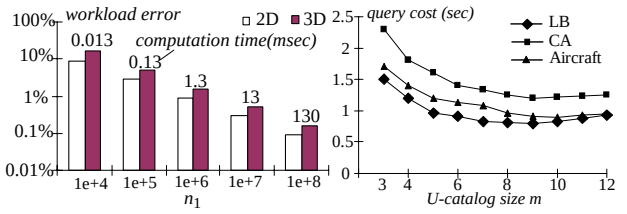


Figure 7: Cost of numerical evaluation

Figure 8: Tuning the catalog size for U-PCR

airports. The value c is uniformly obtained in a (normalized) range [0, 10000]. *Aircraft* contains 100k aircrafts thus obtained, whose uncertainty regions are spheres centering at their reported locations with radius 125. The pdfs deployed are *Uniform*.

The search region of a query is a square/cube with side length qs, and the distribution of the region's location (in the data space) follows that of the underlying data. A workload contains 100 queries with the same parameters qs and p_q (i.e., the probability threshold). Since there is no existing structure for indexing multi-dimensional uncertain data, we compare the U-tree with its variation (called "U-PCR") that stores the PCRs in (leaf and intermediate) entries, as opposed to CFBs. The performance of a structure is measured as the average (I/O and CPU) time of answering all the queries in a workload. All the experiments are performed using a machine with a Pentium III CPU of 800 MHz. The page size is fixed to 4096 bytes.

6.1 Cost of Computing Appearance Probability

We first evaluate the cost of computing the appearance probability of an object using the monte-carlo method discussed in Section 3. The efficiency of monte-carlo depends on the number (n_1 in Equation 3) of points generated in an object's uncertainty region. The goal is to identify the lowest n_1 that leads to accurate results. We use a workload where the query regions have the same size $qs = 500$ (5% of the length of a dimension), but have different intersections with an object's uncertainty region (a 2D circle or 3D sphere). The object's appearance probability for each query is estimated using different values of n_1, and the relative error[8] of each estimate is calculated with respect to the *true* value (obtained with an extremely large n_1). The workload error is the average error of all the queries involved.

Figure 7 shows the workload error as n_1 increases (the accuracy is related only to the area/volume of the uncertainty region, and is independent of the concrete pdf). The numbers on top of the columns indicate the time (in milliseconds) of computing a single probability. Clearly, in 2D space, n_1 must be at least 10^6 to guarantee highly accurate results (with error less than 1%), and the corresponding number is even higher in 3D space (where an uncertainty region is "larger"). In the following experiments, we set n_1 to 10^6 for both 2D and 3D, in which case each application of monte-carlo requires 1.3 milliseconds.

[8]The relative error equals $|act - est|/act$, where act and est are the actual and estimated values, respectively.

	LB	*CA*	*Aircraft*
U-PCR	11.9M	14.0M	40.1M
U-tree	5.0M	5.9M	14.2M

Table 1: Size comparison (bytes)

6.2 Tuning the Catalog Size

The performance of U-PCR is determined by the number m of values in its U-catalog. The second set of experiments aims at identifying the best m that maximizes the effectiveness of U-PCR. Specifically, the catalog contains values 0, $\frac{0.5}{m-1}$, $\frac{1}{m-1}$, ..., 0.5 (recall that all numbers must be in the range [0, 0.5]). For each dataset (*LB*, *CA*, *Aircraft*), we create U-PCR trees with m varied from 3 to 12. The efficiency of each tree is examined with 80 workloads that have $qs = 500$, and their p_q equals 0.11, 0.12, ..., 0.89, 0.9, respectively. Figure 8 plots the average query time of these workloads as a function of m.

