Index for Fast Retrieval of Uncertain Spatial Point Data [*]

Dmitri V. Kalashnikov   Yiming Ma   Sharad Mehrotra   Ramaswamy Hariharan

Information and Computer Science
University of California, Irvine, USA

ABSTRACT

Location information gathered from a variety of sources in the form of sensor data, video streams, human observations, and so on, is often imprecise and uncertain and needs to be represented approximately. To represent such uncertain location information, the use of a probabilistic model that captures the imprecise location as a probability density function (pdf) has been recently proposed. The pdfs can be arbitrarily complex depending on the type of application and the source of imprecision. Hence, efficiently representing, storing and querying pdfs is a very challenging task. While the current state of the art indexing approaches treat the representation and storage of pdfs as a black box, in this paper, we take the challenge of representing and storing any complex pdf in an efficient way. We further develop techniques to index such pdfs to support the efficient processing of location queries. Our extensive experiments demonstrate that our indexing techniques significantly outperform the best existing solutions.

Categories and Subject Descriptors

H.2.8 [**Information System**]: Database Management—*Spatial databases and GIS*

General Terms

Algorithms, Performance

Keywords

Spatial Database, Uncertainty Indexing, U-grid, Probability

1. INTRODUCTION

With the emergence of mobile computing and ubiquitous connectivity, numerous new applications that exploit location information have emerged (e.g, location based search, location based services, etc.) In such applications, location information of individuals (or devices) is determined in various ways. These include location sensors such as GPS, WiFi access point triangulation, cell phones, etc. Location information may also be derived from conversations wherein people describe their location through speech and/or text.

In all of these cases, location information is often imprecise. Devices such as GPS, cell phones, etc. have fundamental limitation on their localization capabilities. When people report their locations or describe events, they usually do so in an imprecise manner – e.g., "an accident happened on Freeway 405 near McArthur Street".

Developing location-based systems requires mechanisms to represent, query and reason with location uncertainty. A model for representing location uncertainty that has recently emerged is the probabilistic spatial model [2,3,6]. In such a model, uncertain location is represented as a probability of the event/object to have occurred at a given location (i.e., a probability density function over the spatial domain). Such a probabilistic model is powerful for a variety of reasons. First, it can be used to represent not only simple cases of uncertainty that arise due to device errors, but also fairly complex types of uncertainties that may arise in interpreting textual descriptions of location and many other applications [15]. Another advantage of the probabilistic model is that it allows for a clear and formal interpretation of spatial queries and has been demonstrated to be effective in practice.

While probabilistic model provides a powerful framework for representing uncertain location data, it introduces new challenges in supporting spatial queries efficiently. In this paper, we develop an efficient approach to indexing uncertain location information represented using probabilistic distributions over space. We develop efficient query processing algorithms that exploit the index structure developed for efficient retrieval. Specifically, we focus on probabilistic range query (RQ) and preference range query (PRQ), which are described later. The **main contributions** of this paper are:

- A novel indexing framework and algorithms for efficient spatial query processing (Sections 5 and 6).
- Extensive empirical evaluation of the proposed approach (Section 7).

The rest of the paper is organized as follows. Section 2 covers the related work. Section 3 defines two types of probabilistic retrieval queries. We then discuss ways to represent uncertain locations in a database in Section 4. Sections 5 and 6 introduce the novel index, U-grid, and query processing algorithms. In Section 7 we empirically evaluate the proposed approach. Finally, we conclude in Section 8.

[*]This research was sponsored by NSF Award number 0331707 and 0331690 to the RESCUE project [12].

2. RELATED WORK

In this section, we review most related work on representing spatial uncertainty, indexing, and query processing.

Uncertain location data representation. We will use the probabilistic model for spatial uncertainty developed in [3, 5–8], because it builds on the formal probability theory and it has been shown to be very effective in practice. Similar models are also employed in [2,4]. In the probabilistic model, an uncertain location ℓ is treated as a continuous random variable (r.v.) which takes values (x, y) inside domain Ω and has a certain probability density function (pdf) $f_\ell(x, y)$ associated with it. Interpreted this way, for any spatial region R, the probability that ℓ is inside R is computed as $\int_R f_\ell(x, y)dxdy$.

Representation. In many applications that deal with uncertain event locations [8, 11], we need to be able to represent pdfs of complex shapes in the database. There are several known methods for such a representation, such as histograms and modeling pdf as a mixture of Gaussians or of other distributions. In our approach, we will represent pdfs as histograms, which are stored in the database as quad-trees. Using quad-trees allows us to achieve fast query response time and also to compress those histograms (Section 4). It is interesting to note that the existing solutions that also deal with probabilistic spatial queries [3, 6] do not address the pdf representation issues directly. The reason is that their empirical evaluation is carried out using only simple densities such as uniform and Gaussian.

Indexing and query processing. In Sections 5 and 6 we introduce novel indexing and query processing techniques. Indexing pdfs to support efficient query processing has been explored in previous literature as well [3, 6]. A typical approach is to restrict possible values of an uncertain location ℓ to be inside an *uncertainty region* U_ℓ such that $f_\ell(x, y) = 0$ if $(x, y) \notin U_\ell$. Figure 1(a) in Section 3 illustrates that concept by showing that locations of events a_1, a_2, a_3, a_4 are restricted to their respective uncertainty regions (shaded). The uncertain location is indexed using data structures such as R-tree based on their uncertainty region. This allows answering certain spatial queries without performing costly integration operations. For instance, by analyzing the uncertainty regions in Figure 1(a) it is clear that the location of a_1 is guaranteed to be inside the range R.

Current state of the art. The x-bounds and U-tree are the current state of the art techniques studied in [6,15] for 1- and n-dimensional cases. Both techniques have been implemented as variations of R-tree. For 1-dimensional case, the idea is to store in each internal node $\mathcal{N}$ of an R-tree extra information: left and right 'x-bounds' for several values of x, e.g. $x = 0.1$ and $x = 0.2$. A 'left x-bound' of $\mathcal{N}$ is any value $l(x) \in \mathbb{R}$ such that for any pdf $f(z)$ covered by $\mathcal{N}$ it follows that $\int_{-\infty}^{l(x)} f(z)dz < x$. A 'right x-bound' is defined similarly. So, if a probabilistic threshold query (see Sec. 3) with range R and threshold p_τ overlaps the MBR of $\mathcal{N}$, but $\mathcal{N}$ stores a left x-bound such that (1) $R \subset (-\infty, l(x)]$; and (2) $p_\tau > x$, then $\mathcal{N}$ is pruned since it cannot contain any object (pdf) that would satisfy the τ-RQ. The idea of x-bounds generalizes to 2-dimensional case, but now, in addition to left and right x-bounds, top and bottom y-bounds should be maintained as well. Keeping x-bounds can increase the height of the R-tree due to the extra information kept in nodes of the R-tree, negatively affecting performance. Cheng et al. show in [6] that the gains of x-bounds outweigh their disadvantages in 1-dimensional case. Notice, unlike traditional methods, the x-bound approach employs for pruning not only spatial part (i.e. MBR), but also values of the pdf.

