Range Search on Multidimensional Uncertain Data

15

YUFEI TAO and XIAOKUI XIAO
Chinese University of Hong Kong
and
REYNOLD CHENG
Hong Kong Polytechnic University

In an uncertain database, every object o is associated with a probability density function, which describes the likelihood that o appears at each position in a multidimensional workspace. This article studies two types of range retrieval fundamental to many analytical tasks. Specifically, a nonfuzzy query returns all the objects that appear in a search region r_q with at least a certain probability t_q. On the other hand, given an uncertain object q, fuzzy search retrieves the set of objects that are within distance ε_q from q with no less than probability t_q. The core of our methodology is a novel concept of "probabilistically constrained rectangle", which permits effective pruning/validation of nonqualifying/qualifying data. We develop a new index structure called the U-tree for minimizing the query overhead. Our algorithmic findings are accompanied with a thorough theoretical analysis, which reveals valuable insight into the problem characteristics, and mathematically confirms the efficiency of our solutions. We verify the effectiveness of the proposed techniques with extensive experiments.

Categories and Subject Descriptors: H.2.2 [**Database Management**]: Physical Design—*Access Methods*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Uncertain databases, range search

ACM Reference Format:

Tao, Y., Xiao, X., and Cheng, R. 2007. Range search on multidimensional uncertain data. ACM Trans. Datab. Syst. 32, 3, Article 15 (August 2007), 54 pages. DOI = 10.1145/1272743.1272745 http://doi.acm.org/10.1145/1272743.1272745

This work was sponsored by two CERG grants from the Research Grant Council of the HKSAR government. Specifically, Y. Tao and X. Xiao were supported by Grant CUHK 1202/06, and R. Cheng by Grant PolyU 5138/06E.

Authors' addresses: Y. Tao and X. Xiao, Department of Computer Science and Engineering, Chinese University of Hong Kong, New Territories, Hong Kong; email: {taoyf; xkxiao}@cse.cuhk.edu.hk; R. Cheng, Department of Computing, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong; email: csckcheng@comp.polyu.edu.hk.

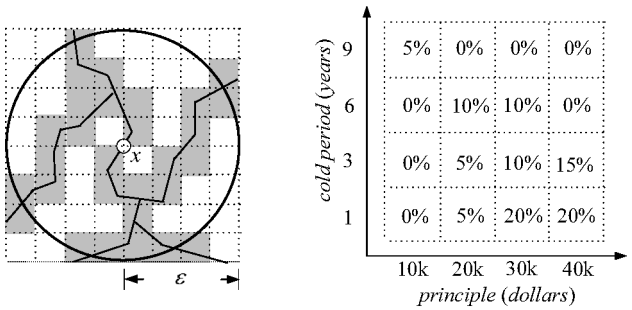
1. INTRODUCTION

Traditionally, a database describes objects with precise attribute values. Real-world entities, however, are often accompanied with uncertainty. Common sources of uncertainty include measurement errors, incomplete information (typically, missing data), variance in estimation from a random sample set, and so on. Furthermore, uncertainty may even be manually introduced in order to preserve privacy, which is the methodology behind "k-anonymity" [Sweeney 2002] and "location-privacy" [Cheng et al. 2006b]. In recent years, the database community has witnessed an increasing amount of research on modeling and manipulating uncertain data, due to its importance in many emerging and traditional applications.

Consider a meteorology system that monitors the temperature, relative humidity, and pollution level at a large number of sites. The corresponding readings are taken by sensors in local areas, and transmitted to a central database periodically (e.g., every 30 minutes). The database has a 3-value tuple for each site, which, however, may not exactly reflect the current conditions. For instance, the temperature at a site may have changed since it was measured. Therefore, various probabilistic models should be deployed to capture different attributes more accurately. For example, the actual temperature may be assumed to follow a Gaussian distribution with a mean calculated based on the last reported value (e.g., in the daytime, when temperature is rising, the mean should be set higher than the latest sensor reading).

In some scenarios, an object cannot be represented with a "regular" model (like uniform, Gaussian, and Zipf distributions, etc.), but demands a complex probability density function (pdf) in the form of a histogram. This is true in location-based services, where a server maintains the locations of a set of moving objects such as vehicles. Each vehicle o sends (through a wireless network) its current location, whenever it has moved away from its previously updated position x by a certain distance ε [Wolfson et al. 1999]. Therefore, at any time, the server does not have the precise whereabout of o, except for the fact that o must be inside a circle centering at x with radius ε, as shown in Figure 1(a). Evidently, o cannot appear anywhere in the circle, since it is constrained by the underlying road network, illustrated with the segments in Figure 1(a). The distribution of o can be approximated using a grid, where o can fall only in the grey cells that intersect the circle and the network simultaneously. The probability that o is covered by a particular cell is decided according to the application requirements. A simple choice is to impose an equal chance for all the grey cells, while more realistic modeling should take into account the distance between the cell and x, the speed limits of roads, and so on.

Unlike in the above environments, where uncertainty is caused by the delay in database updates, the raw data in some applications is inherently imprecise. Imagine a recommender company that assists clients to make promising investment plans, based on their preferences on the principle amount, the number of years before advantageous gains (i.e., the "cold period duration"), etc. In reality, it is difficult, or simply impossible, for a customer to specify unique values for these attributes. For instance, the amount of principle s/he is willing



(a) The location of a vehicle (b) A customer's investment profile

Fig. 1. Probabilistic modeling of uncertain objects.

to lay down may fall in a wide range of [$10k, $40k]. Obviously, depending on the principle, her/his expectation for other attributes may also vary (e.g., with a large principle, it would be reasonable to anticipate a shorter cold period). Therefore, a preference profile is also an uncertain object whose pdf can be described by a histogram, as demonstrated in Figure 1(b). The percentage in each cell indicates the overall interest of the client in a plan whose (principle, cold-period) 2D representation falls in the cell. For instance, s/he would not favor small investments with short cold durations (which would involve high risk), or large investments with lengthy cold periods (it would not be worthwhile to have a huge amount of money nonspendable for a long time).

1.1 Motivation

As in traditional spatial databases, range search is also an important operation in the applications mentioned earlier, and the building block for many other analytical tasks. However, since data is uncertain, it is no longer meaningful to simply declare that an object o appears or does not appear in a query region r_q. Instead, the notion of "appearance" should be accompanied with a value $Pr_{range}(o, r_q)$, capturing the probability that o falls in r_q, according to the uncertainty modeling of o. In practice, users are typically only concerned about events that may happen with a sufficiently large chance, that is, objects o whose $Pr_{range}(o, r_q)$ is at least a certain threshold t_q.

The above requirements lead to a novel type of queries: *probability thresholding range retrieval*. For instance, in the meteorology system discussed before, such a query would "find every high-fire-hazard site satisfying the following condition with at least 50% probability: its temperature is at least 86F degrees, humidity at most 5%, and pollution index no less than 7 (on a scale from 1 to 10)". In this query, t_q equals 50%, and r_q is a 3D rectangle whose projections on the temperature, humidity, and pollution dimensions correspond to ranges [86F, ∞), [0, 5%], [7, 10] respectively. As another example, in the vehicle tracking application, a user may request to "identify all the cabs that are located in the downtown area ($= r_q$) with at least 80% ($= t_q$) likelihood".

Range search can become "fuzzier", when the query region itself is uncertain. Assume that we would like to retrieve every police car o qualifying the next

predicate with a chance no less than 30%: it is currently within 1 mile from the cab q having licence number NY3852. Here, the precise locations of o and q are unknown, but obey pdfs created as explained in Figure 1(a). Intuitively, for every possible location x of q, there is a query region which is a circle centering at x, and has a radius of 1 mile. A police car o qualifies the spatial predicate, as long as it falls in the circle. The complication, however, lies in the large number of search regions that must be considered: we have to exhaust the circles of all the x, in order to calculate the overall probability that the distance between o and q is bounded by 1 mile.

In general, given an uncertain object q, a probability threshold t_q, and a distance threshold ε_q, a *probability thresholding fuzzy range query* returns all the objects o fulfilling $Pr_{fuzzy}(o, q, \varepsilon_q) \geq t_q$, which represents the probability that o is within distance ε_q from q (in the previous example, q is the cab, t_q is 30%, and ε_q equals 1 mile). Fuzzy search is also common in recommender systems. Consider that, in the scenario of Figure 1(b), the company has designed an investment plan that requires a principle of 30k dollars, and its cold period may have a length of 2 (or 3) years with 75% (or 25%) probability (in practice, it may be difficult to conclude a unique cold period duration). Hence, the package is an uncertain object q that can be described by a histogram analogous to a customer profile in Figure 1(b). Then, the manager can identify the potential interested clients, by issuing, on the customer-profile database, a fuzzy query with q and suitable values for t_q and ε_q.

1.2 Contributions and Article Organization

Although range search on traditional "precise" data has been very well studied [Gaede and Gunther 1998], the existing methods are not applicable to uncertain objects, since they do not consider the probabilistic requirements [Cheng et al. 2004b]. As a result, despite its significant importance, multidimensional uncertain data currently cannot be efficiently manipulated. This article remedies the problem with a comprehensive analysis, which provides careful solutions to a wide range of issues in designing a fast mechanism for supporting (both nonfuzzy and fuzzy) range queries.

The core of our techniques is a novel concept of "probabilistically constrained rectangles" (PCRs), which are concise summaries of objects' probabilistic modeling. In terms of functionalities, PCRs are similar to minimum bounding rectangles (MBR) in spatial databases. Specifically, they permit the development of an economical filter step, which prunes/validates a majority of the nonqualifying/qualifying data. Therefore, the subsequent refinement phase only needs to inspect a small number of objects, by invoking more expensive procedures (which, in our context, include loading an object's pdf, and/or calculating its qualification probability). As expected, the pruning/validating heuristics with PCRs are considerably more complicated (than those with MBRs), due to the higher complexity of uncertain objects (than spatial data).

As a second step, we propose the U-tree, an index structure on multidimensional uncertain objects that is optimized to reduce the I/O cost of range queries. The U-tree leverages the properties of PCRs to effectively prune the subtrees

that cannot have any query result, and thus, limits the scope of search to a fraction of the database. The new access method is fully dynamic, and allows an arbitrary sequence of object insertions and deletions.

Finally, we accompany our algorithmic findings with a thorough performance analysis. Our theoretical results reveal valuable insight into the problem characteristics, and mathematically confirm the effectiveness of the proposed algorithms. In particular, we derive cost models that accurately quantify the overhead of range retrieval, and can be utilized by a query optimizer for tuning the parameters of a U-tree, and seeking a suitable execution plan.

The rest of the article is organized as follows. Section 2 formally defines probabilistic thresholding range search. Section 3 introduces PCRs and elaborates the pruning/validating strategies for nonfuzzy queries, while Section 4 extends the heuristics to fuzzy retrieval. Section 5 clarifies the details of the U-tree, as well as the query algorithms. Section 6 presents the theoretical analysis about the effectiveness of our solutions, and applies the findings to U-tree optimization. Section 7 contains an extensive experimental evaluation that demonstrates the efficiency of the proposed techniques. Section 8 surveys the previous work related to ours, and Section 9 concludes the article with directions for future work.

2. PROBLEM DEfiNITIONS

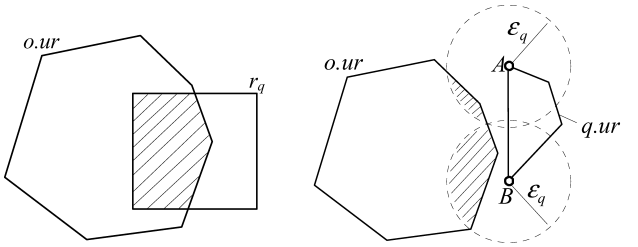
We consider a d-dimensional workspace, where each axis has a unit range [0, 1]. Each uncertain object o is associated with (i) a probability density function $o.pdf(x)$, where x is an arbitrary d-dimensional point, and (ii) an uncertainty region $o.ur$, which confines the area where the actual location of o could possibly reside. Specifically, the value of $o.pdf(x)$ equals 0 for any x outside $o.ur$, whereas $\int_{o.ur} o.pdf(x)\,dx$ equals 1. The pdfs of different objects are mutually independent.

We do not place any other constraint on objects' pdfs. In particular, various objects can have totally different pdfs, that is, an object may have a pdf of the uniform distribution, another could follow the Gaussian distribution, and yet another one could possess an irregular distribution that can only be described using a histogram (as in Figure 1). Furthermore, the uncertainty region of an object does not have to be convex, or can even be broken into multiple pieces (e.g., the object may appear inside two separate buildings, but not on the roads between the buildings).

Definition 1. Let S be a set of uncertain objects. Given a query region r_q, and a value $t_q \in (0, 1]$, a *nonfuzzy probability thresholding range query* returns all the objects $o \in S$ such that $Pr_{range}(o, r_q) \geq t_q$, where $Pr_{range}(o, r_q)$ is the appearance probability of o in r_q, and is computed as

$$Pr_{range}(o, r_q) = \int_{r_q \cap o.ur} o.pdf(x)\,dx. \tag{1}$$

The polygon in Figure 2(a) illustrates the uncertainty region $o.ur$ of an object o, and the rectangle corresponds to a query region r_q. If the possible location



(a) A nonfuzzy query (b) A fuzzy query (under the L_2 norm)

Fig. 2. Range search on uncertain data.

of o uniformly distributes inside $o.ur$, $Pr_{range}(o, r_q)$ equals the area of the intersection between $o.ur$ and r_q (i.e., the hatched region). In general, when r_q is an axis-parallel rectangle, we denote its projection on the ith axis ($1 \leq i \leq d$) as $[r_{q[i],-}, r_{q[i]+}]$. Such r_q is the focus of our analysis, because queries with these search regions are predominant in reality [Gaede and Gunther 1998]. Nevertheless, as elaborated later, our techniques can be extended to query regions of other shapes.

Definition 2. Let S be a set of uncertain objects, and q be another uncertain object that does not belong to S. Given a distance threshold ε_q, and a value $t_q \in (0, 1]$, a *fuzzy probability thresholding range query* returns all the objects $o \in S$ such that $Pr_{fuzzy}(o, q, \varepsilon_q) \geq t_q$, where $Pr_{fuzzy}(o, q, \varepsilon_q)$ is the probability that o and q have distance at most ε_q. Formally, if we regard o (q) as a random variable obeying a pdf $o.pdf(x)$ ($q.pdf(x)$), then

$$Pr_{fuzzy}(o, q, \varepsilon_q) = Pr\{dist(o, q) \leq \varepsilon_q\} \tag{2}$$

Since o and q are independent, it is not hard to see that Equation 2 can be re-written as

$$Pr_{fuzzy}(o, q, \varepsilon_q) = \int_{x \in q.ur} q.pdf(x) \cdot Pr_{range}(o, \odot(x, \varepsilon_q))\, dx. \tag{3}$$

where Pr_{range} is represented in Equation 1, and $\odot(x, \varepsilon_q)$ is a circle that centers at point x and has radius ε_q. As an example, the left and right polygons in Figure 2(b) demonstrate the uncertainty regions of a data object o and a query object q. The figure also shows the $\odot(x, \varepsilon_q)$, when x lies at point A and B, respectively. Again, for simplicity, assume that $o.pdf$ follows a uniform distribution inside $o.ur$. The area of the upper (lower) hatched region equals the probability $Pr_{range}(o, \odot(x, \varepsilon_q))$ for o and q to have a distance at most ε_q, when q is located at $x = A$ (B). In order to calculate $Pr_{fuzzy}(o, q, \varepsilon_q)$, (conceptually) we must examine the $Pr_{range}(o, \odot(x, \varepsilon_q))$ of all $x \in q.ur$.

Definition 2 is independent of the distance metric employed. Although our methodology is applicable to any metric, we derive concrete solutions for the L_∞ and L_2 norms,[1] which are of particular importance in practice. Note that,

[1]Let x_1 and x_2 be two d-dimensional points. Their distance under the L_∞ norm is $\max_{i=1}^{d} |x_1[i] - x_2[i]|$, where $x_1[i]$ and $x_2[i]$ are the i-th coordinates of x_1 and x_2, respectively. The distance under the L_2 norm is simply the length of the line segment connecting x_1 and x_2.

under L_∞, $\odot(x, \varepsilon_q)$ is a square whose centroid falls at x and has a side length $2\varepsilon_q$.

Since all the queries are "probability thresholding", we will omit this phrase in referring to their names. Furthermore, when the query type is clear, we also use the term *qualification probability* for $Pr_{range}(o, r_q)$ or $Pr_{fuzzy}(o, q, \varepsilon_q)$. As mentioned in Section 1.2, we adopt a *filter-refinement framework* in query processing on uncertain data. Specifically, the filter step first retrieves a small *candidate set*, after which the refinement phase computes the exact qualification probabilities of all the objects in the set, and then produces the query result. Hence, the candidate set should be a superset of the final result, that is, any object excluded from the set must violate the spatial predicates with an adequately high chance.

As analyzed in Section 5.3, it is usually expensive to compute the qualification probabilities (typically, Eqs. (1) and (3) can be evaluated only numerically). Therefore, the first objective of the filter step is to *prune as many nonqualifying objects as possible, using computationally economical operations*, which involve only checking the topological relationships between two rectangles (i.e., whether they intersect, contain each other, or are disjoint). Meanwhile, the filter step also achieves another equally important objective: *validating as many qualifying objects as possible*, again by analyzing only rectangles' topological relationships. We need to invoke the expensive process of qualification probability evaluation, only if an object can be neither pruned nor validated.

The above discussion implies that, towards fast query processing, a crucial task is to *efficiently derive tight lower and upper bounds for the qualification probability* of an object. Specifically, the lower bound is for validating (i.e., an object is guaranteed to be a result, if the lower bound is at least the query probability threshold t_q), whereas the upper bound is for pruning. Next, we clarify how to obtain these bounds for each type of queries. Table I lists the symbols frequently used in our presentation (some symbols have not appeared so far, but will be introduced later).

3. NONFUZZY RANGE SEARCH

In this section, we will discuss the fundamental properties of probabilistically constrained rectangles (PCR), particularly, how they can be applied to assist pruning and validating for nonfuzzy range queries. Unless specifically stated, all the queries have axis-parallel rectangular search regions (queries with general shapes of search areas are the topic of Section 3.5).

3.1 Intuition behind Probabilistically Constrained Rectangles

A PCR of an object o depends on a parameter $c \in [0, 0.5]$, and hence, is represented as $o.pcr(c)$. It is a d-dimensional rectangle, obtained by pushing, respectively, each face of $o.mbr$ inward, until the appearance probability of o in the area swept by the face equals c. Figure 3(a) illustrates the construction of a 2D $o.pcr(c)$, where the polygon represents the uncertainty region $o.ur$ of o, and the dashed rectangle is the MBR of o, denoted as $o.mbr$. The $o.pcr(c)$, which is the grey area, is decided by 4 lines $l_{[1]+}$, $l_{[1]-}$, $l_{[2]+}$, and $l_{[2]-}$. Line $l_{[1]+}$ has

Table I. Frequently Used Symbols

Symbol	Description	Section of Definition
d	the workspace dimensionality	2
$o.ur$, $o.pdf$ $o.mbr$	the uncertainty region and pdf of o, the MBR of $o.ur$	2 3.1
$o.pcr(c)$, $[o.pcr_{[i]-}(c), o.pcr_{[i]+}(c)]$	a probabilistically constrained rectangle (PCR) of o, the projection of the PCR on the i-th dimension	3.2 3.2
r_q, t_q	the search region and probability threshold of a nonfuzzy query	2
r_q, ε_q t_q	the search region, distance threshold and probability threshold of a fuzzy query	2
$Pr_{range}(o, r)$	the probability of o appearing in a region r	2
$Pr_{fuzzy}(o, q, \varepsilon_q)$	the probability for o and q to be within distance ε_q	2
$C_1, \ldots, C_m$	the values of a U-catalog (in ascending order)	3.3
$UB_{range}(o, r)$, $LB_{range}(o, r)$	an upper and a lower bound of $Pr_{range}(o, r)$	3.4
$UB_{fuzzy}(r, o, \varepsilon)$, $LB_{fuzzy}(r, o, \varepsilon)$	an upper and a lower bound of $Pr_{fuzzy}(o, q, \varepsilon)$	4.2
$e.mbr(c)$	the MBR of the $o.pcr(c)$ of all the objects o in the subtree of an intermediate U-tree entry e	5.1
$e.sl(c)$	the minimum projection length on any dimension of $o.pcr(c)$ of any object o in the subtree of e	5.1

the property that, the appearance probability of o on the right of $l_{[1]+}$ (i.e., the hatched area) is c. Similarly, $l_{[1]-}$ is obtained in such a way that the appearance likelihood of o on the left of $l_{[1]-}$ equals c (it follows that the probability that o lies between $l_{[1]-}$ and $l_{[1]+}$ is $1 - 2c$). Lines $l_{[2]+}$ and $l_{[2]-}$ are obtained in the same way, except that they horizontally partition $o.ur$.

PCRs can be used to prune or validate an object, without computing its accurate qualification probability. Let us assume that the grey box in Figure 3(a) is the $o.pcr(0.1)$ of o. Figure 3(b) shows the same PCR and $o.mbr$ again, together with the search region r_{q_1} of a nonfuzzy range query q_1 whose probability threshold t_{q_1} equals 0.9. As r_{q_1} does not fully contain $o.pcr(0.1)$, we can immediately assert that o cannot qualify q_1. Indeed, since o falls in the hatched region with probability 0.1, the appearance probability of o in r_{q_1} must be smaller than $1 - 0.1 = 0.9$. Figure 3(c) illustrates pruning the same object with respect to another query q_2 having $t_{q_2} = 0.1$. This time, o is disqualified because r_{q_2} does not intersect $o.pcr(0.1)$ (the pruning conditions are different for q_1 and q_2). In fact, since r_{q_2} lies entirely on the right of $l_{[1]+}$, the appearance probability of o in r_{q_2} is definitely smaller than 0.1.

The second row of Figure 3 presents three situations where o can be validated using $o.pcr(0.1)$, with respect to queries q_3, q_4, q_5 having probability thresholds $t_{q_3} = 0.9$, $t_{q_4} = 0.8$, and $t_{q_5} = 0.1$, respectively. In Figure 3(d) (or Figure 3(f)), o must satisfy q_3 (or q_5) due to the fact that r_{q_3} (or r_{q_5}) fully covers the part of $o.mbr$ on the right (or left) of $l_{[1]-}$, which implies that the appearance probability of o in the query region must be at least $1 - 0.1 = 0.9$ (or 0.1), where 0.1 is the likelihood for o to fall in the hatched area. Similarly, in Figure 3(e), o definitely qualifies q_4, since r_{q_4} contains the portion of $o.mbr$ between $l_{[1]-}$ and $l_{[1]+}$, where the appearance probability of o equals $1 - 0.1 - 0.1 = 0.8$.