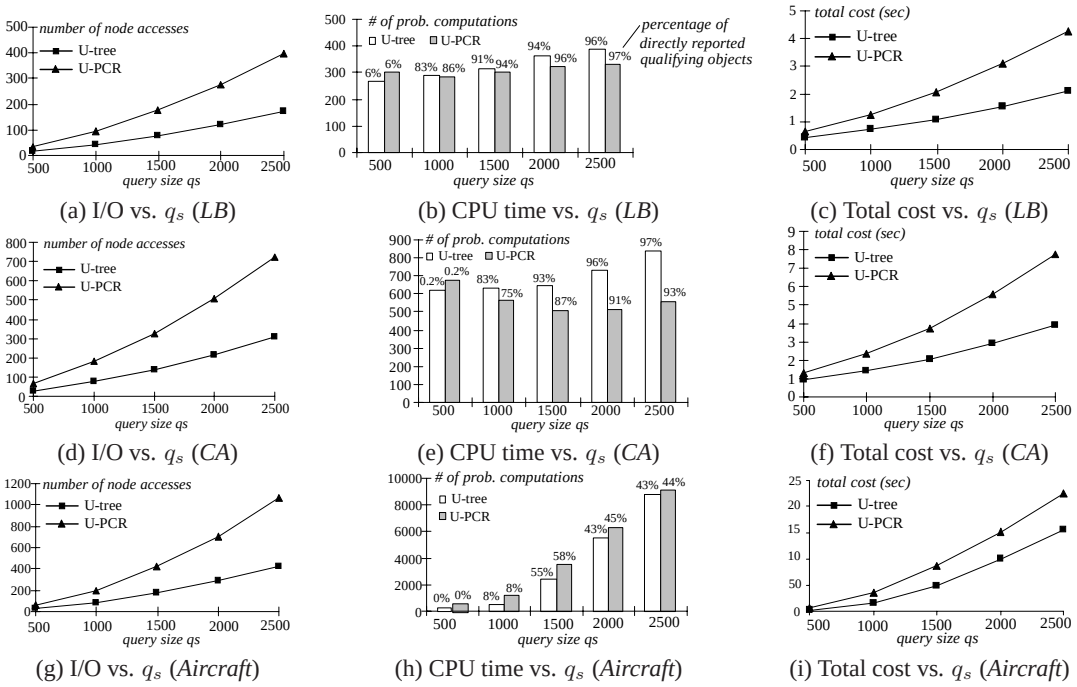
U-PCR initially improves as m increases, but deteriorates as m grows beyond a certain threshold. This is not surprising because a U-PCR with a higher m retains more PCRs in each entry, which permit pruning/validating a greater number of objects directly (without evaluating their appearance probabilities), resulting in less CPU cost. However, a larger catalog size also decreases the node fanout, leading to more page accesses in processing a query. For *LB* and *CA*, the best performance is obtained with $m = 9$, while the optimal m for *Aircraft* equals 10. We use these values in the rest of the experiments.

The catalog tuning for U-trees is much easier. The only disadvantage of using a large size m is that it will compromise the update performance (recall that each object insertion needs to derive m PCRs). As will be shown in Section 6.4, however, the overhead of each PCR computation is low, which allows the U-tree to utilize a sizable catalog — an important advantage of U-trees over U-PCR. In the sequel, we set the U-tree catalog size to 15 (the catalog values are 0, 1/28, ..., 14/28).

6.3 Space Consumption and Query Performance

Table 1 compares the space consumption of the U-tree and U-PCR for various datasets. As expected, U-trees are much smaller due to their greater node capacities. Specifically, each U-tree entry stores at most two CFBs that are represented with totally 16 (24) values in 2D (3D) space, as opposed to 36 (60) values in each U-PCR entry (for recording numerous PCRs). Note that the size of a U-tree is *not* affected by its catalog size.

In Figure 9a, we illustrate the number of page accesses of using the U-tree and U-PCR (on *LB*) to answer workloads whose p_q equals 0.6, and their qs (size of a query region) changes from 500 to 2500. The U-tree significantly outperforms its competitor in all cases, again due to its much larger node fanout. Figure 9b shows the CPU costs in the previous experiments, measured in the average number of appearance probability computations in a query. Each percentage in this diagram indicates the average percentage of qualifying objects which are directly validated by the U-tree/U-PCR in a query. For example, a 90% means that,



(a) I/O vs. q_s (*LB*) (b) CPU time vs. q_s (*LB*) (c) Total cost vs. q_s (*LB*)

(d) I/O vs. q_s (*CA*) (e) CPU time vs. q_s (*CA*) (f) Total cost vs. q_s (*CA*)

(g) I/O vs. q_s (*Aircraft*) (h) CPU time vs. q_s (*Aircraft*) (i) Total cost vs. q_s (*Aircraft*)

Figure 9: Effects of search region sizes on query performance ($p_q = 0.6$)

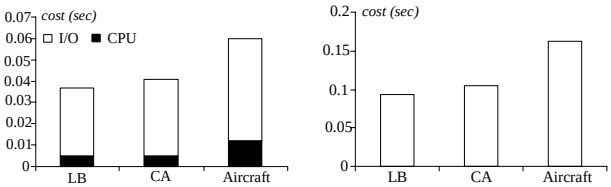
among all the objects satisfying a query, on average only 10% have their appearance probabilities calculated. The CPU overhead of the U-tree is slightly higher because pruning/validating with CFBs is less efficient than with PCRs. The total costs (including both I/O and CPU time) of the two methods are compared in Figure 9c.

The same experiments are also performed for datasets *CA* and *Aircraft* respectively, and their results are presented in Figures 9d-9i, confirming similar observations. The only exception is that in Figure 9h, both methods incur low CPU time for qs equal to 500 and 1000 because the queries in these two workloads have fairly small result sizes. Furthermore, the U-tree has better CPU performance than U-PCR. The reason is that, the CFBs used by the U-tree (for pruning/validating) turn out to be tighter than the PCRs utilized by U-PCR (note that these CFBs and PCRs are defined at different probability values).

Figure 10 illustrates the results of the experiments using workloads whose qs is fixed to the median value 1500, and their q_p falls in the range from 0.3 to 0.9. Each row of the figure contains, for one dataset, three diagrams demonstrating the average I/O cost, number of probability evaluations, and execution time of a query. U-trees are better than U-PCR in terms of overall performance.