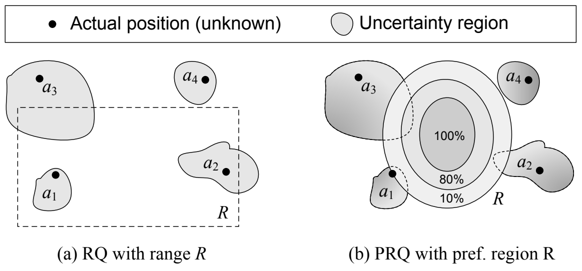


Figure 1: Examples of RQ(R) and PRQ(q).

3. SPATIAL QUERIES

In this section, we formally define two types of probabilistic retrieval queries studied in this paper. Similar queries have been considered in [2,3]. The first fundamental type of query is a *range* or *region* query, such as "find all the events, the location of which can be inside a given region". Those queries can be formally defined as:

Definition 1. *Given a region (range) R and a set of objects $A = \{a_1, a_2, \ldots, a_n\}$, a basic* **region (range) query (RQ)** *returns all the elements in A whose probability of being inside R is greater than zero.*

The analytical formula for computing the probability that a location $\ell \sim f_\ell(x, y)$ is located inside a region R is given by $\mathrm{P}(\ell \in R) = \int_R f_\ell(x, y)dxdy$.

Consider the example in Figure 1(a). If we assume the locations of elements $A = \{a_1, a_2, a_3, a_4\}$ are uniformly distributed in the specified (shaded) uncertainty regions, then the probabilities of these elements of being inside R might be $1.0, 0.7, 0.4, 0.0$ respectively. Then the result set for the corresponding RQ is $\{a_1, a_2, a_3\}$.

Having only elements in the result set might not be always sufficient: it is often necessary to be able to get the actual values of the probabilities associated with those elements, which leads us to the next definition:

Definition 2. *A probabilistic query is a* **detached-probability** *query if it returns elements without the probabilities associated with them. A query is an* **attached-probability** *query, denoted as p-, if its result is a set of tuples where each tuple consists of an element and the probability associated with this element.*

By default, any spatial query is a detached-probability query. The answer set for the p-RQ counterpart of the above RQ is $\{\langle a_1, p_1\rangle, \langle a_2, p_2\rangle, \langle a_3, p_3\rangle\}$. Given that we have assumed p_1, p_2, p_3 have specific values, the answer is $\{\langle a_1, 1.0\rangle, \langle a_2, 0.7\rangle, \langle a_3, 0.4\rangle\}$.

When analyzing results of an RQ, it is often desirable to ignore low-probability answers and present only those elements whose associated probabilities exceed a given *probability threshold* p_τ, where $p_\tau \in \mathbb{R}$, $0 \le p_\tau \le 1$. To generalize this idea, we can define:

Definition 3. *Given a threshold p_τ, query τ-Q is said to be query Q* **with the threshold semantics** *if on the same input as Q it returns all the elements from the result set of Q whose associated probabilities are greater than p_τ.*

To continue with our running example, if we set $p_\tau = 0.5$, then the corresponding τ-RQ will return $\{a_1, a_2\}$ as its result set, because p_1 and p_2 are greater than 0.5 whereas p_3 and p_4 are not. Similarly, $p\tau$-RQ will return $\{\langle a_1, 1.0\rangle, \langle a_2, 0.7\rangle\}$.

Notice, we have deliberately defined each τ-RQ as a single operation, and not as an additional filtering (τ- part) step applied to RQ operation. This is because τ-RQ, as a single query, can be optimized better, which is important for quick query response time.

Finally, observe that a regular region R can be viewed as a discrete function $R(x, y) : \Omega \to \{0, 1\}$, such that $R(x, y) = 1$ when $(x, y) \in R$, and $R(x, y) = 0$ otherwise. The formula for $P(\ell \in R)$ can be written as

$$P(\ell \in R) = \int_\Omega f_\ell(x, y) R(x, y) dxdy. \quad (1)$$

The concept of a regular region generalizes to the concept of a *preference region* $R(x, y) : \Omega \to [0, 1]$, which maps a point to a preference value between zero and one. Similarly, the concept a region query generalizes to the concept of a **preference region query (PRQ)**: everything is the same as for RQ, except $P(\ell \in R)$ is computed using Equation 1. PRQs give the flexibility to specify queries where, for example, the analyst is primarily interested in objects that are within a certain area of space, but also cannot ignore object in a larger area of space. An example of a PRQ with the preference region R is illustrated in Figure 1(b). There, the analyst prefers objects in the inner oval ($R(x, y) = 1$) to object in the middle ring ($R(x, y) = 0.8$) to objects in the outer ring ($R(x, y) = 0.1$). More complex preference regions can be defined, e.g. $R(x, y)$ can be a continuous function.

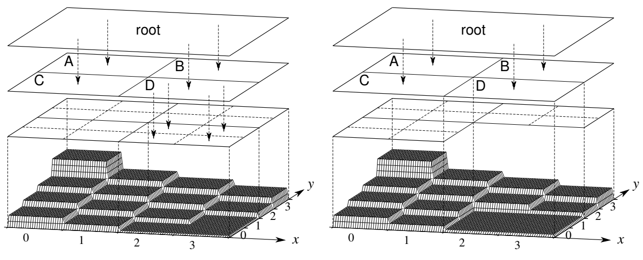
4. REPRESENTING PDFS

4.1 Histogram representation of pdfs

In order to represent and manipulate pdfs with complex shapes, we first quantize the space by viewing the domain Ω as a fine uniform grid G with cells of size $\delta \times \delta$. The grid G is *virtual* and is never materialized. We use the same notation G_{ij} for both the cell in i-th row and j-th column of G, i.e. $G = \{G_{ij}\}$, and for the spatial region it occupies. We refer to cells of the virtual grid G as *vcells*. A cell is treated as the finest element of space granularity and each spatial region can be defined by specifying the set of cells it occupies. A region R cannot occupy only part of a cell: either it occupies the whole cell or it does not occupy the cell at all. We will use vcellsin(R) to denote the number of cells R occupies.