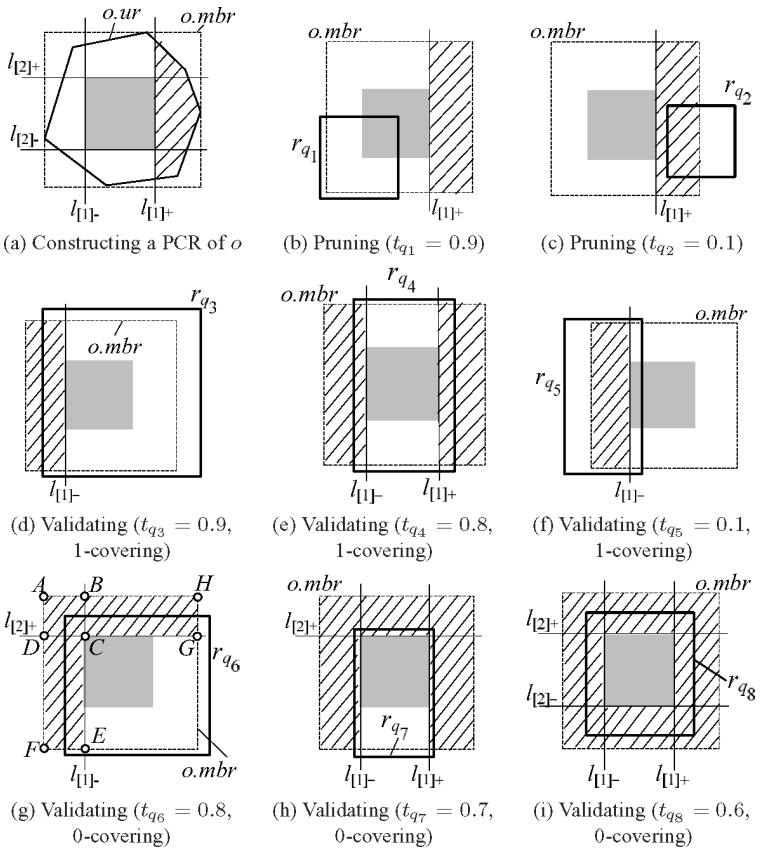


Fig. 3. Pruning/validating with a 2D probabilistically constrained rectangle.

The queries in Figures 3(d)–3(f) share a common property: the projection of the search region contains that of $o.mbr$ along one (specifically, the vertical) dimension. Accordingly, we say that those queries *1-cover* $o.mbr$. In fact, validation is also possible, even if a query *0-covers* $o.mbr$, namely, the projection of the query area does not contain that of $o.mbr$ on any dimension. Next, we illustrate this using the third row of Figure 3, where the queries q_6, q_7, q_8 have probability thresholds $t_{q_6} = 0.8$, $t_{q_7} = 0.7$, and $t_{q_8} = 0.6$, respectively.

In Figure 3(g), o is guaranteed to qualify q_6, since r_{q_6} covers entirely the part of $o.mbr$ outside the hatched area. Observe that the appearance probability of o in the hatched area is *at most* 0.2. To explain this, we decompose the area into three rectangles $ABCD$, $DCEF$, $BCGH$, and denote the probabilities for o to lie in them as ρ_{ABCD}, ρ_{DCEF}, and ρ_{BCGH}, respectively. By the definition of $l_{[1]-}$, we know that $\rho_{ABCD} + \rho_{DCEF} = 0.1$, whereas, by $l_{[2]+}$, we have $\rho_{ABCD} + \rho_{BCGH} = 0.1$. Since ρ_{ABCD}, ρ_{DCEF}, and ρ_{BCGH} are nonnegative, it holds that $\rho_{ABCD} + \rho_{DCEF} + \rho_{BCGH} \leq 0.2$. This, in turn, indicates that o falls in r_{q_6} with probability at least 0.8 ($= t_{q_6}$). With similar reasoning, it is not hard to verify that, in Figure 3(h) (Figure 3(i)), the appearance probability of o in the hatched area is at most 0.3 (0.4), meaning that o definitely satisfies q_7 (q_8).

3.2 Formalization of PCRs

We are ready to formalize PCRs and the related pruning/validating rules.

Definition 3. Given a value $c \in [0, 0.5]$, the *probabilistically constrained rectangle* $o.pcr(c)$ of an uncertain object o is a d-dimensional rectangle, representable by a $2d$-dimensional vector $\{o.pcr_{[1]-}(c), o.pcr_{[1]+}(c), \ldots, o.pcr_{[d]-}(c), o.pcr_{[d]+}(c)\}$, where $[o.pcr_{[i]-}(c), o.pcr_{[i]+}(c)]$ is the projection of $o.pcr(c)$ on the ith dimension. In particular, $o.pcr_{[i]-}(c)$ and $o.pcr_{[i]+}(c)$ satisfy

$$\int_{x[i] \le o.pcr_{[i]-}(c)} o.pdf(x)\,dx = \int_{x[i] \ge o.pcr_{[i]+}(c)} o.pdf(x)\,dx = c, \tag{4}$$

where $x[i]$ denotes the ith coordinate of a d-dimensional point x.

$o.pcr(c)$ can be computed by considering each individual dimension in turn. We illustrate this using Figure 3(a) 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* $o.cdf(y)$ 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 y. Thus, $l_{[1]-}$ can be decided by solving y from the equation $o.cdf(y) = c$, and similarly, $l_{[1]+}$ from $o.cdf(y) = 1 - c$. When $o.pdf(x)$ is regular (e.g., uniform or Gaussian), given a constant c, the y satisfying $o.cdf(y) = c$ can be obtained precisely. In any case, there is a standard sampling approach to evaluate the equation numerically. Specifically, let us randomly generate s points $x_1, x_2, \ldots, x_s$ in $o.mbr$, sorted in ascending order of their x-coordinates. Then, we slowly move a vertical line l from the left edge of $o.mbr$ to its right edge. Every time l crosses a sample, we compute a value $v = \frac{vol}{s'} \cdot \sum_{i=1}^{s'} o.pdf(x_i)$, where s' is the number of samples crossed so far, and vol the area of the part of $o.mbr$ on the left of l. As soon as v exceeds c, and the solution of $o.cdf(y) = c$ is taken as the x-coordinate of the last sample crossed by l.

In general, for any c and c' satisfying $0 \le c < c' \le 0.5$, $o.pcr(c)$ always contains $o.pcr(c')$. Specially, $o.pcr(0)$ is the MBR of the uncertainty region of o, and $o.pcr(0.5)$ degenerates into a point. The next theorem summarizes the pruning rules.

THEOREM 1. *Given a nonfuzzy range query with search region r_q and probability threshold t_q, the following holds:*

(1) *for $t_q > 0.5$, o can be pruned, if r_q does not contain $o.pcr(1 - t_q)$;*
(2) *for $t_q \le 0.5$, o can be pruned, if r_q does not intersect $o.pcr(t_q)$.*

PROOF. The proofs of all lemmas and theorems can be found in the appendix. □

In general, given two d-dimensional rectangles r and r', we say that r *l-covers* r' ($0 \le l \le d$), if there exist l dimensions along which the projection of r encloses that of r'. As a special case, if r d-covers r', then r contains the entire r'. Based on the notion of l-covering, we present the validating rules as follows.

THEOREM 2. *Given a nonfuzzy range query q with search region r_q and probability threshold t_q, consider an object o whose $o.pcr(0)$ is l-covered by r_q ($1 \leq l \leq d$). If $l = d$, o falls in r_q with 100% likelihood. Otherwise, without loss of generality, assume that the projection of $o.pcr(0)$ is not covered by that of r_q on dimensions 1, 2, ..., $d - l$. Then:*

(1) *for any t_q, o definitely satisfies q, if there exist $2(d - l)$ values c_i, c'_i ($1 \leq i \leq d - l$) such that $t_q \leq 1 - \sum_{i=1}^{d-l}(c_i + c'_i)$, and on the ith dimension ($1 \leq i \leq d - l$), $[r_{q[i]-}, r_{q[i]+}]$ contains $[o.pcr_{[i]-}(c_i), o.pcr_{[i]+}(c'_i)]$;*

(2) *for $l = d - 1$ and $t_q \leq 0.5$, o definitely satisfies q, if there exist values c_1, c'_1 with $c_1 \leq c'_1$ such that $t_q \leq c'_1 - c_1$, and $[r_{q[1]-}, r_{q[1]+}]$ contains either $[o.pcr_{[1]-}(c_1), o.pcr_{[1]-}(c'_1)]$ or $[o.pcr_{[1]+}(c'_1), o.pcr_{[1]+}(c_1)]$.*

Theorem 2 generalizes the reasoning behind the validation performed in Figure 3. To illustrate Rule (1), let us review Figure 3(d), where r_{q_3} 1-covers $o.pcr(0)$ ($= o.mbr$). It is the horizontal dimension, denoted as dimension 1, on which $o.pcr(0)$ is not covered. There exist $c_1 = 0$ and $c'_1 = 0.1$, such that $[r_{q[1]-}, r_{q[1]+}]$ contains $[o.pcr_{[1]-}(c_1), o.pcr_{[1]+}(c'_1)]$. Since $t_{q_3} = 0.9 \leq 1 - (c_1 + c'_1) = 0.9$, by Theorem 2, o is guaranteed to satisfy q_3.

As another example, consider Figure 3(h). Here, r_{q_7} 0-covers $o.pcr(0)$. We can find $c_1 = c'_1 = 0.1$ for the horizontal axis, and $c_2 = 0$, $c'_2 = 0.1$ for the vertical axis (dimension 2), which fulfill the following condition: $[r_{q[i]-}, r_{q[i]+}]$ encloses $[o.pcr_{[i]-}(c_i), o.pcr_{[i]+}(c'_i)]$ for both $i = 1$ and 2. As $t_{q_7} = 0.7 \leq 1 - \sum_{i=1}^{2}(c_i + c'_i) = 0.7$, we can assert that o satisfies q_7 according to Rule (1).

Rule (2) can be applied only if the query region $(d - 1)$-covers $o.mbr$. For instance, in Figure 3(f), r_{q_5} does not cover $o.mbr$ only on the horizontal dimension. We may set c_1, c'_1 to 0 and 0.1, respectively, such that $[r_{q[1]-}, r_{q[1]+}]$ includes $[o.pcr_{[1]-}(c_1), o.pcr_{[1]-}(c'_1)]$. Since $t_{q_5} \leq c'_1 - c_1 = 0.1$, by Rule 2, o definitely qualifies r_{q_5}.

3.3 Heuristics with a Finite Number of PCRs

The effectiveness of Theorems 1 and 2 is maximized if we could precompute the PCRs of an object for all $c \in [0, 0.5]$. Since this is clearly impossible, for each object o, we obtain its $o.pcr(c)$ only at some *predetermined values of* c, which are common to all objects and constitute the *U-catalog*.[2] We denote the values of the U-catalog as (in ascending order) C_1, C_2, ..., C_m, where m is the size of the catalog. In particular, C_1 is fixed to 0, that is, $o.mbr = o.pcr(C_1)$ is always captured.

A problem, however, arises. Given an arbitrary query probability threshold t_q, the corresponding PCR needed for pruning/validating may not exist. For instance, in Figure 3(b), as mentioned earlier, disqualifying object o for query q_1 requires $o.pcr(0.1)$. Thus, the pruning cannot be performed if 0.1 is not in the U-catalog.

We solve this problem by applying the heuristics of Section 3.2 in a conservative way. Assuming a U-catalog with $m = 2$ values $\{C_1 = 0, C_2 = 0.25\}$, Figure 4

[2]'U' here reminds of the fact that the catalog is created for uncertain data.

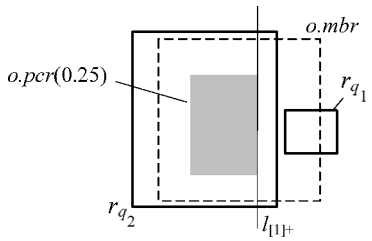


Fig. 4. Pruning and validating using only $o.pcr(0.25)$ and $o.pcr(0)$.

shows an example where the dashed rectangle is $o.mbr$, and the grey box is $o.pcr(C_2)$. Rectangle r_{q_1} is the search region of q_1 whose probability threshold t_{q_1} equals 0.8. Since $o.pcr(0.25)$ is not contained in r_{q_1}, by Rule (1) of Theorem 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.

Let us consider another query q_2 with $t_{q_2}=0.7$ and a search region r_{q_2} shown in Figure 4. We can validate o for q_2 by examining only $o.mbr$ and $o.pcr(0.25)$. In fact, since r_{q_2} completely covers the part of $o.mbr$ on the left of line $l_{[1]+}$, we can assert (by Rule (1) of Theorem 2) that o appears in r_{q_2} with a probability at least 0.75, which is larger than t_{q_2}.

In general, given a finite number of PCRs, we can still prune an object o, if those PCRs allow us to verify that o cannot appear in the query region r_q even with a probability *lower than or equal to* the probability threshold t_q. Based on this reasoning, the next theorem presents the adapted version of the heuristics in Theorem 1:

THEOREM 3. *Given a nonfuzzy range query with search region r_q and probability threshold t_q, the following holds:*

(1) *for $t_q > 1-C_m$ (recall that C_m is the largest value in the U-catalog), o can be pruned, if r_q does not contain $o.pcr(c_{\vdash})$, where $c_{\vdash}$ is the smallest value in the U-catalog that is at least $1-t_q$;*
(2) *for any t_q, o can be pruned, if r_q does not intersect $o.pcr(c_{\dashv})$, where $c_{\dashv}$ is the largest value in the catalog that is at most t_q.*

Similarly, using only m PCRs, we may still validate an object o, as long as we can infer that o falls in r_q with a chance higher than or equal to t_q. Actually, in this case, validating can still be performed using Theorem 2, except that all the c_i, c_i' $(1 \le i \le d-l)$ should be taken from the U-catalog, and Rule 2 is applied only if $t_q \le C_m$ (as opposed to $t_q \le 0.5$ in Theorem 2).

3.4 Computing the Lower and Upper Bounds of Qualification Probability

Application of Theorem 3 is trivial, because both $c_{\vdash}$ and $c_{\dashv}$ are well-defined, and simple to identify. The utility of Theorem 2, however, is less straightforward, because the appropriate values of c_i, c_i' $(1 \le i \le d-l)$ for successful validation are not immediately clear. A naive method (of trying all the possibilities) may entail expensive overhead, because each c_i or c_i' can be any value in the U-catalog, resulting in totally $m^{2(d-l)}$ possibilities.

Algorithm 1: `Nonfuzzy-Range-Quali-Prob-Bounds`(o, r_q)

/* o is an uncertain object and r_q a rectangle. The algorithm returns a lower bound $LB_{range}(o, r_q)$ and an upper bound $UB_{range}(o, r_q)$ of $Pr_{range}(o, r_q)$ (Equation 1). Both bounds tight as far as Theorems 2 and 3 are concerned (see Lemma 2). */

1. if r_q fully covers $o.mbr$ then $LB_{range}(o, r_q) = UB_{range}(o, r_q) = 1$
2. else if r_q is disjoint with $o.mbr$ then $LB_{range}(o, r_q) = UB_{range}(o, r_q) = 0$
3. else

/* Lines 4-9 decide $UB_{range}(o, r_q)$ */

4. $c_{\leq}$ = the smallest value c in the U-catalog such that r_q is disjoint with $o.pcr(c)$
5. if $c_{\leq}$ exists then $UB_{range}(o, r_q) = c_{\leq} - \delta$, where δ is an infinitely small positive
6. else
7. $c_{\geq}$ = the largest value c in the U-catalog such that r_q does not fully cover $o.pcr(c)$ /* $c_{\geq}$ always exists */
8. $UB_{range}(o, r_q) = 1 - c_{\geq} - \delta$, where δ is as defined in Line 5

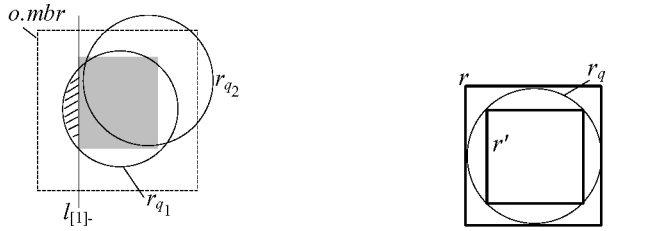
/* The following lines decide $LB_{range}(o, r_q)$ */

9. $LB_{range}(o, r_q) = 0$
10. l = the number of dimensions on which the projection of r_q contains that of $o.mbr$; without loss of generality, let these dimensions be 1, 2, ..., l
11. if r_q contains $o.pcr(C_m)$
12. for i = 1 to $d - l$ /* consider, in turn, each "uncovered" dimension of $o.mbr$ */
13. c_i = the smallest value c in the U-catalog such that $r_{q[i]-} \leq o.pcr_{[i]-}(c)$
14. c'_i = the smallest value c in the U-catalog such that $r_{q[i]+} \geq o.pcr_{[i]+}(c)$ /* c_i and c'_i always exist */
15. $LB_{range}(o, r_q) = \max\{LB_{range}(o, r_q), 1 - \sum_{i=1}^{d-l}(c_i + c'_i)\}$
16. else if $l = d - 1$
17. c_1 = the smallest value c in the U-catalog such that $r_{q[1]-} \leq o.pcr_{[1]-}(c)$ or $r_{q[1]+} \geq o.pcr_{[1]+}(c)$; /* c_1 always exists */
18. c'_1 = the largest value c in the U-catalog such that $r_{q[1]+} \geq o.pcr_{[1]-}(c)$ or $r_{q[1]-} \leq o.pcr_{[1]+}(c)$
19. if c'_1 exists then $LB_{range}(o, r_q) = \max\{LB_{range}(o, r_q), c'_1 - c_1\}$

Fig. 5. Finding a lower and an upper bound of an object's qualification probability.

Figure 5 presents an algorithm that allows us to determine whether an object can be pruned/validated in $O(m + d \log m)$ time. Specifically, the algorithm returns an upper bound $UB_{range}(o, r_q)$ and a lower bound $LB_{range}(o, r_q)$ for the actual qualification probability $Pr_{range}(o, r_q)$ of an object o. Given these bounds, we can prune o if $t_q > UB_{range}(o, r_q)$, or validate o if $t_q \leq LB_{range}(o, r_q)$. Numerical calculation of $Pr_{range}(o, r_q)$ is necessary if and only if $t_q \in (LB_{range}(o, r_q), UB_{range}(o, r_q)]$.

LEMMA 1. *Let $UB_{range}(o, r_q)$ and $LB_{range}(o, r_q)$ be the values produced by Algorithm* 1 *(Figure* 5*). For any query probability threshold $t_q > UB_{range}(o, r_q)$, Theorem* 3 *always disqualifies o from appearing in r_q with a probability at least t_q. For any $t_q \leq LB_{range}(o, r_q)$, Theorem* 2 *always validates o as an object appearing in r_q with a probability at least t_q.*



(a) Pruning possible for q_2, but not for q_1 ($t_{q_1} = t_{q_2} = 0.9$) (b) Reduction to rectangles

Fig. 6. Rationale of supporting circular search regions.

Having shown that Algorithm 1 is correct, we proceed with another lemma that confirms the tightness of the resulting lower and upper bounds, as far as our pruning and validating heuristics are concerned.

LEMMA 2. *Let $UB_{range}(o, r_q)$ and $LB_{range}(o, r_q)$ be the values produced by Algorithm* 1*. Theorem* 3 *cannot prune o, if the query probability threshold t_q does not exceed $UB_{range}(o, r_q)$. Similarly, Theorem* 2 *cannot validate o, if t_q is higher than $LB_{range}(o, r_q)$.*

It is not hard to verify that the computation time of Algorithm 1 is $O(m + d \log m)$. In particular, except Lines 12–14, every other line incurs either $O(1)$ or $O(m)$ cost. Each of Lines 13 and 14 can be implemented in $O(\log m)$ time, by performing a binary search, so that the for-loop initiated at Line 12 entails totally $O((d - l) \cdot \log m) = O(d \log m)$ overhead.

3.5 Supporting Search Regions of Arbitrary Shapes

The above sections assume axis-parallel rectangular regions. The pruning/validating heuristics presented so far, unfortunately, do not apply to queries with arbitrary shapes of search areas. For example, consider Figure 6(a), where the dashed and grey rectangles represent $o.mbr$ and $o.pcr(0.1)$, respectively. Circle r_{q_1} is the search region of a query q_1 with probability threshold $t_{q_1} = 0.9$. Notice that r_{q_1} does not fully cover $o.pcr(0.1)$, and therefore, Theorem 1 would determine o as nonqualifying. This decision, however, may be wrong, because o could have a probability 0.1 falling in the hatched area, and probability 0.8 in the unhatched portion of r_{q_1}.

Interestingly, we can correctly prune/validate an object for a query with any search region r_q, utilizing directly our solutions for axis-parallel rectangles. For this purpose, we resort to a rectangle r that contains r_q, and another rectangle r' that is fully enclosed in r_q (Figure 6(b) demonstrates r and r' for a circular r_q). Then, if an object o appears in r with a probability less than t_q (the query probability threshold), then o can be safely pruned. Likewise, if o falls in r' with a chance at least t_q, o can be immediately validated.

In general, for any search area r_q and uncertain object o, we can use r and r' (obtained as described earlier) to calculate a range $[LB_{range}(o, r_q), UB_{range}(o, r_q)]$, which is guaranteed to contain the actual probability $Pr_{range}(o, r_q)$ of o appearing in r_q. Specifically, $UB_{range}(o, r_q)$ equals the

upper bound returned by Algorithm 1 (Figure 5) with respect to r, whereas $LB_{range}(o, r_q)$ is the lower bound produced by the algorithm for r'. The precise $Pr_{range}(o, r_q)$ needs to be derived, if and only if t_q falls in $(LB_{range}(o, r_q), UB_{range}(o, r_q)]$.

It remains to clarify the computation of r and r'. Although any r (r') containing (contained in) the search region r_q guarantees the correctness of the query result, r (r') should be as small (large) as possible, to maximize the effectiveness of the above approach. Obviously, the smallest r is the minimum bounding rectangle of r_q. On the other hand, when r_q is a polygon, the r' with the largest area can be found using the algorithm in Daniels et al. [1997].

The above approach is especially useful when r_q is a rather irregular region, such that it is difficult to test the topological relationships between r_q and other geometric objects (we will use the approach to tackle fuzzy search in Section 4). However, approximating r_q with r and r' may be overly conservative, resulting in an $[LB_{range}(o, r_q), UB_{range}(o, r_q)]$ that may be unnecessarily wide. In fact, when r_q has only limited complexity, we can achieve better pruning and validating effects by extending the analysis of the previous sections.

Let us examine Figure 6(a) again, this time focusing on r_{q_2}, the search area of query q_2 with probability threshold $t_{q_2} = 0.9$ (the semantics of the dashed and grey rectangles are the same as mentioned before). Observe that r_{q_2} lies completely on the right of (and does not touch) line $l_{[1]-}$. Since o appears on the left of $l_{[1]-}$ with probability 0.1, it falls out of r_{q_2} with at least 90% likelihood; hence, o can be eliminated. Note that pruning o in this case essentially follows the rationale illustrated in Figure 3(b). Indeed, even for general search areas, the reasoning discussed in Section 3.1 of using PCRs for pruning/invalidating is still applicable. Based on such reasoning, we present the generic versions of Theorems 3 and 2.