6.4 Update Overhead

The last set of experiments evaluates the update performance of the U-tree. Figure 11a shows the average cost of an insertion during the index construction for *LB*, *CA*, and *Aircraft*, respectively. Each cost is broken down into the I/O and CPU overhead, respectively. In particular, the CPU time essentially corresponds to the combined cost of (i) the simplex algorithm (for computing CFBs; see Section 4.4), and (ii) calculating the necessary PCRs. Then, we remove all the objects from each U-tree, and measure the amortized cost of a deletion. The results are demonstrated in Figure 11b (CPU time is omitted as it is negligible).



(a) Insertion (b) Deletion

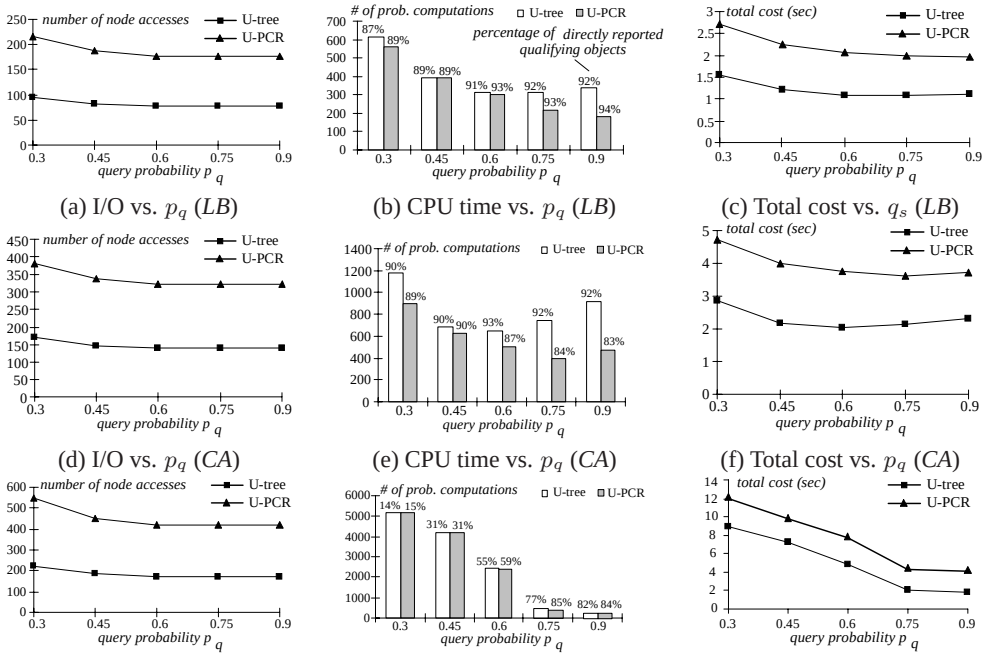
Figure 11: The update overhead

7 Conclusions and Future Work

In this paper, we presented a careful study of the probabilistic range search problem on uncertain data. Our solutions can be applied to objects described by arbitrary pdfs, and process queries efficiently with small space. This work also lays down a solid foundation for further research on uncertain databases. An interesting issue is to investigate the algorithms that deploy U-trees to solve other types of queries (e.g., those defined in [4]). Another exciting direction for future work is to derive analytical models [12] that can accurately estimate the query costs. Such models can be utilized to facilitate query optimization, which is also an important topic to be studied.

Acknowledgements

Yufei Tao and Xiaokui Xiao were supported by Grant CityU 1163/04E from the Research Grants Council (RGC) of HKSAR, Reynold Cheng and Sunil Prabhakar by NSF grants IIS 9985019 and CCR-0010044, and Wang Kay



(a) I/O vs. p_q (*LB*) (b) CPU time vs. p_q (*LB*) (c) Total cost vs. q_s (*LB*)

(d) I/O vs. p_q (*CA*) (e) CPU time vs. p_q (*CA*) (f) Total cost vs. p_q (*CA*)

(g) I/O vs. p_q (*Aircraft*) (h) CPU time vs. p_q (*Aircraft*) (i) Total cost vs. p_q (*Aircraft*)

Figure 10: Effects of probability thresholds on query performance ($q_s = 1500$)

Ngai and Ben Kao by RGC grant HKU 7040/02E. We thank the anonymous reviewers for their insightful comments.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[2] S. Berchtold, C. Bohm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *PODS*, pages 78–86, 1997.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree : An index structure for high-dimensional data. In *VLDB*, pages 28–39, 1996.

[4] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, pages 551–562, 2003.

[5] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *TKDE*, 16(9):1112–1127, 2004.

[6] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB*, pages 876–887, 2004.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, , and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[8] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[9] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance in spatial data structures. In *PODS*, pages 214–221, 1993.

[10] D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. In *SSD*, pages 111–132, 1999.

[11] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD*, pages 331–342, 2000.

[12] Y. Theodoridis and T. K. Sellis. A model for the prediction of R-tree performance. In *PODS*, pages 161–171, 1996.

[13] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

[14] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and imprecision in modeling the position of moving objects. In *ICDE*, pages 588–596, 1998.

[15] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3):257–387, 1999.