The pdf $f_\ell(x, y)$ for any location ℓ is first viewed as a *histogram*: for each cell G_{ij} the probability p^ℓ_{ij} of ℓ to be inside this cell is computed as $p^\ell_{ij} = \int_{G_{ij}} f_\ell(x, y)dxdy$. For an event with location ℓ, we are naturally interested in the set of all the cells in which this event can be located: $U_\ell = \{G_{ij} : p^\ell_{ij} \neq 0\}$. We will call all cells for which $p^\ell_{ij} = 0$ the *zero-cells* (for ℓ). Since $f_\ell(x, y)$ is a pdf, $\sum_{G_{ij} \in U_\ell} p^\ell_{ij} = 1$. We will use the notation U_ℓ to refer to both the set of cells as well as the region they occupy. Notice that the latter simply defines an uncertainty region for ℓ. The uncertainty region U_ℓ along with p^ℓ_{ij} for each $G_{ij} \in U_\ell$ defines the histogram H_ℓ for $f_\ell(x, y)$: $H_\ell = \{U_\ell, \{p^\ell_{ij} : G_{ij} \in U_\ell\}\}$.

At this point a naïve solution is to represent (and store on disk) each pdf as a histogram, e.g. by first identifying the minimum bounding rectangle (MBR) for the histogram and then treating the cells inside the MBR as a 2-dimensional array of real values, which can be stored sequentially on disk.



(a) Before compression. (b) After compression.

Figure 2: Quad-tree representation of pdf.

By replacing a pdf $f_\ell(x, y)$ with its histogram H_ℓ we essentially quantize this pdf: we *approximate* $f_\ell(x, y)$ with a different pdf $f_{H_\ell}(x, y)$. The new pdf $f_{H_\ell}(x, y)$ has constant value inside each cell G_{ij}: $f_{H_\ell}(x, y) = \frac{1}{\delta^2} \int_{G_{ij}} f_\ell(x, y)dxdy = \frac{p^\ell_{ij}}{\delta^2}$ for $(x, y) \in G_{ij}$. Also, the histogram H_ℓ can be viewed as the probability mass function (pmf) that corresponds to $f_\ell(x, y)$ given that the domain Ω is viewed not as a continuous space but as a discrete space of finite number of cells.

When we approximate a pdf with a histogram representation, we lose the information about the precise shape of the pdf, but, in return, we gain several advantages. The main advantage is that manipulations with pmfs are less computationally expensive than the corresponding operations with pdfs, which involve costly integrations. This is essential, given that many types of applications require quick query response time.

It is straightforward to show that all the formulas described in the previous section, which involve pdfs and integration, can now be reformulated in terms of summations and probabilities p^ℓ_{ij}'s. E.g., in Defn. 1, the formula $P(\ell \in R) = \int_R f_\ell(x, y)dxdy$ becomes $P(\ell \in R) = \sum_{ij:G_{ij} \in R} p^\ell_{ij}$.

4.2 Quad-tree representation of pdfs

We might be able to improve the histogram representations of pdfs further by making two observations.

Firstly, a histogram as a representation of pdf, may require large storage overhead for the pdfs that cover large areas, especially if G is a very fine grid. To reduce the storage overhead as well as the number of I/Os required to retrieve a pdf from disk (thus improving the execution time), the analyst must be able to specify that any representation of a given pdf must fit in $s_\tau \in \mathbb{N}$ amount of space. E.g., the analyst might decide that certain types of locations are just not important, and representing each finest detail of them is not required. Let us note that the text of all the original reports is stored in the database as well. Therefore, if the analyst decides to represent certain location in more detail at a later time, that can be achieved from the stored reports.

Secondly, keeping certain aggregate information about probabilities in a given histogram, may allow for more efficient query processing. For instance, consider a τ-RQ with a threshold p_τ and a range R, whose overlap with the MBR of a given histogram H_ℓ for a location ℓ consists of only n cells. Assume for each histogram we keep the maximum value p^ℓ_{max} among all the p^ℓ_{ij} probabilities in the histogram. Then, we might be able to answer the τ-RQ more efficiently. Specifically, if $np^\ell_{max} < p_\tau$, then since $P(\ell \in R) \le np^\ell_{max} < p_\tau$, it immediately follows that ℓ does not belong to the answer set of the τ-RQ, without performing costly computations and many disk I/Os. We can generalize the idea of keeping aggregate information to multiple levels of resolution, not only at the MBR level.

To address the above observations, we can index each histogram H_ℓ using a space partitioning index, such as a quad-tree. First we build a *complete* quad-tree $\mathcal{T}_\ell$ for H_ℓ. The algorithm we use is consistent (and thus omitted) with that for building quad-tree for images [14], where the goal is to index the pixels that belong to a particular image. In our case, a histogram plays the role of an image, and the non-zero cells the role of the pixels.

Each node $\mathcal{N}$ of the quad-tree $\mathcal{T}_\ell$ is adjusted to store certain aggregate information. Assume that the bounding rectangle of $\mathcal{N}$ intersects with n vcells, whose p^ℓ_{ij} values are $p_1, p_2, \ldots, p_n$. Observe that some of those vcells might not belong to U_ℓ, and thus some of the p_i's can have zero values. Then, if $\mathcal{N}$ is a leaf node, it stores a positive real value p_{sum}, and it holds that: $p_1 = p_2 = \cdots = p_n$ and $p_{sum} = \sum_{i=1}^n p_i = np$, where $p = p_1$. If $\mathcal{N}$ is an internal node, it stores two values: $p_{sum} = \sum_{i=1}^n p_i$ and $p_{max} = \max_{i=1,\ldots,n} p_i$, which are used for pruning (in this case, $p_i \neq p_j$ in general). Figure 2(a) shows an example of a (3-level) complete quad-tree built on a 4×4 histogram.