THEOREM 4. *Given a nonfuzzy range query with search region r_q and probability threshold t_q, the following holds:*

(1) *for $t_q > 1 - C_m$, o can be pruned, if r_q falls completely on the right (or left) of the plane containing the left (or right) face of $o.pcr(c_{\vdash})$ on any dimension, where $c_{\vdash}$ is the smallest value in the U-catalog that is at least $1 - t_q$;*
(2) *for any t_q, o can be pruned, if r_q falls completely on the left (or right) of the plane containing the left (or right) face of $o.pcr(c_{\dashv})$, where $c_{\dashv}$ is the largest value in the U-catalog that is at most t_q.*

THEOREM 5. *Given a nonfuzzy range query q with search region r_q and probability threshold t_q, o is guaranteed to satisfy r_q in either of the following situations*

(1) *we can find $2d$ values c_i, c_i' ($1 \le i \le d$) in the U-catalog, such that $t_q \le 1 - \sum_{i=1}^{d}(c_i + c_i')$, and r_q completely covers a d-dimensional rectangle r, whose projection on the i-th dimension ($1 \le i \le d$) is $[o.pcr_{[i]-}(c_i), o.pcr_{[i]+}(c_i')]$;*
(2) *we can find a dimension $i \in [1, d]$, and 2 values c, c' in the U-catalog, such that $t_q \le c' - c$, and r_q completely covers a d-dimensional rectangle r, whose projection on the i-th dimension is $[o.pcr_{[i]-}(c), o.pcr_{[i]-}(c')]$ or*

$[o.pcr_{[i]+}(c'), o.pcr_{[i]+}(c)]$*, and* r *shares the same projection as* $o.mbr$ *on the other dimensions.*

Theorems 4 and 5 can be employed, as long as it is feasible to check (i) whether r_q falls completely on one side of an axis-parallel plane, and (ii) whether r_q contains a d-dimensional rectangle. The only problem, however, is that there does not exist an efficient algorithm for finding the set of $2d$ values in Rule (1) of Theorem 5, in order to perform successful validation (recall that, if r_q is axis-parallel, we can use Algorithm 1 to validate an object o in $O(m + d \log m)$ time, whenever o can be validated by Theorem 2). A brute-force method (of attempting all possible c_i, c_i' for $i \in [1, d]$) may not work, because each c_i or c_i' can be any of the m values in the U-catalog, rendering m^{2d} possibilities. In practice, a useful trick for alleviating the problem is to restrict the number of these $2d$ values that are not zero to a small value α. In this case, there are only $\binom{2d}{\alpha} \cdot m^{\alpha}$ possibilities. For $\alpha = 1$ or 2, it is computationally tractable to examine all of them. In any case, the correctness of the query result is never compromised, because if Theorem 5 cannot validate o, the actual qualification probability of o will be calculated.

It is worth noting that the above technique may not be general enough to support any nonrectangular search regions efficiently, especially if the region has a complex shape. Processing such queries requires dedicated solutions beyond the scope of this article.

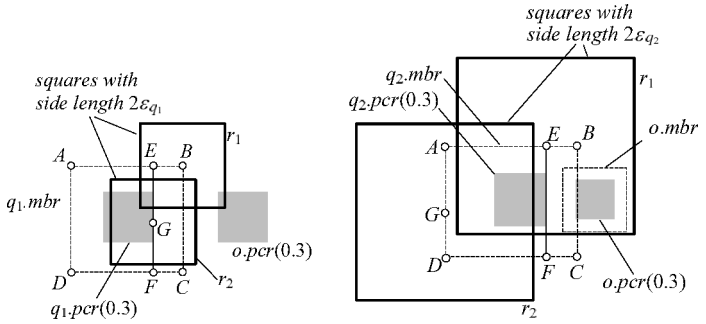
4. FUZZY RANGE SEARCH

We have shown in Section 3 that PCRs enable efficient pruning/validating for nonfuzzy range search. In the sequel, we will demonstrate that PCRs are also useful for fuzzy queries. As formulated in Definition 2, given an uncertain object q, a distance value ε_q, and a probability threshold t_q, such a query finds all the objects o in a dataset satisfying $Pr_{fuzzy}(o, q, \varepsilon_q) \geq t_q$, where $Pr_{fuzzy}(o, q, \varepsilon_q)$ is given in Eq. (3). Section 4.1 provides the rationale behind our heuristics, which are formally presented in Section 4.2.

4.1 Intuition of Pruning and Validating

Evaluation of Eq. (3) is usually costly, especially if q, o, or both have irregular uncertainty regions and pdfs. Our objective is to prune or validate o without going through the expensive evaluation. Next, we explain the underlying rationale, assuming that the distance metric employed is the L_∞ norm; nevertheless, our discussion can be extended to the L_2 norm in a straightforward manner.

Let us consider a query q_1 with probability threshold $t_{q_1} = 0.5$. Assume that we have already calculated $q_1.pcr(0.3)$ and $o.pcr(0.3)$, which are the left and right grey boxes in Figure 7(a), respectively. The dashed rectangle $ABCD$ is the MBR (denoted as $q_1.mbr$) of the uncertainty region of q_1. The parameter ε_{q_1} of q_1 equals half of the side length of square r_1 or r_2. By examining only $q_1.mbr$ and the two PCRs, we can assert that $Pr_{fuzzy}(o, q_1, \varepsilon_{q_1})$ is at most 0.42, and hence, o can be safely eliminated. To explain this, we need to cut $ABCD$ into two disjoint



(a) Pruning o ($t_q = 0.5$) (b) Validating o ($t_q = 0.3$)

Fig. 7. Pruning/validating with PCRs for fuzzy queries (under the L_∞ norm).

rectangles *EBCF* and *AEFD*, and then rewrite Eq. 3 as:

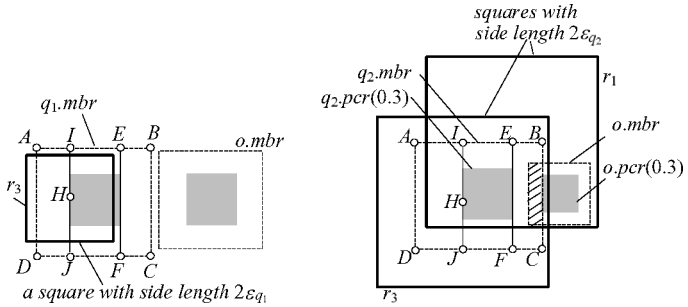
$$Pr_{fuzzy}(o, q_1, \varepsilon_{q_1}) = \int_{x \in EBCF} q_1.pdf(x) \cdot Pr_{range}(o, \odot(x, \varepsilon_{q_1}))dx + \int_{x \in AEFD} q_1.pdf(x) \cdot Pr_{range}(o, \odot(x, \varepsilon_{q_1}))dx, \quad (5)$$

where $\odot(x, \varepsilon_{q_1})$ represents a square that centers at point x, and has a side length of $2\varepsilon_{q_1}$. Observe that, for any $x \in EBCF$, $Pr_{range}(o, \odot(x, \varepsilon_{q_1}))$ must be bounded by 0.7, due to the fact that $\odot(x, \varepsilon_{q_1})$ does not fully cover $o.pcr(0.3)$. For instance, rectangle r_1 illustrates the $\odot(x, \varepsilon_{q_1})$ when x lies at point B; by Rule 1 of Theorem 1, o appears in r_1 with a probability at most 0.7. On the other hand, for any $x \in AEFD$, $Pr_{range}(o, \odot(x, \varepsilon_{q_1}))$ never exceeds 0.3, because $\odot(x, \varepsilon_{q_1})$ does not intersect $o.pcr(0.3)$. As an example, rectangle r_2 shows the $\odot(x, \varepsilon_{q_1})$ for $x = G$; according to Rule (2) of Theorem 1, o falls in r_2 with no more than 0.3 probability. As a result:

$$Pr_{fuzzy}(o, q_1, \varepsilon_{q_1}) \leq 0.7 \int_{x \in EBCF} q_1.pdf(x)dx + 0.3 \int_{x \in AEFD} q_1.pdf(x)\,dx = 0.7 \times 0.3 + 0.3 \times 0.7 = 0.42. \quad (6)$$

Let q_2 be another query with probability threshold $t_{q_2} = 0.3$. The left and right grey boxes in Figure 7(b) demonstrate $q_2.pcr(0.3)$ and $o.pcr(0.3)$, respectively, whereas the larger and smaller dashed rectangles capture $q_2.mbr$ and $o.mbr$, respectively. The parameter ε_{q_2} of q_2 equals half of the side length of square r_1 or r_2. Based on only the above information, we can claim that $Pr_{fuzzy}(o, q_2, \varepsilon_{q_2}) \geq 0.3$, and hence, o can be validated. To clarify this, we again divide $q_2.mbr$ into rectangles *EBCF* and *AEFD*, and scrutinize Eq. (5). Here, for any $x \in EBCF$, $Pr_{range}(o, \odot(x, \varepsilon_{q_2}))$ is a constant 1, because $\odot(x, \varepsilon_{q_2})$ necessarily contains $o.mbr$ (r_1 illustrates an example of $\odot(x, \varepsilon_{q_2})$ for $x = E$). However, when x distributes in *AEFD*, $Pr_{range}(o, \odot(x, \varepsilon_{q_2}))$ may drop to 0, as is the case for r_2, which is the $\odot(x, \varepsilon_{q_2})$ for $x = G$. It follows that

$$Pr_{fuzzy}(o, q_1, \varepsilon_{q_1}) \geq 1 \cdot \int_{x \in EBCF} q_1.pdf(x)dx + 0 \int_{x \in AEFD} q_1.pdf(x)dx = 1 \times 0.3 + 0 \times 0.7 = 0.3. \quad (7)$$



(a) Pruning o ($t_q = 0.4$) (b) Validating o ($t_q = 0.4$)

Fig. 8. Enhanced pruning/validating for fuzzy queries with more "slices" (under the L_∞ norm).

In the above examples, we "sliced" $q.mbr$ into two rectangles for pruning and validating. In fact, stronger pruning/validation effects are possible by performing the slicing more aggressively. Assume that, instead of 0.5, the query q_1 in Figure 7(a) has a lower $t_{q_1} = 0.4$; hence, o can no longer be disqualified as described with Inequality (6) (as $0.42 > t_{q_1}$). However, we can actually derive a tighter upper bound 0.33 of $Pr_{fuzzy}(o, q_1, \varepsilon_{q_1})$, and thus, still eliminate o. For this purpose, we should divide $q.mbr$ into three rectangles $EBCF$, $IEFJ$, and $AIJD$ as in Figure 8(a), which repeats the content of Figure 7(a), except for including $o.mbr$ (i.e., the right-dashed box). Accordingly: $Pr_{fuzzy}(o, q_1, \varepsilon_{q_1}) =$

$$\int_{x \in EBCF} q_1.pdf(x) \cdot Pr_{range}(o, \odot(x, \varepsilon_{q_1}))\, dx + \int_{x \in IEFJ} q_1.pdf(x) \cdot Pr_{range}(o, \odot(x, \varepsilon_{q_1}))\, dx + \int_{x \in AIJD} q_1.pdf(x) \cdot Pr_{range}(o, \odot(x, \varepsilon_{q_1}))\, dx. \quad (8)$$

As analyzed earlier with Figure 7(a), for any point $x \in EBCF$, $Pr_{range}(o, \odot(x, \varepsilon_{q_1})) \leq 0.7$, whereas, for any point $x \in IEFJ \subset ABCD$, $Pr_{range}(o, \odot(x, \varepsilon_{q_1})) \leq 0.3$. Furthermore, notice that, given any point $x \in AIJD$, $Pr_{fuzzy}(o, q_1, \varepsilon_{q_1})$ is always 0, because $\odot(x, \varepsilon_{q_1})$ is disjoint with $o.mbr$. For instance, rectangle r_3 is the $\odot(x, \varepsilon_{q_1})$ when x lies at H; evidently, it is impossible for o to appear in r_3. Therefore, Eq. $8 \geq$

$$0.7 \int_{x \in EBCF} q_1.pdf(x)\, dx + 0.3 \int_{x \in IEFJ} q_1.pdf(x) dx + 0 \int_{x \in AIJD} q_1.pdf(x)\, dx = 0.7 \times 0.3 + 0.3 \times 0.4 + 0 \times 0.3 = 0.33. \quad (9)$$

Similarly, suppose that the query q_2 in Figure 7(b) has a probability threshold $t_{q_2} = 0.4$, in which case o cannot be confirmed as a qualifying object with Inequality (7). Next, we will use Figure 8(b), where the grey and dashed rectangles have the same meaning as in Figure 7(b), to derive a new lower bound 0.42 of $Pr_{fuzzy}(o, q_2, \varepsilon_{q_2})$, which thus validates o.

Let us break $q_2.mbr$ into rectangles $EBCF$, $IEFJ$, and $AIJD$. Then, $Pr_{fuzzy}(o, q_2, \varepsilon_{q_2})$ can be represented as Eq. (8). Following the analysis that led to Inequality (7), we know that, for $x \in EBCF$, $Pr_{fuzzy}(o, q_2, \varepsilon_{q_2}) = 1$, and, for $x \in AIJD$, (obviously) $Pr_{fuzzy}(o, q_2, \varepsilon_{q_2}) \geq 0$. The new observation here is that, for $x \in IEFJ$, $Pr_{fuzzy}(o, q_2, \varepsilon_{q_2}) \geq 0.3$, since $\odot(x, \varepsilon_{q_2})$ always fully covers the

hatched area in Figure 8(b), which is the part of $o.mbr$ on the left of $o.pcr(0.3)$. Rectangle r_3 shows an example of $\odot(x, \varepsilon_{q_2})$ when $x = H$; according to Rule (2) of Theorem 2, o has a probability of at least 0.3 to lie in r_3. Therefore, Eq. (8) $\geq$

$$1 \int_{x \in EBCF} q_1.pdf(x)\,dx + 0.3 \int_{x \in IEFJ} q_1.pdf(x)\,dx + 0 \int_{x \in AIJD} q_1.pdf(x)\,dx$$
$$= 1 \times 0.3 + 0.3 \times 0.4 + 0 \times 0.3 = 0.42. \quad (10)$$

4.2 Formal Results

Before processing a fuzzy query q, we compute its $q.pcr(c)$ at m_q values of c in the range $[0, 0.5]$. We denote these values as QC_1, QC_2, ..., QC_{m_q}, respectively, sorted in ascending order. Similar to the smallest value C_1 in the U-catalog, QC_1 is fixed to 0, so that $q.pcr(c_1)$ always equals $q.mbr$. Note that m_q does not need to be the size m of the U-catalog (we will examine the influence of m_q in the experiments).

Given an object o, we aim at obtaining an upper bound $UB_{fuzzy}(o, q, \varepsilon_q)$ and a lower bound $LB_{fuzzy}(o, q, \varepsilon_q)$ of $Pr_{fuzzy}(o, q, \varepsilon_q)$, by studying only the PCRs of q and o (ε_q is the distance threshold of q). Then, o can be pruned if the query probability threshold t_q is larger than $UB_{fuzzy}(o, q, \varepsilon_q)$, or it can be validated if t_q does not exceed $LB_{fuzzy}(o, q, \varepsilon_q)$. Accurate evaluation of $Pr_{fuzzy}(o, q, \varepsilon_q)$ is performed only when both pruning and validating have failed.

We can reduce the computation of $UB_{fuzzy}(o, q, \varepsilon_q)$ and $LB_{fuzzy}(o, q, \varepsilon_q)$ to the derivation of upper and lower bounds of nonfuzzy range search qualification probabilities, which is solved by Algorithm 1 (Figure 5). In Section 4.2.1, we will first settle a related problem underlying the reduction, which is then clarified in Section 4.2.2.

4.2.1 *Bounds of $Pr_{range}(o, \odot(x, \varepsilon_q))$.* Given any axis-parallel rectangle r, in the sequel, we will develop two values $UB_{Pr}(r, o, \varepsilon_q)$ and $LB_{Pr}(r, o, \varepsilon_q)$ satisfying

$$LB_{Pr}(r, o, \varepsilon_q) \leq Pr_{range}(o, \odot(x, \varepsilon_q)) \leq UB_{Pr}(r, o, \varepsilon_q) \quad (11)$$

for any $x \in r$. To interpret $UB_{Pr}(r, o, \varepsilon_q)$ and $LB_{Pr}(r, o, \varepsilon_q)$ in a more intuitive manner, imagine that we examine all the circles $\odot(x, \varepsilon_q)$ whose centers x fall in r, and for every $\odot(x, \varepsilon_q)$, record the probability $Pr_{range}(o, \odot(x, \varepsilon_q))$ that o appears in $\odot(x, \varepsilon_q)$. Then, $UB_{Pr}(r, o, \varepsilon_q)$ is a value never smaller than any $Pr_{range}(o, \odot(x, \varepsilon_q))$, and similarly, $LB_{Pr}(r, o, \varepsilon_q)$ is never larger than any $Pr_{range}(o, \odot(x, \varepsilon_q))$. A tighter range $[LB_{Pr}(r, o, \varepsilon_q), UB_{Pr}(r, o, \varepsilon_q)]$ leads to stronger pruning/validating power, as will be clear shortly.

We use $\sqcup(r, \varepsilon_q)$ to denote the union of all the $\odot(x, \varepsilon_q)$ with $x \in r$, and $\sqcap(r, \varepsilon_q)$ for the intersection of those circles. Depending on the distance metric deployed, $\sqcup(r, \varepsilon_q)$ and $\sqcap(r, \varepsilon_q)$ have different shapes. Next, we will discuss this about the L_∞ and L_2 norms in 2D space, assuming that r has side lengths $sl_{[1]}$ and $sl_{[2]}$ on the horizontal and vertical dimensions, respectively. The discussion can be directly extended to higher dimensionalities.

Consider the grey rectangle r in Figure 9(a). Its $\sqcup(r, \varepsilon_q)$ under the L_∞ norm is rectangle $ABCD$, which shares the same centroid as r, and has side length $sl_{[i]} + 2\varepsilon_q$ on the ith dimension ($1 \leq i \leq 2$). Corner A of $\sqcup(r, \varepsilon_q)$, for instance, is

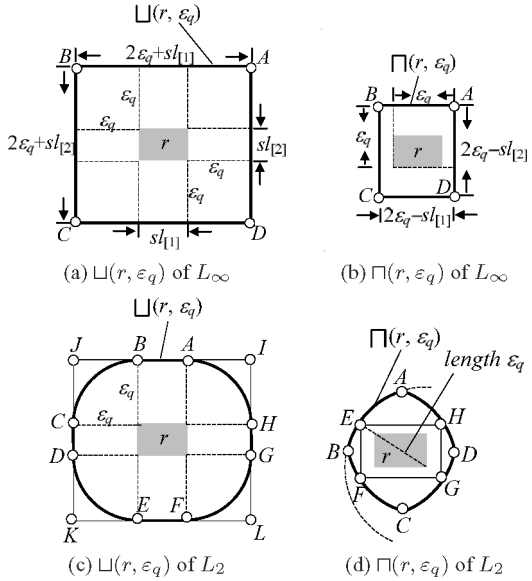


Fig. 9. $\sqcup(r, \varepsilon_q)$ and $\sqcap(r, \varepsilon_q)$ under the L_∞ and L_2 norms.

decided by the $\odot(x, \varepsilon_q)$ when x lies at the upper-right corner of r. Also focusing on L_∞, Figure 9(b) demonstrates $\sqcap(r, \varepsilon_q)$, which is also a rectangle $ABCD$ sharing the same centroid as r, but its side length is $2\varepsilon_q - sl_{[i]}$ along the ith dimension ($1 \leq i \leq 2$). In this case, corner A is determined by the $\odot(x, \varepsilon_q)$ when x is located at the bottom-left corner of r. In general, $\sqcap(r, \varepsilon_q)$ does not exist if $\varepsilon_q < \max_{i=1}^{d}(sl_{[i]}/2)$.

As in Figure 9(c), under the L_2 norm, $\sqcup(r, \varepsilon_q)$ has a more complex contour, which consists of line segments AB, CD, EF, GH, as well as arcs BC, DE, FG, and HA. For example, arc HA is formed by the $\odot(x, \varepsilon_q)$ when x is the upper-right corner of r; segment AB is created jointly by all the $\odot(x, \varepsilon_q)$ as x moves on the upper edge of r. We give the $\sqcap(r, \varepsilon_q)$ of L_2 in Figure 9(d), whose boundary has 4 arcs AB, BC, CD, and DA. Here, AB is determined by the $\odot(x, \varepsilon_q)$ when x is positioned at the bottom-right corner of r. As with its L_∞ counterpart, $\sqcap(r, \varepsilon_q)$ of L_2 is not always present; it exists only if $\varepsilon_q \geq \frac{1}{2}(\sum_{i=1}^{d} sl_{[i]}^2)^{1/2}$. In general, under any distance metric L, $\sqcup(r, \varepsilon_q)$ is essentially the Minkowski sum [Berg et al. 2000] of r and an L-sphere with radius ε_q.

LEMMA 3. *Let r be an axis-parallel rectangle. Given any point $x \in r$, we have (for any distance metric):*

$$Pr_{range}(o, \sqcap(r, \varepsilon_q)) \leq Pr_{range}(o, \odot(x, \varepsilon_q)) \leq Pr_{range}(o, \sqcup(r, \varepsilon_q)), \tag{12}$$

where Pr_{range} is defined in Eq. (1). Specially, if $\sqcap(r, \varepsilon_q)$ does not exist, then $Pr_{range}(o, \sqcap(r, \varepsilon_q)) = 0$.

We employ the above lemma to calculate $UB_{Pr}(r, o, \varepsilon_q)$ and $LB_{Pr}(r, o, \varepsilon_q)$, which are defined in Inequality (11). Specifically, we invoke the Algorithm 1 (Figure 5) with the parameters o and $\sqcup(r, \varepsilon_q)$, and then set $UB_{Pr}(r, o, \varepsilon_q)$ to

the upper bound returned by the algorithm. Similarly, $LB_{Pr}(r, o, \varepsilon_q)$ equals the lower bound produced by the same algorithm, when its parameters are o and $\sqcap(r, \varepsilon_q)$.

There remains, however, a subtle issue: the second parameter of Algorithm 1 must be a rectangle, whereas $\sqcup(r, \varepsilon_q)$ and $\sqcap(r, \varepsilon_q)$ may have irregular shapes under the L_2 norm (see Figures 9(c) and 9(d)). We remedy the problem using the approach explained with Figure 6(b). To derive $UB_{Pr}(r, o, \varepsilon_q)$ of L_2, we pass, instead of $\sqcup(r, \varepsilon_q)$, its MBR (e.g., $IJKL$ in Figure 9(c)) into the second parameter (notice that the MBR is essentially the $\sqcup(r, \varepsilon_q)$ of L_∞). Likewise, for computing $LB_{Pr}(r, o, \varepsilon_q)$, the parameter is set to an "inner rectangle" of $\sqcap(r, \varepsilon_q)$ created as follows. For every corner x of r, we connect it with its opposing corner using a line l. Then, the intersection between l and circle $\odot(x, \varepsilon_q)$ becomes a corner of the inner rectangle. After all the corners of r have been considered, the inner rectangle is fixed. As an example, in Figure 9(d), the inner rectangle of the illustrated $\sqcap(r, \varepsilon_q)$ is $EFGH$ (e.g., E is decided by considering the bottom-right corner of r).

4.2.2 *Bounds of $Pr_{fuzzy}(o, q, \varepsilon_q)$.* We are ready to explain how to determine the upper bound $UB_{fuzzy}(o, q, \varepsilon_q)$ and lower bound $LB_{fuzzy}(o, q, \varepsilon_q)$ of $Pr_{fuzzy}(o, q, \varepsilon_q)$. Recall that we have prepared m_q PCRs: $q.pcr(c)$ at $c = QC_1, ..., QC_{m_q}$. Let us consider the projections of these PCRs along the ith dimension ($1 \leq i \leq d$): $[q.pcr_{[i]-}(c), q.pcr_{[i]+}(c)]$. We slice $q.mbr$ with $2(m_q - 1)$ planes, which are perpendicular to the ith axis, and intersect this axis at $q.pcr_{[i]-}(c)$ or $q.pcr_{[i]+}(c)$, for every $c \in [2, m_q]$. The slicing divides $q.mbr$ into $2m_q - 1$ disjoint rectangles $r_1, r_2, ..., r_{2m_q-1}$, sorted in ascending order of their projections on the ith dimension.

For instance, assume that $m_q = 2$, and $QC_1 = 0$, $QC_2 = 0.3$; rectangle $ABCD$ and the grey box in Figure 8(a) illustrate $q.pcr(QC_1)$ and $q.pcr(QC_2)$, respectively. Lines IJ and EF are the two slicing planes at $q.pcr_{[1]-}(QC_2)$ and $q.pcr_{[1]+}(QC_2)$, where the subscript 1 denotes the horizontal dimension. The result of the slicing is $2m_q - 1 = 3$ rectangles $BCFE$, $IEFJ$, and $AIJD$.

LEMMA 4. *Let q be a fuzzy query with distance threshold ε_q, whose $q.mbr$ has been partitioned into rectangles $r_1, ..., r_{2m_q-1}$ on the ith dimension (for some $i \in [1, d]$) as described earlier. Then, for any object o:*

$$Pr_{fuzzy}(o, q, \varepsilon_q) \leq (1 - 2QC_{m_q}) \cdot UB_{Pr}(r_{m_q}, o, \varepsilon_q) + \sum_{i=1}^{m_q-1} (QC_{i+1} - QC_i) \cdot \Big[UB_{Pr}(r_i, o, \varepsilon_q) + UB_{Pr}(r_{2m_q-i}, o, \varepsilon_q)\Big]. \quad (13)$$

Similarly,

$$Pr_{fuzzy}(o, q, \varepsilon_q) \geq (1 - 2QC_{m_q}) \cdot LB_{Pr}(r, o, \varepsilon_q) + \sum_{i=1}^{m_q-1} (QC_{i+1} - QC_i) \cdot \Big[LB_{Pr}(r_i, o, \varepsilon_q) + LB_{Pr}(r_{2m_q-i}, o, \varepsilon_q)\Big]. \quad (14)$$

The computation of functions $UB_{range}(.)$ and $LB_{range}(.)$ has been discussed in Section 4.2.1.

The right-hand sides of Inequalities (13) and (14) can directly be taken as $UB_{fuzzy}(o, q, \varepsilon_q)$ and $LB_{fuzzy}(o, q, \varepsilon_q)$, respectively. Remember that, there is a set of these two inequalities for every dimension $i \in [1, d]$. To tighten the range $[LB_{fuzzy}(o, q, \varepsilon_q), UB_{fuzzy}(o, q, \varepsilon_q)]$ (for maximizing the pruning and validating power, as discussed at the beginning of Section 4.2), we set $UB_{fuzzy}(o, q, \varepsilon_q)$ (or $LB_{fuzzy}(o, q, \varepsilon_q)$) to the smallest (or largest) of the right-hand sides of the d versions of Inequality (13) (or Inequality (14)) on the d dimensions, respectively.

5. QUERY ALGORITHMS

The analysis in Section 3 (or 4) leads to an algorithm that processes a nonfuzzy (or fuzzy) query by scanning the entire dataset. Specifically, the filter step of the algorithm decides whether each object can be pruned, validated, or must be added to a candidate set, by analyzing the PCRs of o (for fuzzy search, also the PCRs of the query object q). Then, the refinement phase calculates the accurate qualification probability of every object in the candidate set, to determine if it satisfies the query.

This section achieves two objectives. First, we design an access method, called the U-tree, for multidimensional uncertain data (Section 5.1), and utilize it to reduce the cost of the filter step, by avoiding examination of all the objects (Section 5.2). Second, we clarify the details of refinement, and develop solutions with different trade-offs between precision and computation overhead (Section 5.3).

5.1 The U-tree

The U-tree is a balanced external-memory structure, where each node occupies a disk page. We number the levels of the tree in a bottom-up manner; namely, if the tree has a height of h, then all the leaf nodes are at level 0, whereas the root is at level $h-1$. Each entry in a leaf node corresponds an object o. This entry keeps (i) $o.pcr(c)$ for the values of c in the U-catalog: C_1 (= 0), ..., C_m (recall that $o.pcr(0)$ equals $o.mbr$), and (ii) a descriptor about $o.pdf$, whose information depends on the complexity of $o.pdf$. Specifically, if $o.pdf$ is simple (i.e., a common distribution with a regular uncertainty region $o.ur$), then the descriptor contains all the details of $o.pdf$. Otherwise, the descriptor is a pointer that references a disk address where the representation of $o.pdf$ (e.g., a histogram) is stored; in this case, additional I/Os are required to retrieve $o.pdf$ after the leaf entry has been found.

Let e be a level-1 entry, that is, e is the parent entry of a leaf node. Without loss of generality, assume that the node has f objects o_1, ..., o_f. In addition to a pointer to its child node, e also retains (i) m rectangles $e.mbr(C_1)$ ..., $e.mbr(C_m)$, where, for any c in the U-catalog, $e.mbr(c)$ is the MBR of $o_1.pcr(c)$, ..., $o_f.pcr(c)$, and (ii) m values $e.sl(C_1)$, ..., $e.sl(C_m)$ such that $e.sl(c)$ (here, "sl" means side length) equals the length of the shortest projection of $o_1.pcr(c)$, ..., $o_f.pcr(c)$ along all dimensions.

Figure 10 provides an example that illustrates the information recorded in leaf and level-1 entries, assuming that the U-catalog has $m = 2$ values $C_1 = 0$ and $C_2 = 0.3$. The left- and right-dashed rectangles correspond to the MBRs of objects o_1 and o_2, (they are $o_1.pcr(0)$ and $o_2.pcr(0)$, respectively). The grey

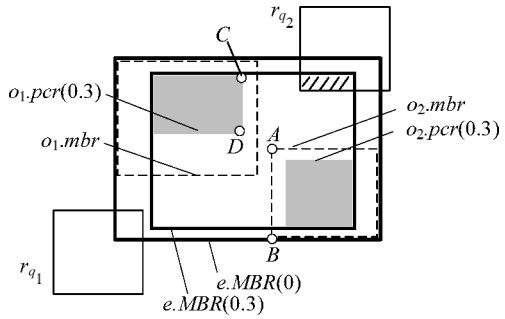


Fig. 10. A leaf node containing objects o_1 and o_2.

box inside $o_1.mbr$ (or $o_2.mbr$) is $o_1.pcr(0.3)$ (or $o_2.pcr(0.3)$), which is associated with the leaf entry of o_1 (or o_2). Consider a leaf node that contains only o_1, o_2, and has e as its parent entry at level 1. Entry e carries two rectangles $e.mbr(0)$ and $e.mbr(0.3)$. As shown in Figure 10, the former rectangle tightly bounds the MBRs of o_1 and o_2, while $e.mbr(0.3)$ is the MBR of $o_1.pcr(0.3)$ and $o_2.pcr(0.3)$. Furthermore, e also stores two values $e.sl(C_1)$ and $e.sl(C_2)$, which equal the lengths of segments AB and CD, respectively. Specifically, $e.sl(C_1) = AB$ because the vertical edge of $o_2.mbr$ is the shortest among all the edges of $o_1.mbr$ and $o_2.mbr$. Similarly, $e.sl(C_2) = CD$ since the vertical edge of $o_1.pcr(0.3)$ has the smallest length among all the edges of $o_1.pcr(0.3)$ and $o_2.pcr(0.3)$.

An entry e of a higher level $i > 1$ has similar formats. To elaborate this, suppose that the child node of e has (intermediate) entries $e_1, \ldots, e_f$. Then, e is associated (i) a pointer to the node, (ii) m rectangles $e.mbr(C_1), \ldots, e.mbr(C_m)$, where $e.mbr(c)$ is the MBR of $e_1.mbr(c), \ldots, e_f.mbr(c)$, for any c in the U-catalog, and (iii) m values $e.sl(C_1), \ldots, e.sl(C_m)$ such that $e.sl(c)$ is the smallest of $e_1.sl(c), \ldots, e_f.sl(c)$.

Note that an intermediate entry e in the U-tree consumes more space than a leaf entry. In particular, e keeps $e.sl(c)$, which is not present at the leaf level (but is needed for improving the I/O performance, as explained in the next subsection). In general, it is reasonable to retain more information at the intermediate levels, if such information can reduce the number of leaf nodes accessed. After all, in processing a query, the cost at the leaf level usually significantly dominates the overall overhead.

The U-tree is dynamic, because objects can be inserted/deleted in an arbitrary order, by resorting to the update algorithms of the R*-tree [Beckmann et al. 1990]. Specifically, a U-tree is analogous to an R*-tree built on the $o.pcr(C_{\lceil m/2 \rceil})$ of the objects ($C_{\lceil m/2 \rceil}$ is the median value in the U-catalog). The difference is that, (conceptually) after objects have been grouped into leaf nodes, the contents of the intermediate entries need to be "filled in" as mentioned earlier. Clearly, with this analogy, a U-tree can also be constructed with the bulkloading algorithm of R*-trees [Leutenegger et al. 1997].

5.2 The Filter Step

Given a nonfuzzy range query q (with search region r_q and probability threshold t_q), the filter step traverses the U-tree in a depth-first manner. Specifically,

Algorithm 2: `Nonfuzzy-Range-Quali-Prob-Upper-Bound`(e, r_q)

/* e is an intermediate entry of a U-tree and r_q a rectangle. The algorithm returns a value $UB_{range}(e, r_q)$ that upper bounds the $Pr_{range}(o, r_q)$ (Equation 1) of any object o in the subtree of e. */

1. $UB_{range}(e, r_q) = 1$
2. for i = 1 to m /* consider the U-catalog values in ascending order */
3. if r_q is disjoint with $e.mbr(C_i)$
4. $UB_{range}(e, r_q) = C_i$; return
5. for $i = m$ downto 1 /* consider the U-catalog values in descending order */
6. r = the intersection between $e.mbr(C_i)$ and r_q /* r is a rectangle */
7. if the projection length of r on any dimension is smaller than $e.sl(C_i)$
8. $UB_{range}(e, r_q) = 1 - C_i$; return

Fig. 11. Finding an upper bound of an object's qualification probability (nonfuzzy range search).

the search starts by accessing the root. For each root entry e, the algorithm computes an upper bound $UB_{range}(e, r_q)$ of the qualification probability of any object that lies in the subtree of e. If $UB_{range}(e, r_q)$ is smaller than t_q, the subtree of e is pruned; otherwise, we fetch the child node of e, and carry out the above operations recursively for the entries encountered there. When a leaf node is reached, we attempt to prune or validate the objects discovered. Objects that can neither be pruned nor validated are added to a candidate set S_{can}. Then, the search backtracks to the previous level, and continues this way until no more subtree needs to be visited.

The filter phase of a fuzzy query q (with distance and probability thresholds ε_q and t_q, respectively) is exactly the same, except that $UB_{range}(e, r_q)$ is replaced with $UB_{fuzzy}(e, q, \varepsilon_q)$, which upper bounds the $UB_{fuzzy}(o, q, \varepsilon_q)$ (as in Eq. (3)) of any object o underneath e.

It remains to clarify the computation of $UB_{range}(e, r_q)$ and $UB_{fuzzy}(e, q, \varepsilon_q)$. We settle the former with Algorithm 2 (Figure 11), assuming a rectangular r_q. To illustrate the algorithm, let us consider Figure 10 again, where, as mentioned earlier, the U-catalog has $m = 2$ values $C_1 = 0$, $C_2 = 0.3$, and e is the parent entry of the leaf node containing only o_1 and o_2. Rectangles r_{q_1} and r_{q_2} are the search regions of two nonfuzzy range queries q_1 and q_2, respectively. Given parameters e and r_{q_1}, Algorithm 2 returns (at Line 4) $UB_{range}(e, r_{q_1}) = 0.3$. This is because r_{q_1} does not intersect $e.mbr(0.3)$, which indicates that the $o.pcr(0.3)$ of any object o must be disjoint with r_{q_1}. Hence, according to Rule 2 of Theorem 1, o cannot appear in r_{q_1} with a probability at least 0.3.

On the other hand, given e and r_{q_2}, Algorithm 2 produces (at Line 8) $UB_{range}(e, r_{q_2}) = 0.7$. To explain why, we need to focus on the hatched area of Figure 10, which is the intersection between r_{q_2} and $e.mbr(0.3)$, and is the rectangle r computed at Line 6 of Algorithm 2. Observe that the length of the vertical projection of r is shorter than $e.sl(0.3)$, which, as discussed in Section 5.1, equals the length of segment CD. This implies that none of the $o.pcr(0.3)$ of any object o in the subtree of e can possibly be entirely covered by r_{q_2}. As a result, by Rule 1 of Theorem 1, o falls in r_{q_2} with a probability less than 0.7.

When r_q is not rectangular, $UB_{range}(e, r_q)$ can be calculated using Algorithm 2, by passing the MBR of r_q as the second parameter. Finally, the $UB_{fuzzy}(e, q, \varepsilon_q)$ for a fuzzy query q can also be calculated using Algorithm 2, leveraging the reduction proposed in Section 4.2. Instead of repeating the theoretical reasoning

Algorithm 3: `Fuzzy-Quali-Prob-Upper-Bound`(e, q, ε_q)

/* e is an intermediate entry of a U-tree, q a query uncertain object, and ε_q the distance threshold of q. Applicable to both the L_∞ and L_2 norms, the algorithm returns a value $UB_{fuzzy}(e, q, \varepsilon_q)$ that upper bounds the $Pr_{fuzzy}(o, q, \varepsilon_q)$ (Equation 3) of any object o in the subtree of e. */

1. $UB_{fuzzy}(e, q, \varepsilon_q) = 1$
2. for $i = 1$ to d /* consider each dimension in turn */
3. partition $q.mbr$ into rectangles r_1, ..., r_{2m_q-1} as in Lemma 4 /* m_q is defined in Section 4.2 */
4. for $j = 1$ to $2m_q - 1$
5. $UB_{Pr}(r_j, e, \varepsilon_q) =$ `Range-Quali-Prob-Upper-Bound`$(e, \sqcup(r_j, \varepsilon_q)$ of $L_\infty)$
   /* Algorithm 2 is given in Figure 11 */
   /* $UB_{Pr}(r_j, e, \varepsilon_q)$ is an upper bound of $Pr_{range}(o, \odot(x, \varepsilon_q))$ for any object o in the subtree of e, and any point x in r_j. The computation of $\sqcup(r_j, \varepsilon_q)$ of L_∞ is illustrated in Figure 9a */
6. UB = the right hand side of Inequality 13, replacing all occurrences of o with e
7. $UB_{fuzzy}(e, q, \varepsilon_q) = \min\{UB_{fuzzy}(e, q, \varepsilon_q), UB\}$

Fig. 12. Finding an upper bound of an object's qualification probability (fuzzy search).

of the reduction, we simply present the details in Figure 12, which applies to both the L_∞ and L_2 norms.

5.3 The Refinement Step

For each object o in the S_{can} output by the filter step, the refinement phase calculates the qualification probability of o, for comparison with the probability threshold t_q. Next, we elaborate the details of the calculation, starting with $Pr_{range}(o, r_q)$ for a nonfuzzy range query q, before discussing $Pr_{fuzzy}(o, q, \varepsilon_q)$ for a fuzzy query q.

If $Pr_{range}(o, r_q)$ can be solved into a closed equation, its computation entails negligible cost. For instance, when $o.ur$ and r_q are rectangles and $o.pdf$ describes a uniform distribution, $Pr_{range}(o, r_q)$ is simply the ratio between the areas of $o.ur \cap r_q$ and r_q, both of which can be easily computed. In general, however, integrating a complex multidimensional function (i.e., $o.pdf$) over a potentially irregular region $o.ur \cap r_q$ is a well-known difficult problem, for which the Monte-Carlo (MC) method [Press et al. 2002] is a standard remedy.

Specifically, to apply MC, we need to formulate a function $f(x)$, which equals $o.pdf(x)$ for a d-dimensional point x in $o.ur \cap r_q$, or 0 for any other x. Then, we take the MBR r_{mbr} of r_q (note that r_q may not be a rectangle), and uniformly generate a number s of points inside $r_{mbr} \cap o.MBR$ (which is a rectangle). Let us denote these points as $x_1, \ldots, x_s$, respectively. Equation (1) can be estimated as (using E to denote the estimate):

$$E = vol \cdot \frac{1}{s} \sum_{i=1}^{s} f(x_i), \tag{15}$$

where vol returns the volume of $r_{mbr} \cap o.MBR$. The value of s may need to be really large in order to produce an accurate estimate. This is why refinement of an object can be rather costly, and should be prevented as much as possible.

$Pr_{fuzzy}(o, q, \varepsilon_q)$ is more expensive to evaluate, because Eq. (3) essentially has two layers of integrals. In particular, the inner layer derives $Pr_{range}(x, \odot(o, \varepsilon_q))$, which can be settled as described earlier. The outer layer can also be calculated by Eq. (15) with the following changes. First, function $f(x)$ should be replaced

with $g(x)$, which equals 0 if x lies outside $q.ur$; otherwise, $g(x)$ is the integrand of Eq. (3). Second, r_{mbr} now becomes $q.mbr$.

6. COST-BASED INDEX OPTIMIZATION

The U-tree takes a parameter m, which is the size of the U-catalog, and has a significant impact on the query performance. A large m leads to more precomputed PCRs, which reduce the chance of calculating an object's actual qualification probability, and hence, the overhead of the refinement step. On the other hand, as m increases, the node fanout decreases, which adversely affects the I/O efficiency of the filter phase.

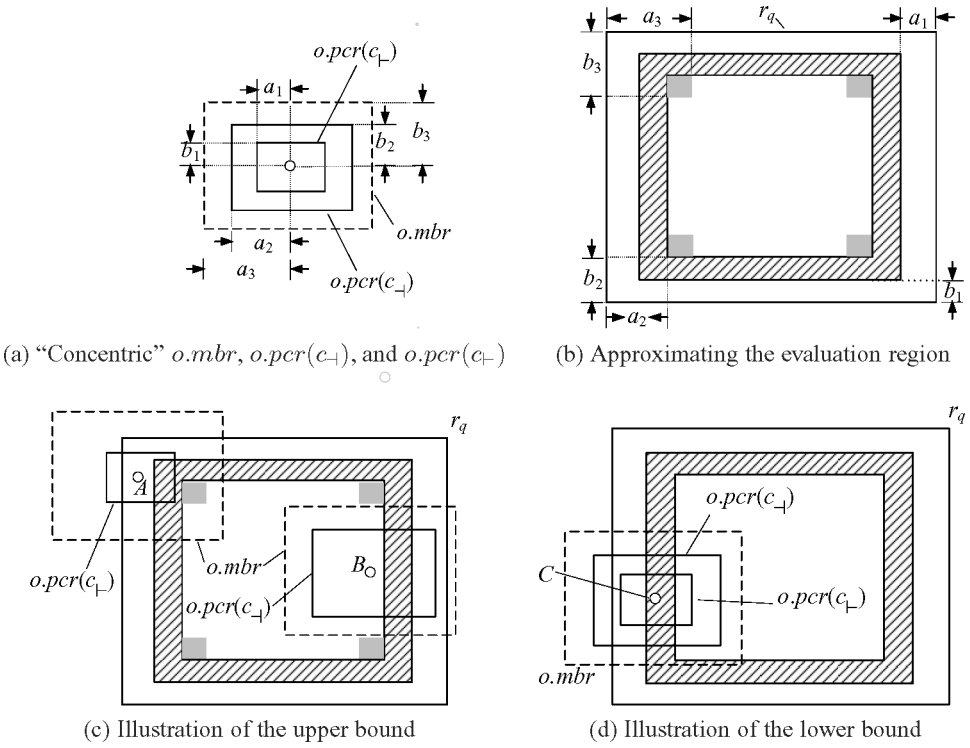
The best m depends on the dataset characteristics. As an extreme example, imagine that all uncertainty regions are so small that their extents can be ignored. In this case, the dataset degenerates into a set of points, for which (intuitively) the best index is simply an R*-tree, or a special U-tree with $m = 1$ (i.e., for each object o, only $o.pcr(0) = o.mbr$ is stored). On the other hand, consider an object o that has a sizable uncertainty region $o.ur$, and a Gaussian pdf with a large variance (i.e., $o.pdf(x)$ peaks at the center of $o.ur$, but quickly diminishes as x drifts away). It is beneficial to materialize $o.pcr(c)$ at some values of $c \in (0, 0.5]$, all of which have significantly smaller extents than $o.mbr$, and effectively prevent the refinement of o.

In the sequel, we provide a method for deciding a good value of m prior to the construction of a U-tree. Our objective is to minimize the cost of nonfuzzy range search (Definition 1). Towards this purpose, Section 6.1 first analyzes how often PCRs can prevent the numerical process discussed in Section 5.3. Then, Section 6.2 derives a formula that accurately captures both the filter and refinement overhead. Finally, Section 6.3 applies the cost model to optimize m for an arbitrary dataset.

6.1 Probability of Numerical Evaluation

Consider an object o with pre-computed $o.pcr(C_1),\ldots, o.pcr(C_m)$, where $C_1,\ldots, C_m$ are the values in the U-catalog. Let q be a nonfuzzy range query with probability threshold t_q and a rectangular search region r_q whose projection on the ith dimension has length $sl_{q[i]}$ (much smaller than 1), and its centroid is uniformly distributed in the workspace. We aim at deriving the probability $o.Pr_{comp}$ that $Pr_{range}(o, r_q)$ must be numerically computed in processing q. Alternatively, $o.Pr_{comp}$ is the likelihood that o can be neither pruned by Theorem 3 nor validated by Theorem 2.

We are mainly interested in query regions that are "larger" than $o.mbr$. Formally, if the projection length of $o.mbr$ on the ith $(1 \le i \le d)$ dimension is $o.sl_{mbr[i]}$, then $sl_{q[i]} \ge o.sl_{mbr[i]}$ for all $i \in [1, d]$. The reasons for concentrating on such *voluminous queries* are two fold. First, they are (much) more expensive than queries with small search regions, and hence, the target of optimization in finding an appropriate U-catalog size m. Second, they simplify Theorem 2, which allows us to avoid excessively complex equations. Specifically, for a voluminous query, at least one value in each pair of c_i, c_i' $(1 \le i \le d)$ must be 0 in Rule 1 of Theorem 2; likewise, in Rule (2), either c_1 or c_1' equals 0.



(a) "Concentric" $o.mbr$, $o.pcr(c_\dashv)$, and $o.pcr(c_\vdash)$ (b) Approximating the evaluation region

(c) Illustration of the upper bound (d) Illustration of the lower bound

Fig. 13. Analysis of $o.Pr_{comp}$ for $t_q > 1 - C_m$.

Our analysis on voluminous nonfuzzy range search proceeds in two parts, focusing on large t_q (Section 6.1.1), small t_q (Section 6.1.2), and median t_q (Section 6.1.3), respectively. Finally, Section 6.1.4 mathematically explains why PCRs are a useful tool for reducing the refinement cost, and elaborates how to support nonvoluminous queries.

6.1.1 *Case* 1: $t_q > 1 - C_m$. As opposed to the original settings (the location of $o.mbr$ is fixed while that of r_q is uniformly distributed), for deriving $o.Pr_{comp}$, it is more convenient to consider the equivalent opposite. Specifically, we fix r_q, but move the centroid of $o.mbr$ around in the workspace, following a uniform distribution. In doing so, we keep track of whether o can be pruned/validated when its MBR equals the current $o.mbr$. After $o.mbr$ has been placed at all possible locations, we have collected an *evaluation region* (ER), which consists of all the centroids of $o.mbr$ that do not allow our heuristics to prune and validate o. Thus, $o.Pr_{comp}$ is exactly the area of this region (remember that the workspace has an area 1).