Compression. The complete quad-tree pdf representation does address the issue of maintaining aggregate information at different levels of resolution. It still does not address the first concern, where the analyst can specify that any representation of a given pdf should fit into a certain amount of space $s_\tau \in \mathbb{N}$. The quad-tree (lossy) compression algorithm resolves this issue. It operates by maintaining the 'compressible nodes' in a priority queue. A node is compressible if all of its children are leaf nodes. In Figure 2(a), the compressible nodes are A, B, C, D. The algorithm extracts from the priority queue the 'best' node to compress next $\mathcal{N}$ and performs a *node compression* operation on $\mathcal{N}$. This operation consists of removing all the children of $\mathcal{N}$ and converting $\mathcal{N}$ into a leaf node. Semantically, the effect of this operation corresponds to replacing p^ℓ_{ij}'s of all the cells, covered by the BR of $\mathcal{N}$, with the same value p, equal to the average of those p^ℓ_{ij}'s. After each node compression operation, the histogram is replaced with another, more 'crude', histogram of the original pdf. In the resulting histograms the new p^ℓ_{ij}'s still sum up to 1. A node compression can make the parent of $\mathcal{N}$ to become a compressible node, in which case it is inserted in the priority queue. The algorithm keeps compressing nodes until the quad-tree fits into the specified amount of space s_τ.

The key for the priority queue can be chosen in a number of ways. Recall that for any histogram H there is pdf $f_H(x, y)$ that corresponds to it. Intuitively, we want the shape of $f_H(x, y)$ for the resulting histogram H to resemble/approximate that of the original pdf as closely as possible. The algorithm above tries to achieve that approximation goal in a greedy fashion, by repeatedly collapsing 'best' compressible nodes. Thus, we need to decide what constitutes the best node to compress next. Graphically, the effect of compressing a node $\mathcal{N}$ with bounding rectangle B corresponds to 'flattening' of the curvature of the histogram for the cells covered by B. Thus, we want to choose those nodes to flatten whose corresponding histogram curvature looks the flattest so that we minimally alter the shape.

Consider a compressible node $\mathcal{N}$ with bounding rectangle B which covers n cells. Suppose the probabilities in those cells are: $p_1, p_2, \ldots, p_n$. By compressing this node, the probabilities will become $p_1 = \cdots = p_n = \mu$, where $\mu = \frac{1}{n}\sum_{i=1}^n p_i$. So, one solution is to choose $\xi = \max_{i=1,\ldots,n} p_i - \min_{i=1,\ldots,n} p_i$ as the priority queue key.

Figure 2(a) shows an example where four nodes are compressible: A, B, C, D, where node A covers cells $[0, 1] \times [2, 4]$, node D covers cells $[2, 3] \times [0, 1]$, etc. The cells covered by D clearly have the least variance, so D is picked for compression. Figure 2(b) shows the result of compressing D. Notice the flattening effect in the cells covered by D.

5. INDEXING

Assume the goal is to evaluate a τ-RQ with some threshold. The quad-tree representation of pdfs might help to evaluate this query over each *individual* event location $\ell \sim f_\ell(x, y)$ stored in the database faster. However, if nothing else is done, answering this query will first require a *sequential scan* over all the event locations stored in the database, which is undesirable both in terms of disk I/O as well as CPU cost.

To solve this problem we can create a *directory index* on top of U_ℓ (or, MBR of the histogram) for each location ℓ in the database, using an index of our choice, such as R*-tree [1]. This solution is illustrated in Figure 3. When processing the τ-RQ with region R, we can effectively use this index to prune away all the U_ℓ which do not intersect with R. Similar techniques have been studied in [3,6]. This method is known to lead to improvement when compared to a sequential scan. However, this method essentially utilizes only the *spatial* part of a pdf (i.e., U_ℓ or the MBR) and disregards the fact that there is another 'dimension' that can be used for pruning as well – the values of the pdf. Let us note that 'dimension' here is used figuratively, as it is not really a dimension, as defined in the traditional multidimensional indexing. Consequently, there is no straightforward way of applying the traditional approach of employing 3-dimensional index [6,15]. Instead, the challenge becomes to create an indexing solution that is capable of using the values of pdfs. We next discuss how to solve this challenge.

5.1 U-grid

In this section, we propose a novel U-grid (**U**ncertain **grid**) indexing structure to effectively index uncertain event data. In addition to storing spatial information in the grid files [9,10,13], U-grid also stores probability summarizations, which can be effectively utilized by the query processor.

Unlike the virtual grid G, the directory U-grid I is materialized. The grid I is much coarser than G and can be viewed as a small 2-dimensional array of cells $I = \{I_{ij}\}$, residing in main memory (but it can also be stored on disk). As before, we will use I_{ij} to refer to both: the cell and the region it represents. We will refer to cells in the directory grid I as *dcells*. The structure of the grid I is shown in Figure 4. Each cell I_{ij} in the grid I stores certain aggregate information about each event location $\ell \sim f_\ell(x, y)$ whose uncertainty region U_ℓ intersects with I_{ij}. Let us denote the set of those locations as the '*set*' L_{ij}.

The most important information in I_{ij} is a *pointer* to a **disk-resident** list, called the '*list*' L_{ij}. For each event location ℓ in the set L_{ij}, there is a list-element e_ℓ in the list L_{ij}, which stores aggregate information $\{oid, p_{max}, p_{sum}, MBR\}$ for ℓ, as illustrated in Figure 4. The attribute $e_\ell.oid$ points

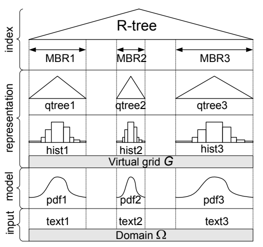


Figure 3: R-tree.

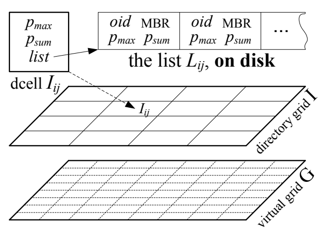

p_{max}
p_{sum}
list
oid MBR p_{max} p_{sum}
oid MBR p_{max} p_{sum}
...
the list L_{ij}, on disk
dcell I_{ij}
I_{ij}
directory grid I
virtual grid G

Figure 4: Grid.

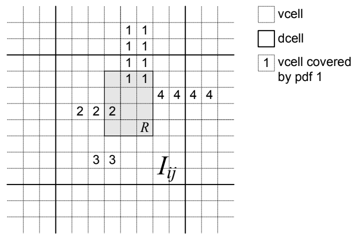


Figure 5: τ-RQ in I_{ij}

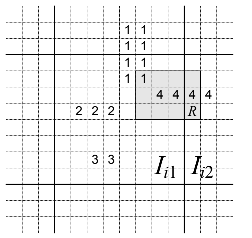


Figure 6: RQ: I_{i1}, I_{i2}

to the location of the quad-tree for ℓ, on disk. The MBR of ℓ is stored in $e_\ell.MBR$. Let $n = \text{vcellsin}(U_\ell \cap I_{ij})$, and let $p_1, p_2, \ldots, p_n$ be the values of the p^ℓ_{ij} probabilities of the n vcells in $U_\ell \cap I_{ij}$. Then, $e_\ell.p_{max}$ and $e_\ell.p_{sum}$ store the values: $e_\ell.p_{max} = \max_{i=1,\ldots,n} p_i$ and $e_\ell.p_{sum} = \sum_{i=1}^n p_i$. The list is **sorted** in descending order on either p_{sum} or p_{max}.