Assume that $o.mbr$ is the dashed rectangle in Figure 13(a), which also demonstrates the $o.pcr(c_\dashv)$ and $o.pcr(c_\vdash)$ of o, where $c_\vdash$ ($c_\dashv$) is the smallest (largest) U-catalog value at least (most) $1 - t_q$. Here, $o.pcr(c_\dashv)$, $o.pcr(c_\vdash)$, and $o.mbr$ are "concentric", that is, they have the same centroid. Although the concentric behavior is not always true, it facilitates explaining the crux of our analysis.

Later, we will generalize the results to the general situation where PCRs are not concentric.

The outmost rectangle in Figure 13(b) corresponds to the query region r_q. Next, we will construct an area that closely approximates the ER. The hatched region in Figure 13(b) is a *ring* bounded by two rectangles, both of which have the same centroid as r_q. In particular, the outer (inner) rectangle is shorter than r_q by $2a_1$ ($2a_2$) on the horizontal dimension, and by $2b_1$ ($2b_2$) on the vertical dimension. As illustrated in Figure 13(a), $2a_1$, $2b_1$ (or $2a_2$, $2b_2$) are the lengths of the horizontal and vertical edges of $o.pcr(c_\vdash)$ (or $o.pcr(c_\dashv)$), respectively. In Figure 13(b), there are 4 *grey boxes* near the corners of the ring. The upper-left grey box is decided by (i) the (inner) corner of the ring that the box is adjacent to, and (ii) the point inside r_q, having horizontal (vertical) distance a_3 (b_3) from the upper-left corner of r_q, where a_3 and b_3 are the projection lengths of $o.mbr$ (see Figure 13(a)). The other grey boxes are obtained in the same way, but with respect to other corners of the ring and r_q.

When the centroid of o.mbr falls outside the hatched and grey area in Figure 13(b), o can always be pruned or validated. For example, if the centroid lies at point A in Figure 13(c), r_q does not fully cover $o.pcr(c_\vdash)$; hence, o is eliminated by Rule 1 of Theorem 3. On the other hand, if the centroid falls at B, o can be validated by Rule 1 of Theorem 2, setting $c_1 = 0$ and $c'_1 = c_\vdash$ (the subscript 1 represents the horizontal dimension). On the other hand, *as long as the centroid of o.mbr lies in the hatched region in Figure 13(b), o can never be pruned/validated*, but always requires numeric evaluation of $o.Pr_{comp}$. For instance, let us examine the case where the centroid lies at point C in Figure 13(d). Given parameters o and r_q, Algorithm 1 returns $[1-c_\vdash, 1-c_\dashv]$, which contains t_q. Hence, as proved in Section 3.4, o can be neither pruned by Theorem 1 nor validated by Theorem 2.

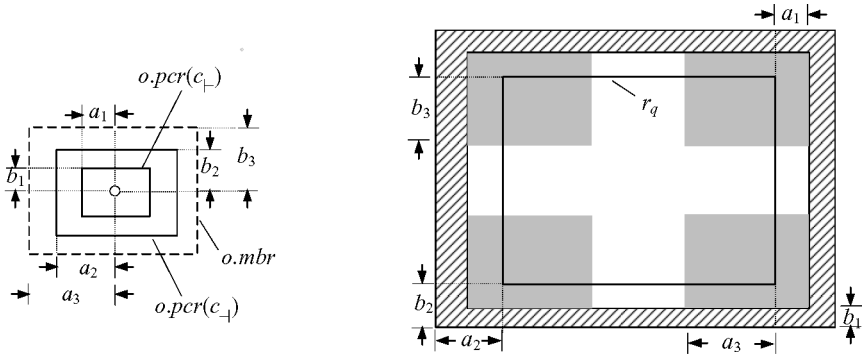
The implication of the above discussion is that $o.Pr_{comp}$ is at most the total area of the hatched and grey regions, but at least the area of the hatched region itself. Formally, $o.Pr_{comp} \in [o.Pr^{LB}_{comp}, o.Pr^{UB}_{comp}]$, with

$$o.Pr^{LB}_{comp} = \prod_{i=1}^{d} \left(sl_{q[i]} - o.sl_{pcr[i]}(c_\vdash)\right) - \prod_{i=1}^{d} \left(sl_{q[i]} - o.sl_{pcr[i]}(c_\dashv)\right), \text{ and} \quad (16)$$

$$o.Pr^{UB}_{comp} = o.Pr^{LB}_{comp} + \prod_{i=1}^{d} \left(o.sl_{mbr[i]} - o.sl_{pcr[i]}(c_\dashv)\right), \quad (17)$$

where function $o.sl_{pcr[i]}(c)$ gives the projection length of $o.pcr(c)$ on the ith dimension. For example, in Figure 13(a), $o.sl_{pcr[1]}(c_\vdash) = 2a_1$ and $o.sl_{pcr[2]}(c_\vdash) = 2b_1$. Notice that we presented the above equations in their general forms applicable to all dimensionalities. In fact, the construction of the approximate ER in Figure 13(b) can be extended to any dimensionality in a straightforward manner; the resulting ring and grey boxes possess the same properties.

So far we have implicitly assumed the presence of $c_\vdash$, which, however, is not always true. If $c_\vdash$ does not exist, Rule 1 of Theorem 3 is no longer applicable; thus, pruning relies on Rule (2), where, for t_q over $1-C_m$, the value c equals C_m. Accordingly, the outer boundary of the ring in Figure 13(b) becomes a rectangle



(a) Concentric $o.mbr$, $o.pcr(c_{\dashv})$, and $o.pcr(c_{\vdash})$ (b) Approximating the evaluation region

Fig. 14. Analysis of $o.Pr_{comp}$ for $t_q \leq C_m$.

(again, sharing the centroid of r_q) with a projection length longer than that of r_q by $o.sl_{pcr[i]}(C_m)$ on the ith dimension ($1 \leq i \leq 2$). As a result, $sl_{q[i]} - o.sl_{pcr[i]}(c_{\vdash})$ should be replaced with $sl_{q[i]} + o.sl_{pcr[i]}(C_m)$ in Eq. (16).

Equations (16) and (17) are valid even if the PCRs of an object are not concentric. In that case, the only modification to all our analysis lies in the ring construction. To explain this, let p be the centroid of $o.mbr$ with coordinates $p[1],\ldots, p[d]$. On each dimension $i \in [1, d]$, the left (or right) edge of the outer rectangle of the ring is obtained by moving the left (or right) edge of r_q inward at the distance of $p[i] - o.pcr_{[i]-}(c_{\vdash})$ (or $o.pcr_{[i]+}(c_{\vdash}) - p[i]$). The inner rectangle is decided in the same way, except that $c_{\vdash}$ is replaced with $c_{\dashv}$.

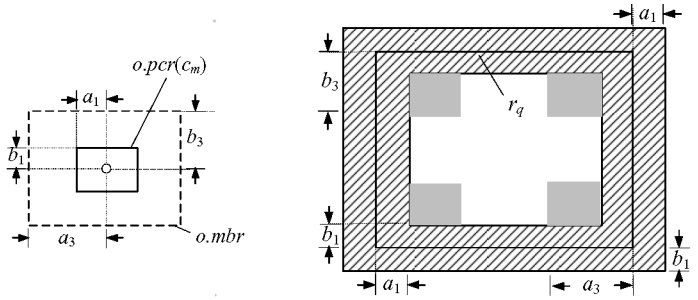
6.1.2 *Case 2: $t_q \leq C_m$.* We will derive $o.Pr_{comp}$ using the methodology in Section 6.1.1. Figure 14(a) repeats the content of Figure 13(a), except that here $c_{\vdash}$ ($c_{\dashv}$) should be understood as the smallest (largest) U-catalog value at least (most) t_q. The approximate ER (evaluation region) also consists of a ring (the hatched region) and four grey boxes. Specifically, the outer (inner) rectangle of the ring shares a common centroid with r_q, but is longer than r_q by $2a_2$ ($2a_1$) and $2b_2$ ($2b_1$) on the horizontal and vertical dimensions, respectively. The grey boxes are obtained in the same way as in Figure 13(b).

The approximate ER bears two properties identical to those in the previous subsection. Namely, o can definitely be pruned or validated if the centroid of $o.mbr$ is outside the hatched and grey area, whereas $Pr_{range}(o, r_q)$ must be numerically calculated if the centroid falls in the hatched region. Therefore, $o.Pr_{comp}$ is guaranteed to fall in a range $[o.Pr^{LB}_{comp}, o.Pr^{UB}_{comp}]$, where the lower and upper bounds are given by two equations analogous to Eqs. (16) and (17):

$$o.Pr^{LB}_{comp} = \prod_{i=1}^{d} \left(sl_{q[i]} + o.sl_{pcr[i]}(c_{\dashv})\right) - \prod_{i=1}^{d} \left(sl_{q[i]} + o.sl_{pcr[i]}(c_{\vdash})\right), \text{ and} \quad (18)$$

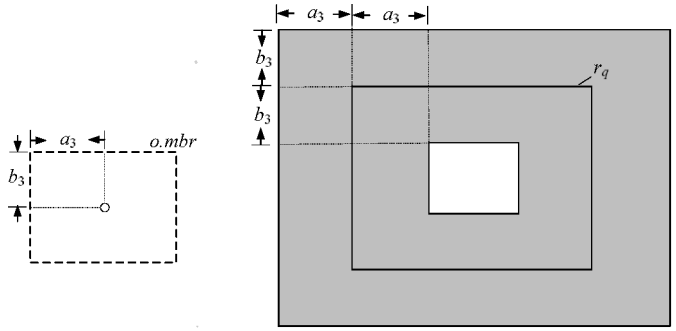
$$o.Pr^{UB}_{comp} = o.Pr^{LB}_{comp} + \prod_{i=1}^{d} \left(o.sl_{mbr[i]} + o.sl_{pcr[i]}(c_{\vdash})\right), \quad (19)$$

Finally, the above formulae are correct in any dimensionality, even when the PCRs of an object are not concentric.



(a) Concentric $o.mbr$, and $o.pcr(c_m)$ (b) Approximating the evaluation region

Fig. 15. Analysis of $o.Pr_{comp}$ for $C_m < t_q \leq 1 - C_m$.



(a) Extents of $o.mbr$ (b) The evaluation region without PCRs (for any t_q)

Fig. 16. Explanation about why refinement is more frequent without PCRs.

6.1.3 *Case 3: $C_m < t_q \leq 1 - C_m$.* This remaining case is the simplest: both $c_{\vdash}$ and $c_{\dashv}$ correspond to c_m, that is, the largest U-catalog value. The counterpart of Figures 13 and 14 here is Figure 15. Since the derivation is similar to that of the previous two cases, we directly present the final equations of $o.Pr^{LB}_{comp}$ and $o.Pr^{UB}_{comp}$ (which, again, are applicable in any dimensionality, regardless of whether $o.mbr$ and $o.pcr(c_m)$ are concentric):

$$o.Pr^{LB}_{comp} = \prod_{i=1}^{d} \left(sl_{q[i]} + o.sl_{pcr[i]}(c_m)\right) - \prod_{i=1}^{d} \left(sl_{q[i]} - o.sl_{pcr[i]}(c_m)\right), \text{ and} \quad (20)$$

$$o.Pr^{UB}_{comp} = o.Pr^{LB}_{comp} + \prod_{i=1}^{d} \left(o.sl_{mbr[i]} - o.sl_{pcr[i]}(c_m)\right), \quad (21)$$

6.1.4 *Discussion.* The analysis of Sections 6.1.1 through 6.1.3 also explain why PCRs can effectively reduce the refinement cost. Imagine that we do not have any PCR. In this case, regardless of t_q, pruning (or validation) of o is possible if and only if r_q does not intersect (or completely covers) $o.mbr$. With respect to the extents of $o.mbr$ in Figure 16(a), the grey portion of Figure 16(b) illustrates the exact evaluation region of o, which unions the centroids of all $o.mbr$ causing $o.Pr_{comp}$ to be numerically computed. The region is a ring, whose

outer (inner) boundary is a rectangle that shares the centroid of r_q, and is longer (shorter) than r_q by $2a_3$ on the horizontal dimension, and by $2b_3$ on the vertical dimension. The area of the evaluation region is much larger than those of the approximate evaluation regions in Figures 13(b), 14(b), and 15(b), which, as explained earlier, give *pessimistic upper bounds* of $o.Pr_{comp}$ (for large, small, and median t_q respectively) when PCRs are used.

The above discussion applies only to voluminous queries (i.e., the projection of r_q is longer than that of $o.mbr$ on every dimension). When a query is not voluminous, its evaluation region is significantly more complex, because o can be validated in many additional ways (the voluminous requirement simplifies Theorem 2, as mentioned at the beginning of Section 6.1). However, $o.Pr_{comp}$ can still be estimated as follows. First, we generate a large number s_1 of $o.mbr$, by randomly distributing their centroids in the workspace. Then, we count the number s_2 of $o.mbr$ that leads to numerical evaluation of $Pr_{range}(o, r_q)$, after which $Pr_{comp}(o, r_q)$ can be approximated as s_1/s_2. We note that nonvoluminous queries are much less important than the voluminous counterpart for index optimization, as explained at the beginning of Section 6.1.

6.2 A Cost Model

In this section, we will derive analytical formulae that quantify the overhead of nonfuzzy range search on multidimensional uncertain data. Specifically, let q be a query with a probability threshold t_q, and a rectangular search region r_q that has projection length $sl_{q[i]}$ on the ith dimension ($1 \leq i \leq d$), and its centroid follows a uniform distribution in the workspace. The objective is to compute the expected query time $cost(q)$, which sums the cost $cost_{flt}(q)$ of the filter step, and the refinement overhead $cost_{rfn}(q)$.

In Section 6.2.1, we settle the problem for a "regular" dataset generated as follows. First, we create n objects o with the same $o.ur$ and $o.pdf$. Then, these objects are positioned in the workspace such that the centroids of their $o.mbr$ distribute uniformly. Section 6.2.2 generalizes our analytical results to arbitrary datasets.

6.2.1 *Regular Data.* In a regular dataset, the PCRs (at the same U-catalog value c) of all objects are equally large, that is, $o_1.sl_{pcr[i]}(c) = o_2.sl_{pcr[i]}(c)$ for any dimension $i \in [1, d]$ and arbitrary objects o_1, o_2. Therefore, for every object o, the likelihood $o.Pr_{comp}$ that o can be neither pruned nor validated is equivalent. This leads to:

$$n_{NE} = n \cdot o.Pr_{comp}, \tag{22}$$

where n_{NE} is the number of numerical evaluations needed to answer a query. As for $o.Pr_{comp}$, we employ the following estimation:

$$o.Pr_{comp} = \left(o.Pr_{comp}^{LB} + o.Pr_{comp}^{UB}\right)/2, \tag{23}$$

where $o.Pr_{comp}^{LB}$ and $o.Pr_{comp}^{UB}$ are represented in Eqs. (16) and (17) for $t_q > 1-C_m$, Eqs. (18) and (19) for $t_q \leq C_m$, or Eqs. (20) and (21) for $C_m < t_q \leq 1-C_m$. Hence,

$$cost_{rfn} = n_{NE} \cdot o.cost_{rfn} = n \cdot o.Pr_{comp} \cdot o.cost_{rfn}, \tag{24}$$

where $o.cost_{rfn}$ is the overhead of MC (Monte-Carlo) for calculating the qualification probability of a single object (see Section 5.3). $o.cost_{rfn}$ is identical for all objects, since it depends only on the cost of loading $o.pdf$ and the number of samples used in MC.

On the other hand, $cost_{flt}(q)$ corresponds to the time of accessing the leaf nodes of the U-tree in the filter step (we do not include the overhead of visiting the intermediate nodes, since it is by far dominated by the cost at the leaf level, especially if the intermediate levels are buffered). The probability $nd.Pr_{acs}$ that a leaf node nd is accessed depends on the characteristics of the data inside nd. For a regular dataset, the characteristics are the same across the entire workspace; hence, $nd.Pr_{acs}$ is equivalent for all leaf nodes. It follows that:

$$n_{NA} = n \cdot o.Pr_{acs}, \tag{25}$$

where n_{NA} the number of nodes accessed in processing a query. Therefore,

$$cost_{flt}(q) = n_{NA} \cdot cost_{ranIO} = (n/f) \cdot nd.Pr_{acs} \cdot cost_{ranIO}, \tag{26}$$

where f is the average fanout of a node, n/f is the total number of leaf nodes, and $cost_{ranIO}$ the time of a random I/O. Note that f is determined by the page size, and very importantly, the U-catalog size m (recall that each leaf entry keeps m PCRs of an object).

Let e be the parent entry of a leaf node nd. As mentioned in Section 5.1, for each U-catalog value c, e retains (i) $e.mbr(c)$, which is the MBR of the $o.pcr(c)$ of all the objects o in nd, and (ii) a value $e.sl(c)$, equal to the smallest projection length of any $o.pcr(c)$ on any dimension. In the sequel, we use $e.sl_{mbr[i]}(c)$ to denote the projection length of $e.mbr(c)$ on the ith dimension.

Calculating $nd.Pr_{acs}$ requires the values of $e.sl_{mbr[i]}(c)$ and $e.sl(c)$. The analysis of the former can be reduced to estimating the MBR size of a leaf node in an R-tree, by regarding nd as a leaf R-tree node, and $e.mbr(c)$ its MBR. Leveraging the findings[3] of Theodoridis and Sellis [1996], we have:

$$e.sl_{mbr[i]}(c) = o.sl_{pcr[i]}(c) + (f/n)^{1/d} \tag{27}$$

where $o.sl_{pcr[i]}(c)$ captures the projection length of the $o.pcr(c)$ of an object o on the ith dimension. On the other hand, the analysis of $e.sl(c)$ is straightforward:

$$e.sl(c) = \min_{i=1}^{d} o.sl_{pcr[i]}(c). \tag{28}$$

We are ready to elaborate the derivation of $nd.Pr_{acs}$. Recall that nd must be visited, if and only if the following "access condition" holds: t_q is at most the $UB_{range}(e, r_q)$ returned by Algorithm 2. Let us first consider the case $t_q > 1 - C_m$, where we use $c_{\vdash}$ to denote the smallest U-catalog value at least $1 - t_q$. Thus, the access condition is satisfied only when the rectangle $r_q \cap e.mbr(c_{\vdash})$ has a

[3]Consider an R-tree that indexes n equivalent rectangles, which are uniformly distributed in a d-dimensional workspace, and their side lengths on the ith ($1 \le i \le d$) dimension equal $x[i]$. Let $y[i]$ denote the side length, on the ith dimension, of the MBR of a level-1 entry in an R-tree. If each dimension of the workspace has a unit length, $y[i]$ can be accurately estimated as $x[i] + (f/n)^{1/d}$, where f is the average tree fanout. This result is due to the grid-modeling of the MBRs of the leaf nodes in Theodoridis and Sellis [1996].

projection length at least $e.sl(c_{\vdash})$ on all the dimensions. Note that $c_{\vdash}$ may not exist, in which scenario the access condition is equivalent to r_q intersecting $e.mbr(0)$. Similarly, for $t_q \leq 1 - C_m$, we deploy $c_{\dashv}$ to represent the largest U-catalog value at most t_q (there is always such a value). Accordingly, the access condition is valid if and only if $e.mbr(c_{\dashv})$ intersects r_q.

It is clear from the above discussion that $nd.Pr_{acs}$ is essentially the probability that (a uniformly distributed) r_q intersects a rectangle $e.mbr(c)$ in a certain way, where c is an appropriate U-catalog value selected as mentioned earlier. This is a problem that has been studied by Pagel et al. [1993]. Based on their results[4] we have $nd.Pr_{acs} =$

$$\begin{cases} \prod_{i=1}^{d} \left(sl_{q[i]} + e.sl_{mbr[i]}(c_{\vdash}) - e.sl(c_{\vdash})\right) & ift_q > 1 - C_m \text{ and } c_{\vdash} \text{ exists} \\ \prod_{i=1}^{d} \left(sl_{q[i]} + e.sl_{mbr[i]}(0)\right) & ift_q > 1 - C_m \text{ and } c_{\vdash} \text{ does not exist} \\ \prod_{i=1}^{d} \left(sl_{q[i]} + e.sl_{mbr[i]}(c_{\dashv})\right) & ift_q \leq 1 - C_m \end{cases} \quad (29)$$

where functions $e.sl_{mbr[i]}(.)$ and $e.sl(.)$ are presented in Eqs. (27) and (28), respectively.

All the components in Eqs. (24) and (26) have been represented as functions of the dimensionality d, the query parameters, the PCR sizes of an object, the dataset cardinality n, and the node fanout f, all of which are readily obtainable. Therefore, we have derived the expected cost of nonfuzzy range search on a regular dataset.

6.2.2 *Arbitrary Data.* The dataset-dependent parameters to the above "regular" cost model involve only the cardinality n, and projection length $o.sl_{pcr[i]}(c)$ of the $o.pcr(c)$ of an object o on the ith dimension ($1 \leq i \leq d$), where c is a U-catalog value. Given an arbitrary dataset, a naive approach of applying the model to estimate $cost(q)$ is to feed those parameters with the average statistics. In particular, $o.sl_{pcr[i]}(c)$ can be set to the average projection length on the ith dimension of the $o'.pcr(c)$ of all objects o' in the dataset. This approach, however, may not produce accurate estimates, due to the potentially large variance in objects' projection lengths.

This problem can be alleviated using the *local smoothing technique* [Theodoridis and Sellis 1996] originally proposed for capturing R-tree performance. The basic observation underlying the technique is that, the variance in the characteristics of objects in a small region is (possibly much) lower than that in the whole workspace. Hence, the application of the regular model to a subset of the dataset tends to be more effective.

Specifically, we divide the data space into a grid of λ^d identical cells, where λ is the *resolution* of the grid, and equals 5 in our experiments. For each cell cl, we will develop a value $cl.cost(q)$ that estimates the expected cost of q, when the

[4]Consider two d-dimensional axis-parallel rectangles whose side lengths on the ith ($1 \leq i \leq d$) dimension equal $x[i]$ and $y[i]$, respectively. Let the two rectangles (independently) uniformly distribute in a workspace where each axis has a unit length. Examine the probability that they intersect into a box whose side length on the ith ($1 \leq i \leq d$) dimension is at least $z[i]$. The analysis of Pagel et al. [1993] shows that the probability equals $\Pi_{i=1}^{d}(x[i] + y[i] - z[i])$. As a corollary, the probability that the two rectangles intersect is $\Pi_{i=1}^{d}(x[i] + y[i])$.

centroid of $q.mbr$ lies in cl. Obviously, λ^d values are obtained after examining all the cells. Our final estimate for the expected cost $cost(q)$ (of all queries) equals the average of these values.

It remains to explain the computation of $cl.cost(q)$. For this purpose, we need the number $cl.n$ of objects o' such that the centroids of their $o'.mbr$ are contained by cl. Furthermore, for these objects o' and every value c in the U-catalog, we compute the average projection length $cl.sl_{pcr[i]}(c)$ on the ith dimension ($1 \leq i \leq d$) of $o'.pcr(c)$. Then, $cl.cost(q)$ is calculated by the regular model in Section 6.2.1, after setting $n = cl.n \cdot \lambda^d$, and $o.sl_{pcr[i]}(c) = cl.sl_{pcr[i]}(c)$. Note that the statistics (i.e., $cl.n$ and $cl.sl_{pcr[i]}(c)$) required for calculating $cost(q)$ can be obtained by a single scan of the dataset.

6.3 Optimizing the U-catalog Size

We close this section by elaborating the procedures of tuning the U-catalog size m. Remember that the cost model of Section 6.2.1 is, in fact, a function m, which influences the query overhead in two important ways. First, m directly decides the node fanout f (which drops if more PCRs are retained for each object). Second, various m leads to different values in the U-catalog, which in turn affect the $c_{\dashv}$ and $c_{\vdash}$ in Eqs. (16), (17), (18), (19), and (29). These formulae determine both the filter and refinement cost.

The number of possibilities of m is limited, because this parameter should be a small integer, for example, no more than 10. Therefore, we use our analytical formulae to predict the expected query cost for every possible value of m, and choose the value that yields the lowest prediction. This strategy, however, raises two questions. First, what are the m values in the U-catalog? Second, how to decide the query parameters (i.e., $sl_{q[1]}, \ldots, sl_{q[d]}$ and t_q) to be plugged into the model?

We settle the first question by fixing the first value to 0, and placing the other $m - 1$ values evenly in the range of $(0, 0.5]$. For example, for $m = 2$, the U-catalog consists of {0, 1/4}, whereas the catalog becomes {0, 1/6, 2/6} for $m = 3$. The answer to the second question largely depends on the preferences of the database administrator. For instance, s/he could manually select some parameter values that are proved to be popular among users based on the past statistics. Another option is to generate many sets of parameters (e.g., one set concerns large r_q and t_q, while another explores their small counterparts), and deploy our model to produce an estimate for every set. The overall quality of the m under consideration can be gauged by the average of the estimates of all sets (possibly assigning different weights to the estimates of various sets).

7. PERFORMANCE EVALUATION

In this section, we empirically evaluate the effectiveness and efficiency of the proposed techniques. All the experiments are performed on a machine running a Pentium IV 3.6GHz CPU. The disk page size is fixed to 4096 bytes. The workspace is normalized to have a domain of [0, 10000] on every dimension.

Given a set X of points, we generate uncertain data as follows to simulate a database storing the positions of mobile clients in a location-based service

[Wolfson et al. 1999]. For each point $p \in X$, we create an uncertain object o, whose uncertainty region $o.ur$ is an L_2 circle that centers at p, and has a radius rad_o. We examine the type of $o.pdf(x)$ that has been experimented most in the literature: Gaussian. A traditional Gaussian distribution, however, has an infinite domain, that is, $o.pdf(x)$ is a positive value for any x in the entire workspace, which contradicts the requirement that $o.pdf(x)$ equals 0 at a point x outside $o.ur$. Hence, following the practice of Cheng et al. [2004b], we consider the "constrained Gaussian" distribution. Formally, let $g(x)$ be a conventional Gaussian function whose mean falls at the centroid of $o.ur$, and its variance equals $(rad_o/2)^2$ (i.e., the standard deviation $rad_o/2$ is half the radius of an object's uncertainty region). Then, the corresponding constrained Gaussian $o.pdf(x)$ is defined as:

$$o.pdf(x) = \begin{cases} g(x)/\int_{x \in o.ur} g(x)\,dx & \text{if } x \in o.ur \\ 0 & \text{otherwise} \end{cases} \tag{30}$$

By setting X respectively to two-dimensional point sets LB, CA, and RAN, we obtain uncertain databases where the centroids of objects' uncertainty regions follow three different distributions. Both LB and CA are real spatial datasets downloadable at the R-tree portal (*http://www.rtreeportal.org*), and are produced from the Tiger project of the US Census Bureau (*http://tiger.census.gov*). Specifically, the former and latter contain 53k and 62k points representing addresses in the Long Beach county and Los Angeles, respectively. RAN consists of 100k points randomly distributed in the workspace. In the sequel, we will use LB-rad_o, CA-rad_o, and RAN-rad_o to refer to the uncertain datasets where the radii of objects' uncertainty regions equal rad_i (e.g., LB-100 indicates the dataset created with $X = LB$ and $rad_o = 100$).

The search region r_q of a nonfuzzy range query is a circle, under the L_∞ or L_2 norm, that has a radius rad_q, and its center follows the distribution of the points in X (i.e., the original dataset used to synthesize uncertain objects). In particular, when the L_∞ norm is used, the search region is a square with side length $2rad_q$. On the other hand, for fuzzy range search, the query object q is randomly sampled from the underlying uncertain dataset. As with nonfuzzy retrieval, a fuzzy query is also associated with a defining norm (L_∞ or L_2), which governs the distance metric in Definition 2.

In our experiments, we will often execute a *workload* of 10000 similar queries in order to measure their average performance. Specifically, a workload has four properties: (i) fuzzy or nonfuzzy, (ii) rad_q (or ε_q) for a nonfuzzy (or fuzzy) workload, (iii) the defining norm, and (iv) t_q. For instance, "a nonfuzzy L_∞ workload with $rad_q = 500$ and $t_q = 0.3$" contains purely nonfuzzy queries whose search regions are squares with side length 1000, and their probability thresholds equal 0.3. We sometimes set t_q to a special value "mixed", to indicate a workload where the probability threshold of a query is randomly generated in the range of [0.1, 0.9].

Table II summarizes the data/query parameters mentioned earlier, together with their values to be tested, and the default values in bold fonts.

Table II. Data and Query Parameters Varied in our Experiments

Parameter	Meaning	Values
rad_o	The radius of an object's uncertainty region	5, **100**, 250
rad_q	The radius of the search region of a nonfuzzy range query	250, **500**, 750
ε_q	The distance threshold of a fuzzy range query	250, **500**, 750
t_q	The probability threshold of a query	**Uniform in [0.1, 0.9]**

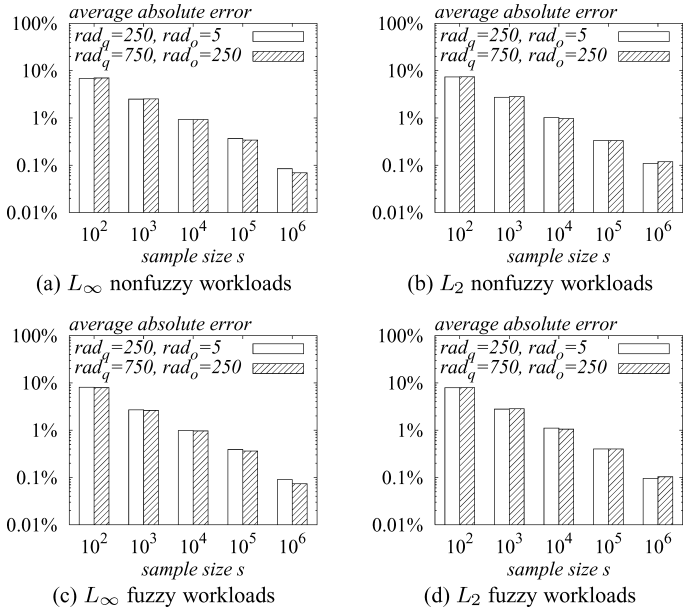
7.1 Cost of Evaluating the Exact Qualification Probability

Given a nonfuzzy query, calculating the qualification probability $Pr_{range}(o, r_q)$ of an object o (Eq. (1)) requires integrating $o.pdf(x)$, as given in Eq. (30), inside the intersection between the uncertainty region $o.ur$ of o and the search area r_q. Remember that $o.ur$ is an L_2 circle, and r_q can be a square or an L_2 circle. In any case, $o.ur \cap r_q$ may have an irregular shape, thus preventing the result of the integral from being solved into a closed form. The same problem also exists in computing the qualification probability $Pr_{fuzzy}(o, q, \varepsilon_q)$ of o (Eq. (3)) with respect to a fuzzy query. In fact, the computation here is even more difficult, because the evaluation of $Pr_{fuzzy}(o, q, \varepsilon_q)$ requires solving $Pr_{range}(.)$ in the first place.

Since $Pr_{range}(o, r_q)$ and $Pr_{fuzzy}(o, q, \varepsilon_q)$ can only be calculated numerically, the result of the calculation cannot be always guaranteed to match the theoretical value. This raises an important question: what should be accepted as a "correct result"? As the numerical process is carried out with the Monte-Carlo method (discussed in Section 5.3), a natural answer to this question is: the result obtained from a sample set with a sufficiently large size s. Therefore, the first set of experiments aims at identifying the magic value s.

Our methodology is as follows. First, we obtain an extremely accurate estimate of the real $Pr_{range}(o, r_q)$ (or $Pr_{fuzzy}(o, q, \varepsilon_q)$), by using a huge sample size 10^{10}. Then, we increase s gradually from a small value, and measure the error obtained from each s against the accurate estimate obtained earlier. Note that, for a nonfuzzy (or fuzzy) query, the error depends on the relative positions of $o.ur$ and r_q (or $q.ur$). Therefore, after fixing an object o, we create a special workload with 1000 random queries satisfying the condition $0 < Pr_{range}(o, r_q)$ (or $Pr_{fuzzy}(o, q, \varepsilon_q)) < 1$ (the qualification probability of o should not be 0 and 1, since numerical evaluation is not needed otherwise). Then, given a particular s, the workload error is measured as the average of the absolute errors of all queries contained.

In Figure 17(a) (or 17(b)), we demonstrate the workload error as a function of s using L_∞ (or L_2) workloads of nonfuzzy queries. Here, we examine two extreme combinations of parameters rad_o and rad_q: small queries on a small object ($rad_q = 250$, $rad_o = 5$), and large queries on a large object ($rad_q = 750$, $rad_o = 250$). Figures 17(c) and 17(d) illustrate the results of similar experiments utilizing workloads of fuzzy queries. It turns out that the precision of Monte-Carlo is dependent solely on the sample size s, and is not affected by the data and query parameters. Recall that the rationale of Monte-Carlo stems from the sampling theory, that is, its accuracy is decided only by two factors: the fraction of samples falling into the integration region, and the function being integrated [Press et al. 2002]. These factors are identical in all the experiments



(a) L_∞ nonfuzzy workloads (b) L_2 nonfuzzy workloads

(c) L_∞ fuzzy workloads (d) L_2 fuzzy workloads

Fig. 17. Error of Monte-Carlo vs. sample size.

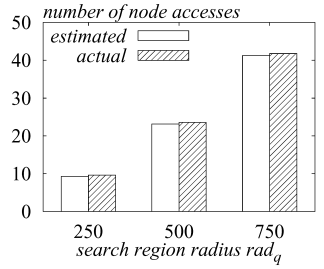
in Figures 17(a) and 17(b) with the same s, which explains the analogous behavior in those figures. Finally, the similarity between nonfuzzy and fuzzy queries is because the calculation of $Pr_{fuzzy}(.)$ is reduced to $Pr_{range}(.)$.

We will set s to 10^4 in the rest experiments, since it is the smallest sample size that leads to a reasonable workload error 1%. In other words, from now on, an approximate result derived from this value of s will be claimed as *correct*. Accordingly, a single evaluation of $Pr_{range}(.)$ and $Pr_{fuzzy}(.)$ demands approximately 1.7 milliseconds and 0.35 seconds, respectively.

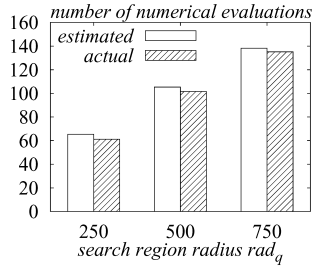
7.2 Tuning the U-catalog Size

The size m of a U-catalog has important influence on query performance. In Section 6, we presented a method for automatic tuning of this parameter, according to the characteristics of the input dataset. In this section, we demonstrate the effectiveness of the method.

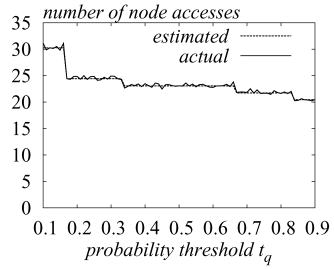
7.2.1 *Cost Model Accuracy.* A fundamental component of our tuning approach is a cost model that predicts the overhead of nonfuzzy range search. To verify the accuracy of the model, we use an $m = 3$ catalog (the values in the catalog are decided as elaborated in Section 6.3), employ the dataset *RAN*-100 (i.e., generated from the point set *RAN* with $rad_o = 100$), and compare the estimated query cost (from our model) against the actual cost. In particular, the comparison includes two aspects: (i) the I/O overhead, which is the cost of the filter step, and proportional to the number of leaf node accesses in the U-tree, and (ii) the CPU time, which is dominated by the overhead of the refinement phase, and proportional to the number of numerical evaluations of $Pr_{range}(.)$.



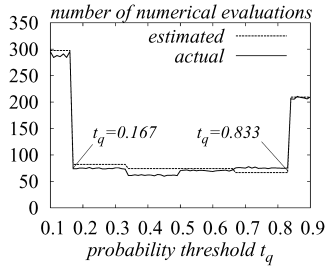
(a) I/O cost vs. rad_o (t_q uniform in [0.1, 0.9])



(b) CPU cost vs. rad_o (t_q uniform in [0.1, 0.9])



(c) I/O cost vs. t_q (rad_q = 500)



(d) CPU cost vs. t_q (rad_q = 500)

Fig. 18. Accuracy of the proposed cost model (dataset: *RAN*-100; L_∞ nonfuzzy workloads).

Figures 18(a) and 18(b) plot the I/O and CPU comparison as a function of the radius rad_q of search regions, respectively. Here, at each value x of rad_q, the actual I/O (or CPU) cost corresponds to the average number of node accesses (or numerical evaluations) in executing a query in a nonfuzzy L_∞ workload with parameters $rad_q = x$ and t_q = mixed. Given a query, we estimate its I/O and CPU overhead using Eqs. (25) and (22), respectively. Figures 18(c) and 18(d) illustrate the comparison as t_q varies from 0.1 to 0.9. The actual and estimated results are obtained in the same manner as explained earlier, except that the workloads have a t_q equal to the value being tested, and an rad_q fixed to 500.

It is clear that the proposed model is highly accurate, incurring an error less than 5% in all cases. In particular, our analytical formulae capture exactly the changes of query cost. First of all, there should be no surprise in witnessing the cost increase continuously with rad_q (see Figures 18(a) and 18(b)): a larger search region leads to more qualifying objects, thus entailing more expensive I/O and CPU overhead. The change behavior with respect to t_q is more complex. As this parameter grows, the I/O cost decreases monotonically, while the number of numerical evaluations initially decreases until t_q reaches 0.167, stays low for a wide range of t_q, and then bounces up when t_q becomes $1 - 0.167 = 0.833$.

To understand the above "t_q-phenomenon", recall that this experiment is based on a U-catalog with three values: 0, 0.167, and 0.334. Let o be an object whose uncertainty region partially intersects the search region (the other objects can always be pruned/validated, and hence, are irrelevant to explaining the phenomenon). The power of Theorem 3 in pruning o is increasingly stronger as t_q grows (see the footnote[5]), which is the reasoning behind Figure 18(c). As

[5]Specifically, for $t_q \in [0, 0.167)$, o can be never be pruned. For $t_q \in [0.167, 0.334)$, o can be pruned

for Figure 17d, we point out that, for $t_q \in [0, 0.167)$, it is not possible to prune o with Theorem 3, although o may be validated using Theorem 2. The opposite is true for $t_q \in (0.833, 1]$, that is, o may be pruned, but it can never be validated. Our heuristics are most effective when t_q distributes in [0.167, 0.833]. In this case, both pruning and validation of o are likely; therefore, the least number of numerical evaluations is needed.

7.2.2 *Effects of the U-catalog Size.* We are ready to inspect the effectiveness of the proposed method for tuning the U-catalog size m. For this purpose, we employ only the uncertain databases generated from the real datasets *CA* and *LB*. As discussed in Section 6.2.2, when the data distribution is irregular, our tuning solution applies the local smoothing technique based on a $\lambda \times \lambda$ histogram; in the sequel, λ is fixed to 5.

We aim at minimizing the expected overall overhead (i.e., including both I/O and CPU time) of a nonfuzzy query whose rad_q equals the median value 500, and its t_q follows a uniform distribution in [0.1, 0.9]. To achieve this goal, given a particular m, we utilize our cost model to estimate the expected overhead of queries with eighty-one $\{rad_q, t_q\}$ combinations, respectively: {500, 0.1}, {500, 0.11},..., {500, 0.89}, {500, 0.9}. Each estimate is the sum of $cost_{frn}$ and $cost_{flt}$ computed from Eqs. (26) and (24), respectively, setting $cost_{ranIO}$ to 20 milliseconds, and $cost_{rfn}$ to 1.7 milliseconds (according to the experiments in Section 7.1). The *penalty* of m is the average of the estimates of all the combinations. The best m is the one with the lowest penalty.

In the experiment of Figure 19(a), we select the uncertain dataset *CA*-5 (where the uncertainty region of each object is a circle with radius $rad_o = 5$). The curve labeled as "actual" presents the average query cost in a nonfuzzy L_∞ workload with $rad_q = 500$ and $t_q =$ mixed, when the catalog size m varies from 1 to 10. The curve "estimated" shows the penalties of m. Figures 19(b) through 19(f) demonstrate similar results for datasets *LB*-5, *CA*-100, *LB*-100, *CA*-250, and *LB*-250, respectively.

In every figure, the two curves are very close to each other, which proves that our performance analysis is effective also for irregular data distributions. Furthermore, the optimal U-catalog size m is clearly related to the data characteristics. In particular, when objects have very small uncertainty regions (as in *CA*-5 and *LB*-5), the best m equals 1, that is, only a single PCR (i.e., the MBR $o.pcr(0)$) of each object o should be indexed. This is reasonable because, if the query region r_q is much larger than $o.ur$, the chance of r_q partially intersecting $o.pcr(0)$ is very low, meaning that o can already be pruned or validated with a very high probability even if no other PCR is available.

As shown in Figures 19(c) through 19(d), when objects have sizable uncertainty regions, the optimal m tends to increase with rad_o. For each dataset, before m reaches its optimal value, enlarging the U-catalog brings more PCRs to each object, strengthens the pruning and validating power of our heuristics,

---

if the query region r_q is disjoint with $o.pcr(0.167)$, while for $t_q \in [0.334, 0.666)$, pruning can be performed if r_q does not intersect $o.pcr(0.334)$, which is smaller than $o.pcr(0.167)$. For $t_q \in [0.666, 0.833)$ (or [0.666, 0.833)), we can prune o if r_q does not contain $o.pcr(0.334)$ (or $o.pcr(0.167)$).

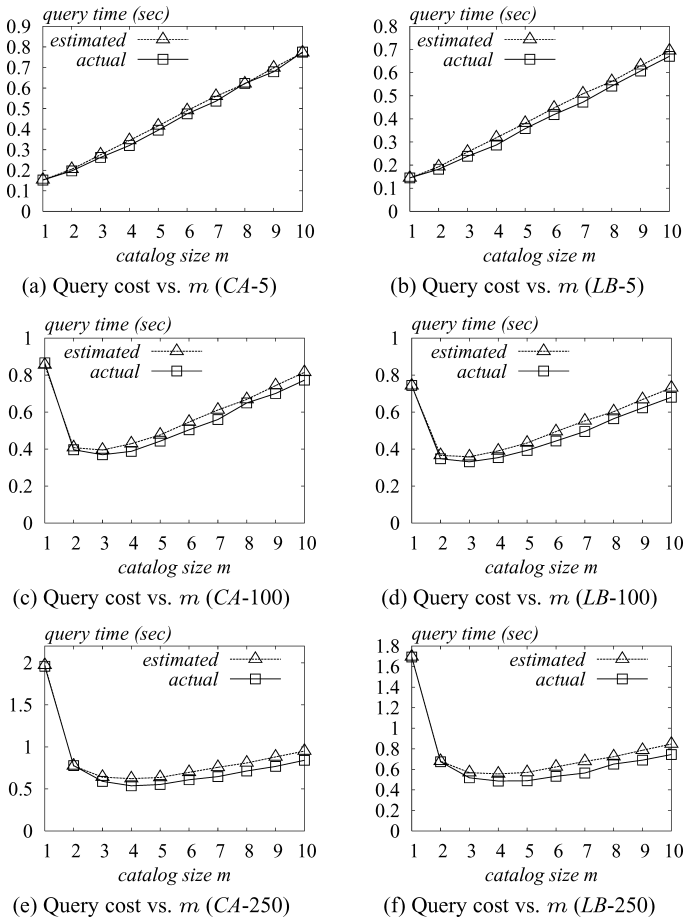
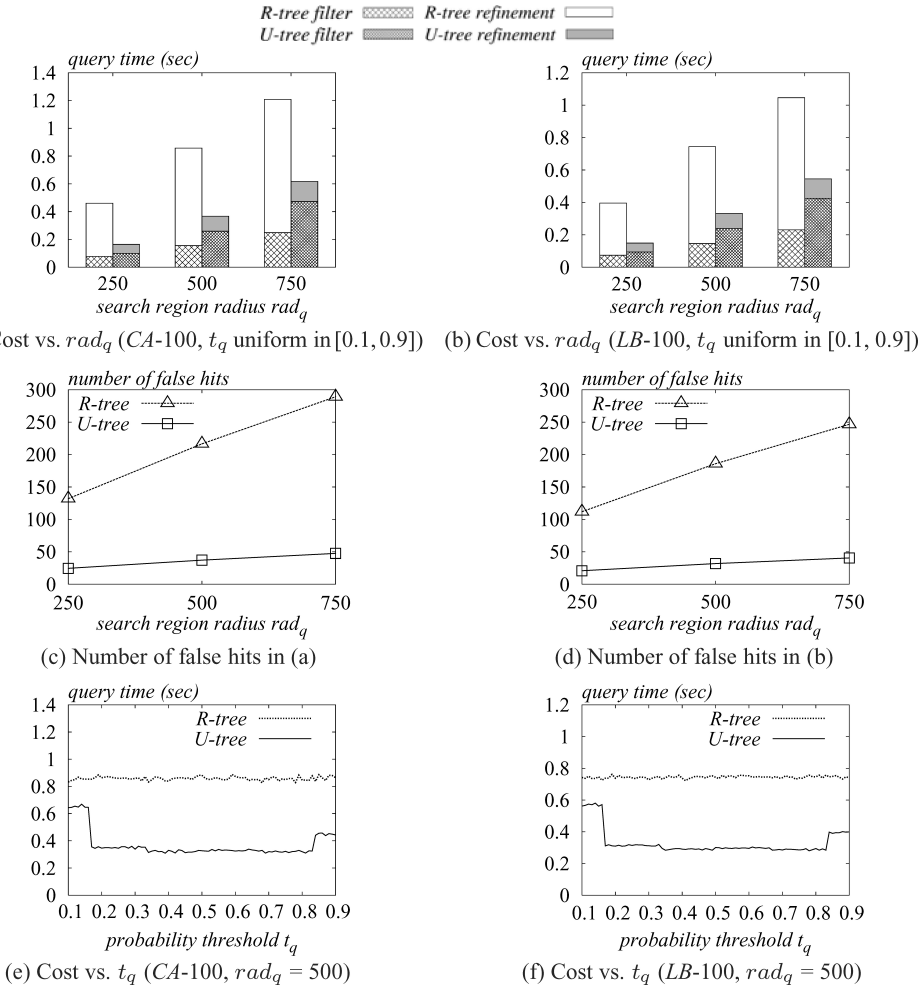


Fig. 19. Effectiveness of our method for U-catalog size tuning ($rad_q = 500$, t_q uniform in [0.1, 0.9]).

and reduces the query cost. After m passes the optimum, however, further increasing it no longer enhances the effectiveness of the heuristics significantly, but necessitates more I/Os (due to the decrease of node fanout), thus compromising query performance. Our tuning method captures such behavior precisely, and always identifies the optimal catalog size.