Clustering. Observe that we also can retrieve $\{p_{max}, p_{sum}, MBR\}$ information directly from the root node of the quad-tree for ℓ, instead of the list-element e_ℓ for ℓ, so why do we store it in e_ℓ again? The reason is that, when we analyze elements of the list sequentially, if we access the quad-trees, we will incur an extra disk I/O per *each* location stored in the list. By storing all those attributes we achieve *clustering* of locations.

Other aggregate information. The lists in the directory grid I can store other aggregate information for the purpose of pruning, not only the information from the root node of the quad tree $\{p_{max}, p_{sum}, MBR\}$. We refer to this technique as *L0Sketch* (level-0 sketch).

We can naturally extend this technique, by storing more levels of the quad-tree. In particular, we shall see that using *L1Sketch* produces the best results. *L1Sketch* requires small additional overhead, compared to *L0Sketch*, since children's Bounding Rectangles (BR) can be derived from $e_\ell.MBR$, and thus they do not need to be stored. The pruning power of *L1Sketch* outweighs that of *L0Sketch* and easily compensates for the storage overhead.

One can also consider storing x-bounds here, but that turned out not to work well in practice.

Other attributes of I_{ij}. In addition to the pointer to the list L_{ij}, the dcell I_{ij} also has p_{max} and p_{sum} attributes, which store the maximum over all p_{max}'s and over all p_{sum}'s in the list L_{ij} respectively. In the next section, we discuss methods for efficient query processing using the grid.

6. EFFICIENT QUERY PROCESSING

Processing of τ-RQs and τ-PRQs consists of two logical phases: the index (pruning) phase and the object (post-processing) phase. The first phase employs the directory-index to prune the locations that cannot satisfy the query. Two different factors are important in this process: the speed and the quality of pruning. The locations that cannot be pruned using the directory index are inserted in the special list L_{proc} to be *post-processed* later, on the second phase. The way in which each individual event location $\ell \in L_{proc}$ is post-processed, with respect to a given query, is independent from the choice of directory-index. The procedure that achieves that is rather straightforward. It simply traverses the quad-tree to compute the desired probability for a given query. It might stop earlier, without computing the total probability, e.g. when it becomes clear that the probability for given ℓ to satisfy the query either will or will not exceed the threshold p_τ. The details of that procedure are omitted.

6.1 Processing τ-RQs using U-grid

In this section, we present the algorithms for processing of a τ-RQ with a range R and a threshold p_τ using the directory U-grid index I. The idea of the solution is to avoid the costly computation of the exact probability for a location ℓ to be in R, i.e. $\text{P}(\ell \in R) \overset{\text{def}}{=} \sum_{ij:G_{ij} \in U_\ell \cap R} p^\ell_{ij}$. We accomplish that by being able to find a good upper-bound λ for it, such that $\text{P}(\ell \in R) \leq \lambda$. The value λ will be computed based on the aggregate information stored in the grid I. The pruning will utilize the fact that, if $\lambda < p_\tau$, then $\text{P}(\ell \in R) \leq \lambda < p_\tau$, and thus ℓ cannot satisfy the τ-RQ and can be pruned without computing the exact value of $\text{P}(\ell \in R)$.

One-cell scenario. Let us first consider the case where R is completely inside of only one cell I_{ij} of I, as illustrated in Figure 5. The pruning phase of the τ-RQ algorithm consists of four levels of pruning:

(1) *The grid-level pruning.* Given R intersects with only one dcell I_{ij}, we know that the grid index I is constructed such that only the locations in L_{ij} can satisfy the τ-RQ. Thus, the other locations can be pruned.

(2) *The dcell-level pruning.* Let us set $\lambda = \min(m \times I_{ij}.p_{max}, p_{sum})$, where $m = \text{vcellsin}(R)$. Observe that if $\lambda < p_\tau$, then no object $\ell \in L_{ij}$ satisfies the τ-RQ, because $\text{P}(\ell \in R) \leq \lambda < p_\tau$. Let us note that this pruning is carried out using only the content of the dcell I_{ij}, without performing any disk I/Os.

(3) *The list-level pruning.* If the above condition does not hold, the algorithm starts to process the disk-resident list L_{ij} sequentially. In the example shown in Figure 5, the list L_{ij} will consist of four elements, for the locations $\ell_1, \ell_2, \ell_3, \ell_4$. Assume that the elements in L_{ij} are sorted in descending order of p_{sum}. Suppose that the algorithm currently observes a list-element e_ℓ that corresponds to a location ℓ. Let us choose $\lambda = e_\ell.p_{sum}$. Observe that if $\lambda < p_\tau$, then neither ℓ nor the *rest* of the location in the list will satisfy the τ-RQ, i.e. **all** of them can be pruned.

(4) *The element-level pruning.* If the above pruning is not successful for the element e_ℓ, the algorithm might still be able to prune ℓ alone, but not the whole list. Let us set $\lambda = n \times e_\ell.p_{max}$, where $n = \text{vcellsin}(R \cap e_\ell.MBR)$. If $\lambda < p_\tau$, then ℓ is pruned. Else, ℓ is not pruned, and ℓ is inserted in the list L_{proc} to be post-processed. After that, the algorithm extracts the next element $e_\ell \in L_{ij}$ and goes to Step 3 to apply the list-level pruning until all the element of L_{ij} are processed. We can use *L1Sketch* to further reduce the upper bound by summing up the estimations from each quadrant: $\lambda = \sum_{i=1,4} n_i \times e_\ell.p_{max\_i}$.

Two-cell scenario. Let us now consider the situation

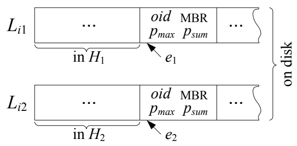


Figure 7: Two cells

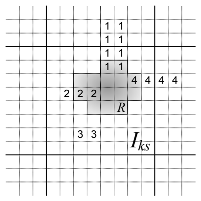


Figure 8: PRQ in I_{ks}

shown in Figure 6, where R can intersect with more than one dcell. We study only two-cell scenario for illustration purposes; the technique generalizes to any number of dcells.
(1) *The grid-level pruning.* Assume that R intersects with only two dcells: I_{i1} and I_{i2}. We know that the index grid I is constructed such that only the locations in L_{i1} and L_{i2} can satisfy the τ-RQ, the rest of the locations are pruned.
(2) *The dcell-level pruning.* Let $m_1 = \text{vcellsin}(R \cap I_{i1})$, $m_2 = \text{vcellsin}(R \cap I_{i2})$, and $\lambda = \min(m_1 \times I_{i1}.p_{max}, I_{i1}.p_{sum}) + \min(m_2 \times I_{i2}.p_{max}, I_{i2}.p_{sum})$. Then, if $\lambda < p_\tau$, then no objects in L_{ij} satisfies the τ-RQ, hence all of them are pruned.
(3) *The list-level pruning.* Otherwise, the algorithm scans two lists L_{i1} and L_{i2} at the same time in a sequential manner, as illustrated in Figure 7. Assume that the elements in those lists are sorted in descending order of p_{sum}. Suppose that the algorithm currently observes elements $e_1 \in L_{i1}$ and $e_2 \in L_{i2}$, which correspond to locations ℓ_1 and ℓ_2. Let us note that ℓ_1 and ℓ_2 can be the same object. All the previously observed but not-yet-pruned elements from L_{i1} and L_{i2} are kept in two sets $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively.

We need to upper-bound the probability $P(\ell \in R)$ for each location ℓ stored in the rest of the lists L_{i1} and L_{i2}, such that ℓ is not yet marked for post-processing (i.e., $\ell \notin H_1, H_2$). This probability is upper-bounded by $\lambda = e_1.p_{sum} + e_2.p_{sum}$. Thus, if $\lambda < p_\tau$, we do not need to load the rest of the lists from disk, and **all** the locations stored there can be pruned.
(4) *The element-level pruning.* Otherwise, we will not be able to prune *all* the locations, but we shall try to prune ℓ_1 and ℓ_2 *individually*. Let us consider how to prune ℓ_1, the case for ℓ_2 is similar. First, we compute two values: λ_1 and λ_2 to bound $P(\ell \in R \cap I_{i1})$ and $P(\ell \in R \cap I_{i2})$ respectively, so that $P(\ell \in R) \leq \lambda_1 + \lambda_2$. We compute $\lambda_1 = \min(n_1 \times e_1.p_{max}, e_1.p_{sum})$, where $n_1 = \text{vcellsin}(R \cap I_{i1} \cap e_1.MBR)$. The value of λ_2 depends on whether or not we have already observed ℓ_1 on the previous steps. Namely, if there is already an element e_{ℓ_1} in $\mathcal{H}_2$ for ℓ_1, then $\lambda_2 = \min(n_2 \times e_{\ell_1}.p_{max}, e_{\ell_1}.p_{sum})$, where $n_2 = \text{vcellsin}(R \cap I_{i2} \cap e_{\ell_1}.MBR)$. Else $\lambda_2 = e_2.p_{sum}$. Now we can compute $\lambda = \lambda_1 + \lambda_2$ and if $\lambda < p_\tau$, then we can prune ℓ_1. In the latter case, ℓ_1 is removed from $\mathcal{H}_2$ and L_{proc} – if it is already there. If the above methods cannot prune ℓ_1, then ℓ_1 is inserted into L_{proc} and e_1 into $\mathcal{H}_1$.

After ℓ_2 is processed in a similar manner, the algorithm extracts the next two elements e_1 and e_2 from L_{i1} and L_{i1} and repeats the pruning procedure from step 3, until all the elements in the lists are processed. Similar to the one-cell scenario, $L1Sketch$ can be used to further reduce the upper bound. We skip the details due to page limit.

6.2 Processing τ-PRQs using U-grid

Assume a τ-PRQ has a threshold p_τ and a preference region $R(x, y)$. Let us denote the average value of $R(x, y)$ in the cell G_{ij} as r_{ij}: $r_{ij} = \frac{1}{\delta^2}\int_{G_{ij}} R(x, y)dxdy$. Then, preference region $R(x, y)$ can be viewed as the set of tuples: $\{(G_{ij}, r_{ij}) : r_{ij} \neq 0\}$ and Equation 1 transforms into $P(\ell \in R) \overset{\text{def}}{=} \sum_{ij:G_{ij} \in R \cap U_\ell} p^\ell_{ij} r_{ij}$. The goal is, to avoid the costly computation of the exact probability $P(\ell \in R)$ by being able to find a good upper-bound λ for it, such that $P(\ell \in R) \leq \lambda$. The value of λ will be derived from the aggregate information stored in the directory grid I. The pruning will be based on the observation that, if $\lambda < p_\tau$, then we can prune ℓ since it cannot be in the answer set of the τ-PRQ.

Due to the page limit, we study only the scenario illustrated in Figure 8, where R intersects with only one dcell I_{ks}. The technique generalizes to any number of cells. First observe that any preference region $R(x, y)$ induces a regular region $R'(x, y) = \{(x, y) : R(x, y) \neq 0\}$. That regular region can be obtained by treating all non-zero r_{ij}'s of the preference region as 1's. Observe that $P(\ell \in R) = \sum_{ij:G_{ij} \in R \cap U_\ell} p^\ell_{ij} \times r_{ij} \leq \sum_{ij:G_{ij} \in R \cap U_\ell} p^\ell_{ij} \times 1 = P(\ell \in R')$. Thus, we can use the techniques from Section 6.1 to upper-bound $P(\ell \in R')$ and that would upper-bound $P(\ell \in R)$ as well, because $P(\ell \in R) \leq P(\ell \in R') \leq \lambda < p_\tau$.