7.3 Cost of Nonfuzzy Range Search

Range search on multidimensional uncertain data is a novel topic that has not been previously studied. Since there does not exist a nontrivial competitor, next we compare the U-tree against the R-tree in nonfuzzy retrieval. Specifically, an R-tree refers to a special U-tree with a U-catalog size $m = 1$. We will focus on two uncertain datasets: CA-100 and LB-100, for both of which the best U-catalog size equals 3, as shown in Figure 19. A memory cache is introduced to buffer all the intermediate levels of an index.



(a) Cost vs. rad_q (*CA*-100, t_q uniform in $[0.1, 0.9]$)
(b) Cost vs. rad_q (*LB*-100, t_q uniform in [0.1, 0.9])
(c) Number of false hits in (a)
(d) Number of false hits in (b)
(e) Cost vs. t_q (*CA*-100, rad_q = 500)
(f) Cost vs. t_q (*LB*-100, rad_q = 500)

Fig. 20. Nonfuzzy query cost comparison between R- and U-trees (L_∞-workloads).

Focusing on *CA*-100, the experiment of Figure 20(a) uses three nonfuzzy L_∞ workloads with t_q = mixed, and rad_q = 250, 500, and 750, respectively. For each workload, we measure the average overhead of processing a query with R- and U-trees, respectively. At each value of rad_q, the result of a method is further broken into two parts, representing the cost of its filter and refinement steps, respectively. Figure 20(b) shows the comparison for *LB*-100 under the same settings.

The U-tree outperforms its rival in all cases, achieving a maximum speedup of 3. In particular, the U-tree entails higher filter cost, but is significantly faster in the refinement phase. This is expected because (i) by retaining several PCRs per object, the U-tree has a lower fanout, and hence, a larger number of nodes, rendering more node accesses in filtering; (ii) the U-tree needs to refine much fewer objects.

An object is a *false hit*, if its precise qualification probability is calculated, but it does not qualify the query. False hits should be avoided as much as possible to ensure fast response time. Figures 20(c) and 20(d) demonstrate the average number of false hits per query in the experiments of Figures 20(a) and 20(b), respectively. Obviously, the U-tree incurs significantly fewer false hits.

Figure 20(e) and 20(f)) plots the query time of R- and U-trees as a function of t_q, using nonfuzzy L_∞ workloads with $rad_q = 500$, and $t_q = 0.1,\ldots, 0.9$, respectively. Again, the U-tree is the clear winner, and its behavior is similar to that illustrated in Figure 18(d). The performance of the R-tree is not affected by t_q, since keeping only the objects' MBRs offers equivalent pruning/validating power for all t_q.

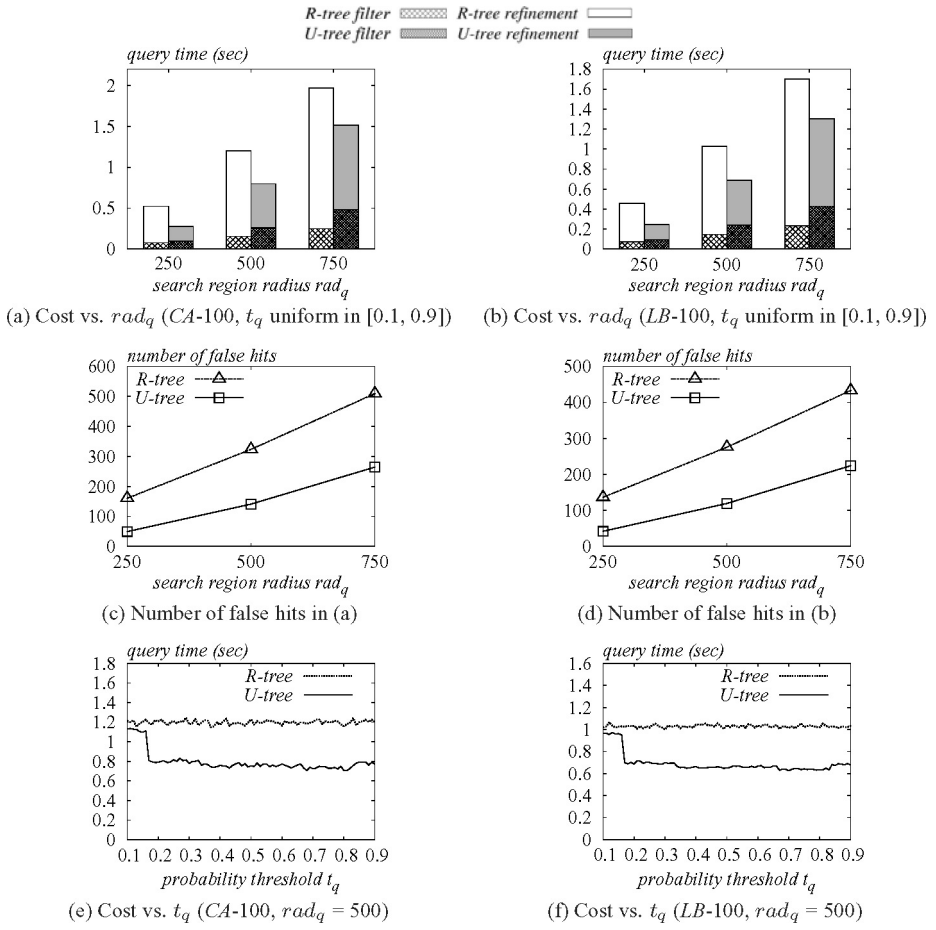
Figure 21 presents the results of the same experiments in Figure 20 but with respect to L_2 workloads. These results confirm the phenomena observed from L_∞ workloads, except that U-trees achieve a lower performance speedup over R-trees. To explain this, recall that we process an L_2 query, by conservatively bounding its search region r_q using an outside rectangle r and an inside rectangle r' respectively, as illustrated in Figure 6(b). The conservative approach reduces the pruning/validating power of our heuristics. Although both U- and R-trees are affected, the effects on U-trees are more significant, since only limited pruning/validating is possible for R-trees in any case.

7.4 Cost of Fuzzy Search

Now we continue to evaluate the algorithm in Section 4 for fuzzy range queries, also using the datasets *CA*-100 and *LB*-100. As mentioned in Section 4.2, the algorithm requires a parameter m_q, which is the number of PCRs computed for the query object q. Hence, we first select an appropriate value for this parameter. For this purpose, given a particular m_q, we measure the average time of processing a query with a U-tree in a fuzzy workload with $\varepsilon_q = 500$ and $t_q =$ mixed. Figure 22(a) and (22(b)) plots the average cost as a function of m_q for both *CA*-100 and *LB*-100, using L_∞ (L_2) workloads. Clearly, an excessively small m_q results in expensive query overhead, because in this case only a limited amount of information about the query is available for pruning and validating. On the other hand, once m_q reaches 10, further increasing this parameter does not lead to significant improvement, indicating that 10 PCRs of q is already sufficient for efficient processing. In the rest experiments, we fix m_q to 10.

Next, we compare the query performance of R- and U-trees, by repeating the experiments of Figures 20 and 21 with respect to fuzzy workloads. The results are presented in Figures 23 and 24. Each column can be interpreted as either the overall query time or refinement overhead (averaged over all the queries in a workload). In fuzzy search, the filter step cost is negligible compared to the overall time, and is demonstrated on top of each column.

The U-tree is again the better solution in all experiments, having a maximum speedup of 5 over the R-tree (in Figure 23(c)). As in Figures 23 and 24, the cost changes of U-trees with respect to t_q are much smoother than those in nonfuzzy search (see Figures 20 and 21). Namely, the "t_q-phenomenon", defined



(a) Cost vs. rad_q (CA-100, t_q uniform in [0.1, 0.9])
(b) Cost vs. rad_q (LB-100, t_q uniform in [0.1, 0.9])
(c) Number of false hits in (a)
(d) Number of false hits in (b)
(e) Cost vs. t_q (CA-100, rad_q = 500)
(f) Cost vs. t_q (LB-100, rad_q = 500)

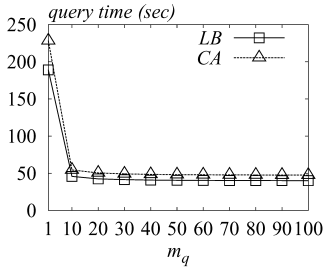
Fig. 21. Nonfuzzy query cost comparison between R- and U-trees (L_2-workloads).

in Section 7.2.1, disappears. To understand this, recall that, as elaborated in Section 7.2.1, the condition of the phenomenon is that, for a nonfuzzy query q, only one PCR (selected according to t_q) is used for pruning/validating an object. The condition no longer holds: given a fuzzy query, multiple PCRs may be utilized by our pruning/validating approach in Section 4.

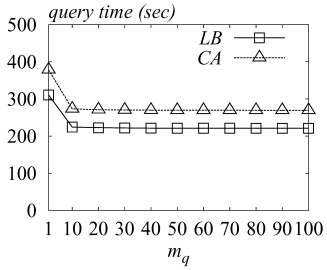
Finally, as in the nonfuzzy scenario, the performance superiority of U-trees over R-trees is more obvious in L_∞ queries (than L_2). Similar to the reasons given in Section 7.3, this is due to the conservative approximation deployed in processing an L_2 query, except that here the approximation is illustrated in Figures 9(c) and 9(d).

7.5 Index Construction Overhead

We proceed to evaluate the efficiency of the U-tree's construction algorithm. In Figure 25(a) and (25(b)), we demonstrate the cost of building a U- and an R-tree on dataset *CA-100* (*LB*-100), by incrementally inserting all the objects.

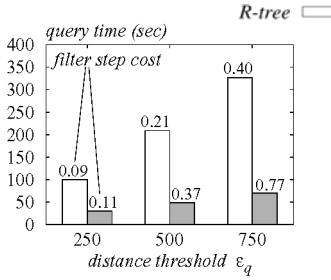


(a) U-tree query cost vs. m_q (L_∞ workloads)

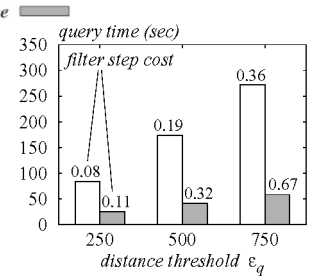


(b) U-tree query cost vs. m_q (L_2 workloads)

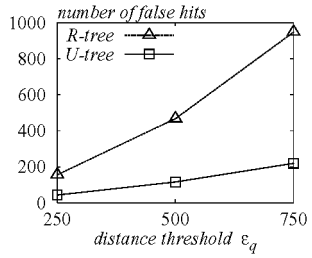
Fig. 22. Tuning m_q for fuzzy retrieval ($\varepsilon_q = 500$, t_q uniform in [0.1, 0.9]).



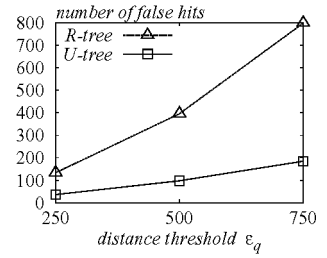
(a) Cost vs. ε_q (CA-100, t_q uniform in [0.1, 0.9])



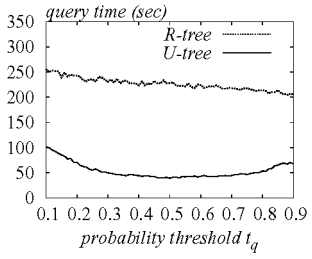
(b) Cost vs. ε_q (LB-100, t_q uniform in [0.1, 0.9])



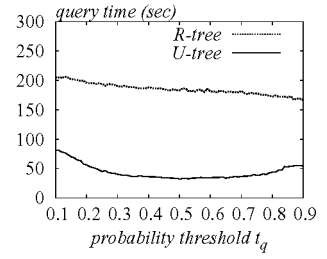
(c) Number of false hits in (a)



(d) Number of false hits in (b)

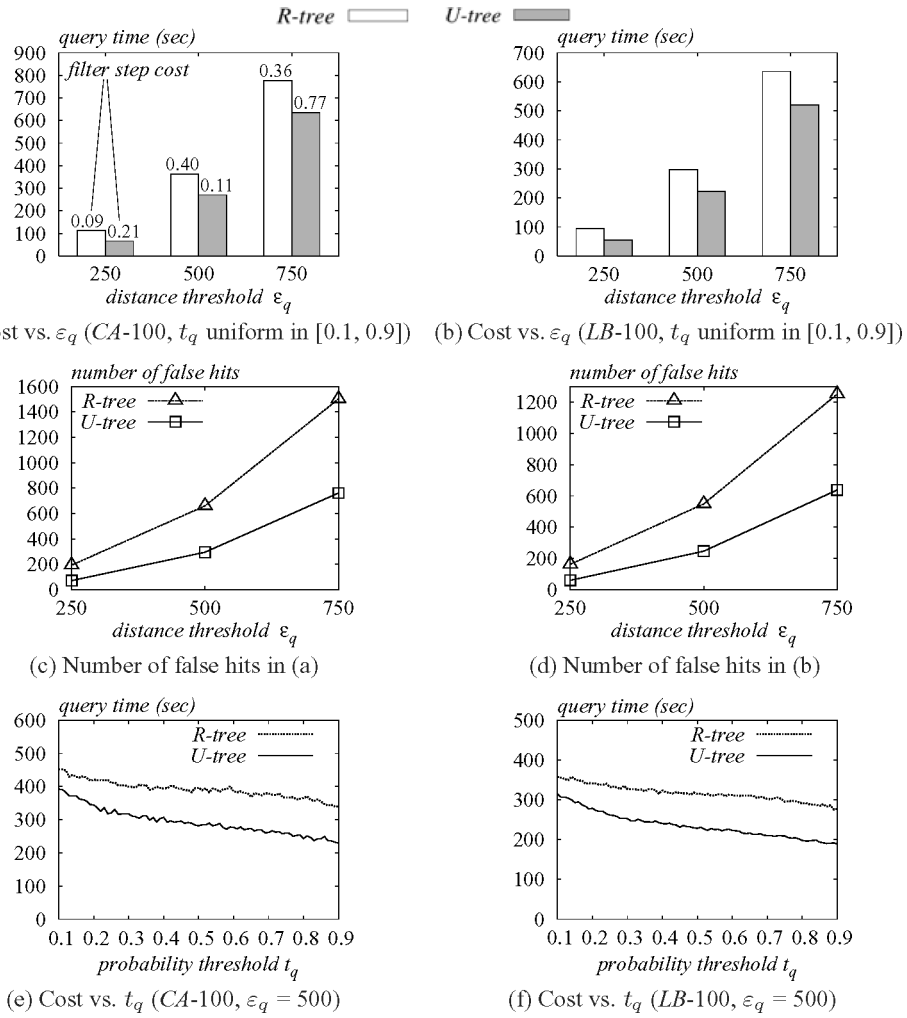


(3) Cost vs. t_q (CA-100, ε_q = 500)



(f) Cost vs. t_q (LB-100, ε_q = 500)

Fig. 23. Fuzzy query cost comparison between R- and U-trees (L_∞-workloads).



(a) Cost vs. ε_q (CA-100, t_q uniform in [0.1, 0.9]) (b) Cost vs. ε_q (LB-100, t_q uniform in [0.1, 0.9])

(c) Number of false hits in (a) (d) Number of false hits in (b)

(e) Cost vs. t_q (CA-100, $\varepsilon_q = 500$) (f) Cost vs. t_q (LB-100, $\varepsilon_q = 500$)

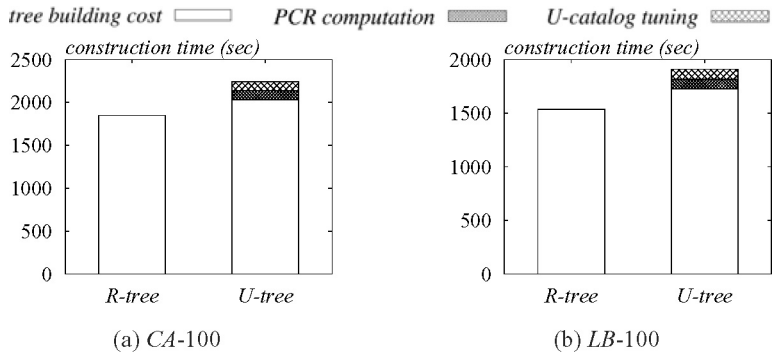
Fig. 24. Fuzzy query cost comparison between R- and U-trees (L_2-workloads).

In particular, the U-tree result consists of three components, capturing respectively the overhead of (i) optimizing the U-catalog size, (ii) preparing the PCRs of each object, and (iii) incremental insertion. The U-tree catalog contains 3 values.

Evidently, the cost of U-catalog optimization and PCR computation accounts for a very small fraction of the overall overhead (particularly, finding all the PCRs of an object takes around 1.8 milliseconds). After those two tasks are completed, a U-tree can be built in time similar to an R-tree: on average 30 milliseconds to insert one object.

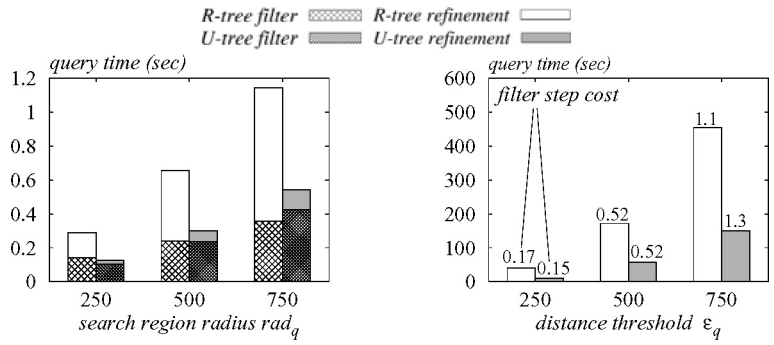
7.6 Three-Dimensional Results

So far, we have focused on 2D data. The last set of experiments examines the performance of U-trees on three-dimensional objects. For this purpose, we generate



(a) *CA*-100 (b) *LB*-100

Fig. 25. Index construction cost.



(a) Nonfuzzy search (b) Fuzzy search

Fig. 26. Query cost on three dimensional data (*RAN-3D*, L_∞-workloads, t_q uniform in [0.1, 0.9]).

a dataset *RAN-3D* in the same way as *RAN*-250, except that each object's uncertainty region is a 3D sphere (with radius 250), and the function $g(x)$ in Eq. (30) is the pdf of a three-variate normal distribution with standard deviation 125. The U-tree on *RAN-3D* has a U-catalog size 3. A (nonfuzzy/fuzzy) query workload is created in the same manner as a 2D counterpart.

Figure 26(a) and (26(b)) compares the cost of answering a nonfuzzy (fuzzy) workload with the U- and R-trees on *RAN-3D*, when rad_q (ϵ_q) changes from 250 to 750. Each cost is broken down into the overhead of the filter and refinement steps, respectively. Since the filter costs are unnoticeable in Figure 26(b), we illustrate them on top of the columns. The U-tree consistently outperforms significantly the R-tree in all cases.

8. RELATED WORK

Section 8.1 first surveys various approaches for modeling uncertainty. Then, Section 8.2 discusses query algorithms for producing probabilistic results.

8.1 Uncertainty Models

Several models have been proposed to incorporate uncertain objects in databases. These models differ mainly in the semantics and complexities of

the data in the underlying applications [Sarma et al. 2006]. In general, uncertainty can be represented in a "qualitative" or "quantitative" manner. A qualitative model captures the presence/absence of data, typically using a "NULL" keyword to describe a missing value. Accordingly, SQL needs to be augmented with additional keywords for querying such incomplete tuples, for example, "definite", "indefinite", "maybe" and "must" [Liu and Sunderraman 1987, 1991; Sistla et al. 1997].

To provide a more rigorous treatment of uncertain data, a quantitative approach describes uncertainty through mathematical modeling. These approaches include the fuzzy model [Galindo et al. 2006], the Dempster–Shafer (evidence-oriented) model [Lee 1992; Lim et al. 1996] and the probabilistic model. In particular, the probabilistic model can be further classified into three categories: "table-based", "tuple-based" and "attribute-based" solutions, which handle different granularities of uncertainty. Specifically, a table-based approach concerns the "coverage" of a table, that is, how much percentage of tuples are present in a table [Widom 2005]. A tuple-based solution, on the other hand, associates each individual tuple with a probability, which indicates the likelihood that the tuple exists in the table [Dalvi and Suciu 2004; Dalvi and Suciu 2005; Fuhr 1995]. This methodology has been applied to various forms of semistructured data, such as XML documents [Nierman and Jagadish 2002] and other acyclic graphs [Hung et al. 2003]. Finally, when an attribute of a tuple is not known precisely, an attribute-based method introduces a probability distribution for describing a set of possible values, together with their occurring probabilities [Cheng et al. 2003; Deshpande et al. 2004; Pfoser and Jensen 1999; Wolfson et al. 1999].

The attribute-based category, which is the focus of this article, has received a large amount of attention in the literature of spatiotemporal databases and sensor networks. For example, the modeling of vehicle locations illustrated in Figure 1(a) is due to Wolfson et al. [1999]. This model is extended by Pfoser and Jensen [1999] to enable estimation of the modeling error, by Trajcevski et al. [2004] to support trajectories, and by Teixeira de Almeida and Güting [2005] to road networks. A one-dimensional version of the model of Wolfson et al. [1999] is also employed to handle continuous sensor data in Cheng et al. [2003, 2006a]. In a similar context [Deshpande et al. 2004], a joint pdf of multiple attributes is deployed to capture the correlation of physical entities (e.g., temperature and pressure). The above work concentrates on continuous attributes, whereas uncertainty of discrete attributes is discussed in Barbará et al. [1992] and Lakshmanan et al. [1997]. The solutions developed in our article can be applied to all the models mentioned earlier.

Finally, there is a bulk of research [Cheng et al. 2003, 2006a; Khanna and Tan 2001; Olston et al. 2001; Olston and Widom 2000, 2002] that investigates how to reduce the cost of monitoring objects' uncertain representations (e.g., as temperature is rising, the sensor must decide whether to issue an update to the server, taking into account the tradeoff between communication overhead and the precision of modeling). The approaches there are complementary to our work, because they can be applied to generate objects' pdf updates to the U-tree.

8.2 Query Evaluation

In a broad sense, a "probabilistic query" is a user inquiry that retrieves the objects qualifying a set of predicates with certain probabilistic guarantees. Such queries are usually raised against a tuple-based or attribute-based uncertainty model. In particular, queries with respect to tuple-based modeling are formulated through the notion of either "intensional semantics" [Fuhr 1995] or "extensional semantics" [Dalvi and Suciu 2004, 2005]. For the attribute-based category, [Cheng et al. 2003, 2006a] present a detailed taxonomy that classifies a variety of probabilistic search, based on factors such as whether the result values are continuous or discrete, whether there is any relationship among the retrieved objects, and so on. Cheng et al. [2003, 2006a] also develop algorithms for evaluating queries of each class in the taxonomy. These algorithms are later adapted to solve problems in spatiotemporal databases [Cheng et al. 2004a], and sensor networks [Cheng et al. 2006a; Deshpande et al. 2004; Han et al. 2007]. Recently, join operations between two uncertain datasets are investigated in Kriegel et al. [2006].

In practice, the above methods may incur expensive cost, since they must compute the actual qualification probability of every object. Motivated by this, (targeting attribute-based modeling), Cheng et al. [2004b] introduce the concept of "probability thresholding", as formally defined in Section 2. In Cheng et al. [2004b], the authors also explore access methods that minimize the I/O cost of one dimensional probability threshold range search. They argue that uncertain databases are inherently more difficult to handle (than the precise counterpart), and support their claim by proving an asymptotical lower bound for the optimal I/O performance. They also develop several index structures that (almost) achieve the lower bound, but, unfortunately, are limited to one-dimensional spaces. In the preliminary version [Tao et al. 2005] of the current article, we tackle multidimensional data with the basic version of the heuristics in Section 3, and describe a compression-based implementation of the U-tree.

The techniques proposed in this article extend beyond the methods in Cheng et al. [2004b] and Tao et al. [2005]. Specifically, we (i) present a thorough set of heuristics for pruning and validation of nonqualifying and qualifying objects, respectively, (ii) perform a careful theoretical analysis to prove the effectiveness of those heuristics, and (iii) devise fast algorithms for fuzzy range search, which is not addressed in Cheng et al. [2004b] and Tao et al. [2005].

9. CONCLUSIONS AND FUTURE WORK

As has been proved in spatial databases, range search is a problem fundamental to a large number of analytical tasks [Gaede and Gunther 1998]. Unfortunately, there has been no formal research about optimizing this operation on multidimensional uncertain objects, thus currently preventing such data from being manipulated and analyzed efficiently. This article alleviates the situation by presenting a comprehensive study on two forms of range retrieval common in practice: *nonfuzzy* and *fuzzy* search. Based on a novel concept of "probabilistically constrained rectangle" (PCR), we carefully developed a set of heuristics for effectively pruning (or validating) nonqualifying (or qualifying) objects. PCR

also motivates a new index structure called the U-tree for minimizing the I/O overhead of range queries. Finally, we accompany our algorithmic findings with a thorough performance analysis, which explains the reasoning behind the efficiency of the proposed techniques, and leads to the development of a cost model that can be applied to query optimization.

Query processing on multidimensional uncertain databases, in general, remains an attractive research topic that has not been extensively explored. The work of this article lays down the foundation for designing fast algorithms towards accomplishing various data mining goals, such as clustering, outlier detection, etc (investigation of these algorithms would very likely motivate alternative access methods, which, in turn, may actually inspire improvement of the U-tree). A challenge, however, lies in the semantics/representations of the mined results. For example, what does a "cluster of uncertain objects" mean exactly? How to store a cluster concisely without losing its semantics? As another example, what is a (global/local) "outlier" in a dataset when each object is described with a pdf? Answers to these questions would naturally spawn new, practical, definitions of the existing data mining concepts.

APPENDIX: PROOFS OF LEMMAS AND THEOREMS

PROOF OF THEOREM 1. We will only prove Rule (1), because Rule (2) can be established in a similar manner. Since r_q does not fully contain $o.pcr(1-t_q)$, there must be a face of $o.pcr(1-t_q)$ such that, both $o.pcr(1-t_q)$ and r_q lie on the same side of the d-dimensional plane containing the face. Let us denote the place as l. Consider the portion of $o.ur$ that lies on the opposite side of l with respect to r_q. By the definition of $o.pcr(1-t_q)$, o has probability $1-t_q$ to appear in that portion. As a result, the probability that o falls in r_q is smaller than $1-(1-t_q)=t_q$. Thus, o can be safely pruned. □

PROOF OF THEOREM 2. To prove Rule (1), we aim at obtaining a rectangle r which is completely covered by r_q. The rectangle r has the property that o has at least $1-\sum_{i=1}^{d-l}(c_i+c_i')$ probability to appear in r, which, combined with the fact $1-\sum_{i=1}^{d-l}(c_i+c_i')\geq t_q$, confirms that o indeed qualifies q. More specifically, at the beginning, r is initialized to $o.pcr(0)$. Then, we will shrink r along each of the $d-l$ dimensions on which $o.pcr(0)$ is not covered by r_q. During the whole process, we will use ρ to denote a lower bound for the probability that o appears in the current r. The starting value of ρ is 1, and its final value will be exactly $1-\sum_{i=1}^{d-l}(c_i+c_i')$.

On dimension 1, we shrink r by moving its left edge to the left boundary of $o.pcr(c_1)$. Compared to the r before the shrinking, the appearance probability of o inside r has been reduced by at most c_1, due to the formulation of $o.pcr(c_1)$. Hence, with respect to the new r, the value of ρ can be updated to $1-c_1$. Similarly, we shrink r again by moving its right edge (still, on dimension 1) to the right boundary of $o.pcr(c_1')$, and update ρ to $1-c_1-c_1'$.

Performing the above shrinking on all the dimensions $1, 2, \ldots, d-l$, we end up with (i) a rectangle r whose left (or right) edge along the ith dimension $(1\leq i\leq d-l)$ coincides with that of $o.pcr(c_i)$ (or $o.pcr(c_i')$), and (ii) a ρ with value

$1 - \sum_{i=1}^{d-l}(c_i + c'_i)$. Hence, the projections of r on all these $d - l$ dimensions are contained by those of r_q. Furthermore, r is covered by r_q along the remaining l dimensions; therefore, we have discovered an r and a ρ needed for proving the first rule, as stated at the beginning of the proof.

Finally, Rule (2) can be established in a similar, but simpler, way. Due to symmetry, let us consider the case where $[r_{q[1]-}, r_{q[1]+}]$ encloses $[o.pcr_{[1]-}(c_1), o.pcr_{[1]-}(c'_1)]$. Initially, r equals a rectangle that shares the same extents as $o.mbr$ on all the dimensions except the first one, along which r has a projection $[o.pcr_{[1]-}(0), o.pcr_{[1]-}(c'_1)]$. We set ρ to c'_1, which is the exact probability that o appears in r. Then, we shrink r on the first dimension, by moving its left edge to the left boundary of $o.pcr(c_1)$. Accordingly, ρ can be updated to $c'_1 - c_1$. The current r is contained in r_q; hence, Rule (2) holds. □

Proof of Theorem 3. Let us first establish Rule (1). Since r_q does not fully cover $o.pcr(c_{\vdash})$, by Rule (1) of Theorem 1, o does not qualify q if its probability threshold t_q were $1 - c_{\vdash}$. In fact, $c_{\vdash} \geq 1 - t_q$, that is, the actual t_q is at least $1 - c_{\vdash}$; therefore, o can be safely eliminated. Rule (2) can be verified in a similar manner. Specifically, since r_q is disjoint with $o.pcr(c_{\dashv})$, according to Rule (2) of Theorem 1, o does not satisfy q even if its t_q were $c_{\dashv}$, which is at most the actual t_q. Hence, o can again be pruned. □

Proof of Lemma 1. The lemma trivially holds if r_q contains or is disjoint with $o.mbr$. In the sequel, we discuss the case where r_q partially overlaps $o.mbr$, starting with the pruning part of the lemma.

Pruning Case 1: $t_q \leq 1 - C_m$. In this scenario, the value $c_{\leq}$ at Line 4 of Algorithm 1 must exist; otherwise, $UB_{range}(o, r_q)$ would be decided at Line 8, and larger than $1 - C_m$, violating the condition $t_q > UB_{range}(o, r_q)$. Hence, $UB_{range}(o, r_q) = c_{\leq} - \delta$ (where δ is an infinitely small positive), leading to $c_{\leq} \leq t_q$. Let $c_{\dashv}$ be the largest value in the U-catalog that is at most t_q. It follows that $c_{\leq} \leq c_{\dashv}$, that is, $o.pcr(c_{\leq})$ contains $o.pcr(c_{\dashv})$. By the way $c_{\leq}$ is decided, $o.pcr(c_{\leq})$ is disjoint with r_q. Therefore, $o.pcr(c_{\dashv})$ is also disjoint with r_q, so that o is pruned by Rule (2) of Theorem 3.

Pruning Case 2: $t_q > 1 - C_m$. Assume, on the contrary, that o cannot be pruned by Theorem 3. Thus, r_q fully covers $o.pcr(c_{\vdash})$, where $c_{\vdash}$ is the smallest value in the U-catalog at least $1 - t_q$; otherwise, o would have been eliminated by Rule (1) of Theorem 3. Therefore, r_q definitely contains $o.pcr(C_m)$, where C_m is the largest value in the U-catalog. It follows that $c_{\leq}$ does not exist at Line 4. Let us examine the $c_{\geq}$ produced at Line 7. As $t_q > UB_{range}(o, r_q) = 1 - c_{\geq} - \delta$, we have $c_{\geq} \geq 1 - t_q$. Hence, $c_{\geq} \geq c_{\vdash}$ (recall the criterion of choosing $c_{\geq}$), and $o.pcr(c_{\vdash})$ contains $o.pcr(c_{\geq})$. Since, due to the way $c_{\geq}$ is selected, r_q does not fully cover $o.pcr(c_{\geq})$, r_q cannot enclose $o.pcr(c_{\vdash})$, either. Here, we arrive at a contradiction.

We proceed with the validating part of the lemma, also considering two cases.

Validating Case 1: $LB_{range}(o, r_q)$ produced at Line 15. Let us apply the c_1, c'_1, ..., c_{d-l}, c'_{d-l} calculated at Lines 13 and 14 in Rule (1) of Theorem 2 (the projection of r_q does not contain that of $o.mbr$ on dimensions 1, ..., $d - l$). By

the way these $2(d-l)$ values are decided, the projection of r_q on each dimension $i \in [1, d-l]$ encloses $[o.pcr_{[i]-}(c_i), o.pcr_{[i]+}(c_i')]$. Since $t_q \leq LB_{range}(o, r_q) = 1 - \sum_{i=1}^{d-l}(c_i + c_i')$, o is validated by Rule (1).

Validating Case 2: $LB_{range}(o, r_q)$ produced at Line 19. In this scenario, $l = d-1$. We apply the values of c_1 and c_1' computed at Lines 17 and 18 in Rule (2) of Theorem 2. According to the manner these two values are selected, the projection of r_q on dimension 1 encloses either $[o.pcr_{[1]-}(c_1), o.pcr_{[1]-}(c_1')]$ or $[o.pcr_{[1]+}(c_1'), o.pcr_{[1]+}(c_1)]$. In both situations, as $t_q \leq c_1' - c_1$, o is validated by Rule (2). □

Proof of Lemma 2. The lemma is obviously true if r_q completely covers or is disjoint with $o.mbr$. In the sequel, we focus on the case where r_q partially overlaps $o.mbr$. To prove the part of the lemma about pruning, assume, on the contrary, that pruning with Theorem 3 is possible for a $t_q \leq UB_{range}(o, r_q)$. We discuss two cases separately.

Pruning Case 1: $UB_{range}(o, r_q)$ produced at Line 5 of Algorithm 1. Accordingly, $t_q \leq UB_{range}(o, r_q) < C_m$; hence, only Rule (2) of Theorem 3 could have eliminated o, meaning that r_q is disjoint with $o.pcr(c_\dashv)$, where $c_\dashv$ is the largest U-catalog value at most t_q. Let $c_\leq$ be the value computed at Line 4, that is, $c_\leq = UB_{range}(o, r_q) + \delta$, where δ is an infinitely small positive. Since $c_\dashv \leq t_q < c_\leq$, $c_\leq$ is no longer the smallest value c in the U-catalog such that $o.pcr(c)$ is disjoint with r_q, which violates the definition of $c_\leq$.

Pruning Case 2: $UB_{range}(o, r_q)$ produced at Line 8. Let $c_\geq$ be the value computed at Line 7, that is, $c_\geq = 1 - UB_{range}(o, r_q) - \delta < 1 - t_q$. As Line 7 has been executed, all $o.pcr(c)$ (for any c in the U-catalog) must intersect r_q. Thus, o can be pruned only by Rule (1) of Theorem 3, meaning that r_q does not fully cover $o.pcr(c_\vdash)$, where $c_\vdash$ is a U-catalog value at least $1 - t_q$. Since $c_\vdash \geq 1 - t_q > c_\geq$, $c_\geq$ is no longer the largest value c in the U-catalog such that r_q does not fully cover $o.pcr(c)$. This violates the definition of c.

We continue to prove the part of the lemma about validating. Assume, on the contrary, that validating with Theorem 2 is possible for a $t_q > LB_{range}(o, r_q)$. We again distinguishing two cases.

Validating Case 1: o is validated by Rule (1) of Theorem 2. In this scenario, r_q encloses $o.pcr(C_m)$, and thus, $LB_{range}(o, r_q)$ is determined at Line 15. Let $c_1, c_1', \ldots, c_{d-l}, c_{d-l}'$ be the $2(d-l)$ values computed at Lines 13 and 14. Similarly, we use $c_1^*, c_1^{*'}, \ldots, c_{d-l}^*, c_{d-l}^{*'}$ to denote the values used in Rule (1) for validating o. For every $i \in [1, d-l]$, where l is as defined at Line 10, we have $c_i \leq c_i^*$ and $c_i' \leq c_i^{*'}$, due to the way that c_i and c_i' are selected. Therefore, $LB_{range}(o, r_q) = 1 - \sum_{i=1}^{d-l}(c_i + c_i') \geq 1 - \sum_{i=1}^{d-l}(c_i^* + c_i^{*'}) \geq t_q$, leading to a contradiction.

Validating Case 2: o is validated by Rule (2) of Theorem 2. When this happens, r_q does not contain $o.pcr(C_m)$ (otherwise, it is easy to observe that Rule (1) can also validate o); hence, $LB_{range}(o, r_q)$ is obtained at Line 19. Let c_1, c_1' be the values computed at Lines 17, 18, and $c_1^*, c_1^{*'}$ the values used in Rule (2) for validating o. By the way that c_1 and c_1' are chosen, we have $c_1' \geq c_1^{*'}$ and $c_1 \leq c_1'$. Therefore, $LB_{range}(o, r_q) = c_1' - c_1 \geq c_1^{*'} - c_1^* \geq t_q$, resulting in a contradiction. □

PROOF OF THEOREM 4. The theorem is a direct corollary of the definitions of PCRs. □

PROOF OF THEOREM 5. Same as the proof of Theorem 2, except that, in the part establishing Rule (1), 'l' should be replaced with 'd', while, in the part about Rule (2), "dimension 1" is now "dimension i". □

PROOF OF LEMMA 3. Inequality (12) follows immediately the definition of Pr_{range} and the fact that $\sqcap(r, \varepsilon_q) \subseteq \odot(x, \varepsilon_q) \subseteq \sqcup(r, \varepsilon_q)$. □

PROOF OF LEMMA 4. Due to symmetry, it suffices to prove only Inequality (13). Given $r_1, \ldots, r_{2m_q-1}$, Eq. (3) can be rewritten as

$$Pr_{fuzzy}(o, q, \varepsilon_q) = \sum_{i=1}^{2m_q-1} \int_{x \in r_i} q.pdf(x) \cdot Pr_{range}(o, \odot(x, \varepsilon_q))dx. \tag{31}$$

When $x \in r_i$, $Pr_{range}(o, \odot(x, \varepsilon_q)) \leq UB_{Pr}(r_i, o, \varepsilon_q)$ (see Inequality (11)). Hence:

$$Pr_{fuzzy}(o, q, \varepsilon_q) \leq \sum_{i=1}^{2m_q-1} \left(UB_{Pr}(r_i, o, \varepsilon_q) \cdot \int_{x \in r_i} q.pdf(x)dx \right). \tag{32}$$

Furthermore:

$$\int_{x \in r_i} q.pdf(x)dx = \begin{cases} QC_{i+1} - QC_i & \text{if } i \in [1, m_q - 1] \\ 1 - 2QC_{m_q} & \text{if } i = m_q \\ QC_{2m_q-i+1} - QC_{2m_q-i} & \text{if } i \in [m_q + 1, 2m_q - 1] \end{cases} \tag{33}$$

Substituting the above equation into Inequality (32), we arrive at Inequality (13). □

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their insightful comments.

REFERENCES

BARBARÁ, D., GARCIA-MOLINA, H., AND PORTER, D. 1992. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng. 4*, 5, 487–502.

BECKMANN, N., KRIEGEL, H.-P., SCHNEIDER, R., AND SEEGER, B. 1990. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD*. ACM, New York. 322–331.

BERG, M., KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O. 2000. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, New York.

CHENG, R., KALASHNIKOV, D., AND PRABHAKAR, S. 2006a. The evaluation of probabilistic queries over imprecise data in constantly-evolving environments. *Inf. Syst. 32*, 1, 104–130.

CHENG, R., KALASHNIKOV, D. V., AND PRABHAKAR, S. 2003. Evaluating probabilistic queries over imprecise data. *In Proceedings of ACM SIGMOD*. ACM, New York. 551–562.

CHENG, R., KALASHNIKOV, D. V., AND PRABHAKAR, S. 2004a. Querying imprecise data in moving object environments. *IEEE Trans. Knowl. Data Eng. 16*, 9, 1112–1127.

CHENG, R., XIA, Y., PRABHAKAR, S., SHAH, R., AND VITTER, J. S. 2004b. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proceedings of the Symposium on Very Large Databases*. 876–887.

CHENG, R., ZHANG, Y., BERTINO, E., AND PRABHAKAR, S. 2006b. Preserving user location privacy in mobile data management infrastructures. In *Proceedings of the Privacy Enhancing Technology*

Workshop (PET 2006) (Cambridge, UK, June). Lecture Notes in Computer Science. Springer-Verlag, New York, 393–412.

DALVI, N. AND SUCIU, D. 2005. Answering queries from statistics and probabilistic views. In *Proceedings of the Symposium on Very Large Databases*. 805–816.

DALVI, N. N. AND SUCIU, D. 2004. Efficient query evaluation on probabilistic databases. In *Proceedings of the Symposium on Very Large Databases*. 864–875.

DANIELS, K. L., MILENKOVIC, V. J., AND ROTH, D. 1997. Finding the largest area axis-parallel rectangle in a polygon. *Comput. Geom. 7*, 125–148.

DE ALMEIDA, V. T. AND GÜTING, R. H. 2005. Supporting uncertainty in moving objects in network databases. In *Proceedings of the ACM International Symposium on Advances in Geographie Information Systems*. ACM, New York, 31–40.

DESHPANDE, A., GUESTRIN, C., MADDEN, S., HELLERSTEIN, J., AND HONG, W. 2004. Model-driven data acquisition in sensor networks. In *Proceedings of the Symposium on Very Large Databases*. 588–599.

FUHR, N. 1995. Probabilistic datalog - a logic for powerful retrieval methods. In *SIGIR*. 282–290.

GAEDE, V. AND GUNTHER, O. 1998. Multidimensional access methods. *ACM Comput. Surv. 30*, 2, 170–231.

GALINDO, J., URRUTIA, A., AND PIATTINI, M. 2006. *Fuzzy Databases: Modeling, Design, and Implementation*. Idea Group Publishing, ISBN: 1-59140-324-3.

HAN, S., CHAN, E., CHENG, R., AND LAM, K. Y. 2007. A statistics-based sensor selection scheme for continuous probabilistic queries in sensor networks. *Real-Time Syst. J. 35,* 1, 33–58.

HUNG, E., GETOOR, L., AND SUBRAHMANIAN, V. S. 2003. PXML: A probabilistic semistructured data model and algebra. In *Proceedings of the IEEE International Conference on Data Engineering*, IEEE Computer Society Press, Los Alamistos, CA, 467.

KHANNA, S. AND TAN, W. 2001. On computing functions with uncertainty. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ACM, New York, 171–182.

KRIEGEL, H.-P., KUNATH, P., PFEIFLE, M., AND RENZ, M. 2006. Probabilistic similarity join on uncertain data. In *Proceedings of the International Conference on Database Systems for Advanced Applications*. 295–309.

LAKSHMANAN, L., LEONE, N., ROSS, R., AND SUBRAHMANIAN, V. 1997. Probview: A flexible probabilistic database system. *Trans. Datab. Syst. 22*, 3, 419–469.

LEE, S. K. 1992. An extended relational database model for uncertain and imprecise information. In *Proceedings of the Conference on Very Large Databases*. 211–220.

LEUTENEGGER, S. T., EDGINGTON, J. M., AND LOPEZ, M. A. 1997. STR: A simple and efficient algorithm for r-tree packing. In *Proceedings of the IEEE International Conference on Data Engineering,* IEEE Computer Society Press, Los Alamitos, CA, 497–506.

LIM, E.-P., SRIVASTAVA, J., AND SHEKHAR, S. 1996. An evidential reasoning approach to attribute value conflict resolution in database integration. *IEEE Trans. Knowl. Data Eng. 8*, 5, 707–723.

LIU, K. AND SUNDERRAMAN, R. 1987. An extension to the relational model for indefinite databases. In *Proceedings of the ACM-IEEE Computer Society Fall Joint Computer Conference*. ACM, New York, 428–435.

LIU, K. AND SUNDERRAMAN, R. 1991. A generalized relational model for indefinite and maybe information. *IEEE Trans. Knowl. Data Eng. 3*, 1, 65–77.

NIERMAN, A. AND JAGADISH, H. V. 2002. ProTDB: Probabilistic data in XML. In *Proceedings of the Conference on Very Large Databases*. ACM, New York, 646–657.

OLSTON, C., LOO, B. T., AND WIDOM, J. 2001. Adaptive precision setting for cached approximate values. In *Proceedings of the ACM SIGMOD Symposium*. ACM, New York, 355–366.

OLSTON, C. AND WIDOM, J. 2000. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proceedings of the Conference on Very Large Databases*. ACM, New York, 144–155.

OLSTON, C. AND WIDOM, J. 2002. Best-effort cache synchronization with source cooperation. In *Proceedings of the ACM SIGMOD Symposium*. ACM, New York, 73–84.

PAGEL, B.-U., SIX, H.-W., TOBEN, H., AND WIDMAYER, P. 1993. Towards an analysis of range query performance in spatial data structures. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM, New York, 214–221.

PFOSER, D. AND JENSEN, C. S. 1999. Capturing the uncertainty of moving-object representations. In *Proceedings of the Symposium on Advances in Spatial Databases*. Springer-Verlag, New York, 111–132.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 2002. *Numerical Recipes in C++*. Cambridge University Press, Cambridge, MA.

SARMA, A. D., BENJELLOUN, O., WIDOM, J., AND HALEVY, A. 2006. Working models for uncertain data. In *Proceedings of the IEEE International Conference on Data Engineering*. IEEE Computer Society Press, Los Alamitos, CA.

SISTLA, A. P., WOLFSON, O., CHAMBERLAIN, S., AND DAO, S. 1997. Querying the uncertain position of moving objects. In *Temporal Databases, Dagstuhl*. 310–337.

SWEENEY, L. 2002. k-anonymity: A model for protecting privacy. *Int. J. Uncer. Fuzziness Knowl.-based Syst. 10*, 5, 557–570.

TAO, Y., CHENG, R., XIAO, X., NGAI, W. K., KAO, B., AND PRABHAKAR, S. 2005. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proceedings of the Symposium on Very Large Databases*. 922–933.

TEIXEIRA DE. ALMEIDA, V., AND GÜTING, R. H. 2005. Supporting uncertainty in moving objects in network databases. In *Proceedings of the GIS,* 31–40.

THEODORIDIS, Y. AND SELLIS, T. K. 1996. A model for the prediction of R-tree performance. In *Proceedings of the ACM SIGACT-SIGMOD SIGART Symposium on Principles of Database Systems*. ACM, New York, 161–171.

TRAJCEVSKI, G., WOLFSON, O., HINRICHS, K., AND CHAMBERLAIN, S. 2004. Managing uncertainty in moving objects databases. *Trans. Datab. Syst. 29*, 3, 463–507.

WIDOM, J. 2005. Trio: A system for integrated management of data, accuracy, and lineage. In *Proceedings of CIDR*. 262–276.

WOLFSON, O., SISTLA, A. P., CHAMBERLAIN, S., AND YESHA, Y. 1999. Updating and querying databases that track mobile units. *Distrib. Paral. Datab. 7*, 3, 257–387.

Received June 2006; revised January 2007; accepted April 2007