Cauchy-Shwarz inequality. The most effective technique employed for pruning for τ-PRQ is based on Cauchy-Shwarz inequality: $\sum_{i=1}^n x_i y_i \leq (\sum_{i=1}^n x_i^2)^{\frac{1}{2}} (\sum_{i=1}^n y_i^2)^{\frac{1}{2}}$. Using it, we have: $P(\ell \in R) \overset{\text{def}}{=} \sum \ldots \leq (\sum_{ij:G_{ij} \in I_{ks}} (p^\ell_{ij})^2)^{\frac{1}{2}} \times (\sum_{ij:G_{ij} \in I_{ks}} (r_{ij})^2)^{\frac{1}{2}} = S_\ell S_R$, where S_ℓ and S_R are the two $(\cdot)^{\frac{1}{2}}$ terms. The value $S_R \in \mathbb{R}$ is computed once per query R and cell I_{ks}. The value $S_\ell \in \mathbb{R}$ is apriori stored in each element e_ℓ of the list L_{ks} along with other aggregate information. Now, when processing list-element $e_\ell \in L_{ij}$, we can use $\lambda = S_R \times e_\ell.S_\ell$ as one of the upper-bounds for $P(\ell \in R)$.

7. EXPERIMENTAL EVALUATION

In this section, we experimentally study the efficiency of the proposed approach. We ran all our experiments on a P4-2GHz PC with 1GB RAM. The results will demonstrate the significant speedup achieved by the proposed solution, compared to the current state of the art methods.

7.1 Experimental Setup

Indexing methods. Table 1 summarizes the basic probability summarization techniques that can be used by an indexing structure. The existing techniques are 1 and 2, see [15]. The new ones, proposed in this paper, are 3, 4, and 5. While the new techniques have been described in the context of U-grid, one can notice that some of them are also applicable to R*-tree and vice versa. Thus, to have a complete study, we also incorporate our techniques in R*-tree and evaluate their results as well.

Table 1 illustrated to which part of various indexing structures (R*-tree and U-grid) a given summarization technique is applicable. For instance, it shows that L1 technique can be applied to the element level of U-grid and leaf level of

Strategies	**Novel**	**R*-tree**		**U-grid**		
		internal	leaf	dcell	list	element
(1) Spatial-bound		Y	Y	Y	Y	Y
(2) x-bound		Y	Y			Y
(3) Max-Sum (MS)	Y	Y	Y	Y	Y	Y
(4) L1 Sketch (L1)	Y		Y			Y
(5) CS	Y		Y	Y	Y	Y

Table 1: Pruning Strategies

R*-tree. ***Max-Sum (MS)*** refers to using (p_{max}, p_{sum}) summary to prune objects. It can apply to R*-tree and U-grid at all levels. ***CS*** is the Cauchy-Shwarz technique employed by PRQ queries. Therefore, we have many individual index variations and we will only test most prominent solutions.

To evaluate these techniques effectively, we group them under two schemes: R*-tree and U-grid. We use R*-tree with Spatial-bound pruning as our comparison baseline, to evaluate the following existing and novel techniques:

• **Existing:** (1) R*-tree with x-bounds, also known as U-PCR [15], (2) R*-tree with compressed x-bounds, the current state of the art technique also known as U-tree [15] (3) Naïve Grid;

• **Novel:** (1) U-grid-MS with *MS* pruning, (2) U-grid with x-bounds (U-grid-x) and with compressed x-bounds (U-grid-x$^+$), (3) U-grid-MSL1 for *MS* pruning and *L1Sketch*, (4) RTree-MS, (5) RTree-MSL1.

Naïve Grid is a standard grid, which does not use probabilities for pruning. Note that to reduce the indexing overhead, x-bounds can also be compressed using linear approximation method [15]. U-grid-x and U-PCR store nine x-bounds for values from 0.1 to 0.9 at the interval of 0.1; the compressed versions – U-grid-x$^+$ and U-tree – are built on top of these nine x-bounds.

Domain. We propose U-grid as the best solution for city-level domains. In this paper, we use Manhattan Area of New York with a 400×400 virtual grid is overlaid on top of it.

Uncertain location data for testing the Efficiency is derived from reports filed by NYPD Officers. From these reports, 2359 uncertain locations are extracted and mapped into the probabilistic representation, using the framework from [8]. The number of the locations might not be sufficient for testing the scalability. Hence, we have generated several larger (10K–100K) synthetic datasets, based on this real dataset, as follows. We first partition the real events into three uncertainty categories, based on the size of their spatial uncertainty regions: the low (the pdf covers less than 10×10 vcells), medium (less than 20×20 vcells), and high (less than 100×100 vcells). To generate a synthetic location, we randomly choose a real uncertain location, and generate a new one with a similar pdf on top of a landmark (building, street intersection) in the domain. When storing the new pdf as a quad-tree and paginating it to the disk, an average-sized pdf occupies 5 to 10 disk pages. Using the lossy compression techniques, we constrain the size of any pdf to 100 pages maximum.

We can now control the *data uncertainty level* of a synthetic dataset, by mixing objects with different uncertainty levels. In our experiments, a dataset with the *medium* data uncertainty level has object uncertainty mixture ratio of $(low^{90\%} : med^{5\%} : high^{5\%})$, the low and high levels are defined as: $low = (98\% : 1\% : 1\%)$, and $high = (50\% : 30\% : 20\%)$. The *medium* data uncertainty has roughly the same mixture ratio as the 2359 real events.

Queries. Similar to the concept of object uncertainty levels, spatial queries have the low, medium and high *coverage* levels to characterize query size: 10×10, 20×20, and 100×100 vcells. They also have *query uncertainty levels* to characterize mixes of queries of various coverage levels: *low / med / high* are the same as data uncertainty levels.

Table 2 summarizes the default experimental settings. We vary those settings to analyze the performance of the different indexing strategies. For each setting, we execute a large number of spatial queries and then report the *average* disk I/O per one query.

Parameter	**Value**
Data Size	30,000
p_τ	0.8
Grid Size	16×16

Parameter	**Value**
Data Uncertainty	Medium
Query Uncertainty	Medium
Page Size	1024

Table 2: Default System Settings

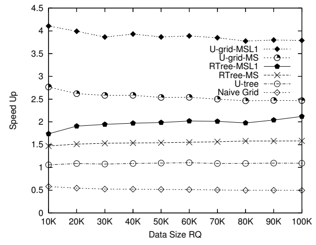


Figure 9: Data Size

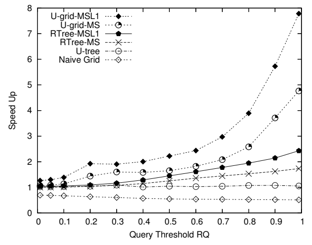


Figure 10: Threshold

7.2 Experiments

Table 3 studies the impact of different database sizes and query thresholds on the I/O cost of τ-RQs for the different indexing techniques. All tested techniques are several orders of magnitude better than linear scan (not shown). In terms of the overall execution time, linear scan takes more than 2 minutes to complete even for 10K of data. All the techniques, except for linear scan, have two phases as discussed in the beginning of Section 6: the indexing (phase I) and post-processing (phase II).

Data size. Table 3 shows the phase I and total I/O costs separately. U-grid-MS has the best phase I costs, and U-grid-MSL1 has the best overall costs. In all experiments, Rtree-MS has better performance than R-tree (baseline). This proves *MS* pruning is effective in R-tree index. However, the existing techniques based on the x-bounds (U-PCR and U-tree) do not perform well. This shows that *the x-bound idea does not work well in 2-d case*; the overhead in storing the x-bounds and compressed x-bounds clearly outweighs the gains from probability pruning. For a U-PCR indexing node, it needs to store additional 36 real numbers than a R-tree node. This reduces fan out by more than 8 times. Even with the compression techniques, to store the parameters used in the linear approximations, U-tree still needs to reduce the fan out by more than 4 times. The U-tree's performance is only slightly better than that of the baseline, and U-PCR has the worst overall performance.

Figure 9 shows the *improvement ratio (speedup)* of various indexing techniques, compared to the baseline (R-tree). The best technique from Grid group is U-grid-MSL1. It shows 4 times speedup over the baseline, and also almost 4 times improvement over the best existing solution, U-tree.

The best technique from R-tree group is Rtree-MSL1, but it is only about 2 times better than the baseline. The main reason for this difference is that for U-grid, the regions that correspond to lists are dcells, hence they cannot overlap; but, for RTree-List, the regions can (and do) overlap. Hence, the probabilistic pruning techniques become less effective. The experimental results also show that $L1Sketch$ is a very effective technique for both, R-tree and U-grid.

Query Threshold. Table 3 also shows similar behavior when the query threshold p_τ for τ-RQs is varied. As we can observe from Figure 10, the gain ratio for U-grid-MSL1 gets

Methods	Novel	*Overall I/O Cost*						*PhaseI I/O Cost*					
		Size			τ			**Size-I**			**τ-I**		
		10k	**50k**	**100k**	**0.1**	**0.5**	**0.8**	**10k**	**50k**	**100k**	**0.1**	**0.5**	**0.8**
R*-tree		88	385	740	437	259	235	24	65	98	44	44	44
U-PCR		109	457	853	553	306	238	60	209	359	160	136	136
U-tree		83	351	677	475	241	219	34	104	183	82	72	72
Rtree-MS	**Y**	60	250	467	432	207	154	**22**	**60**	**95**	**40**	**39**	**37**
U-tree-MS	**Y**	68	279	520	474	221	180	34	102	168	87	76	76
Rtree-MSL1	**Y**	**51**	**194**	**349**	**414**	**178**	**121**	36	116	194	73	68	60
Naïve Grid		152	746	1489	650	472	449	6	23	43	14	14	14
U-grid-MS	**Y**	32	152	300	384	157	91	**5***	**16***	**29***	**14***	**12***	**10***
U-grid-x$^+$-MS	**Y**	37	174	343	407	162	104	11	43	83	38	31	27
U-grid-MSL1	**Y**	**22***	**100***	**195***	**315***	**116***	**60***	10	39	74	34	27	25

Table 3: Disk I/O for Various Index Strategies

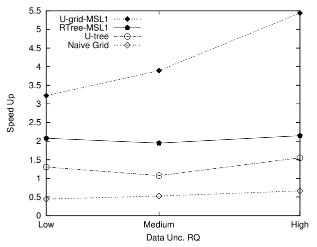


Figure 11: Data Unc.

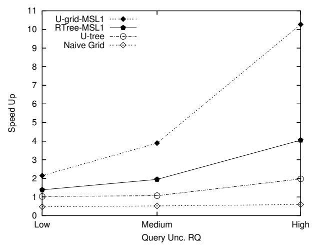


Figure 12: Query Unc.

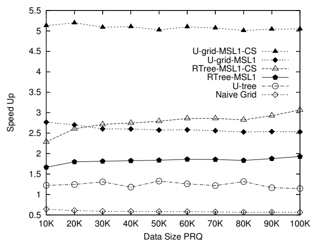


Figure 13: PRQ

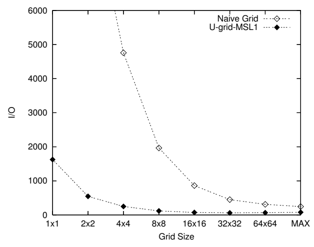


Figure 14: Grid size

even better (up to the factor of 8) when the threshold increases. It shows the combined pruning strategy (MS+L1) can dismiss most of the false positive objects instead of pushing them to phase-II.

Misc. performance. In Figures 11 and 12 we vary the data and query uncertainty levels. As data and query become more uncertain, pruning in phase I becomes very important since there is more objects intersecting with the query region. The results show that our combined pruning strategy (MS+L1) works very well for U-grid, but less effective for R-tree, since the overlapping MBRs in R-tree degrade its performance.

U-grid size. Figure 14 plots the impact of the grid size on the performance of τ-RQs for the U-grid-MSL1 technique. When the U-grid has only small number of cells, e.g. 1×1 or 2×2 cells, the overall I/O cost is high, since many objects are pushed to phase-II for post-processing. As the number of cells increases, the I/O cost **stabilizes**: having finer than 16×16 grid does not lead to significant improvement. Therefore, keeping the directory grid I in main memory, except for its disk-resident L_{ij} lists, is feasible.

PRQs. Figure 13 studies the effect of the data size on the performance of τ-PRQs. The new technique, U-grid-MSL1-CS, achieves 5 time speedup over the baseline, and 4 time speedup over the best existing solution, U-tree.

8. CONCLUSION

This paper presents a solution for indexing and fast retrieval of uncertain spatial point data, reflected as pdfs. The proposed approach will benefit a variety of applications that require fast query processing over uncertain (spatial) data. The proposed grid-based index, U-grid, achieves its effectiveness by using values of pdfs as another indexing 'dimension', namely by employing high-level summaries/sketches of the underlying pdfs. The paper also demonstrates that (regardless of the actual underlying representation of pdfs) there can be a merit of building an individual index (e.g., Quad-tree) on top of each pdf, to summarize it, and further increase query processing speed.

9. REFERENCES

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

[2] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

[3] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE TKDE*, 16(9), Sept. 2004.

[4] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluation of probabilistic queries over imprecise data in constantly-evolving environments. *Information Systems Journal*, 2006. to appear.

[5] R. Cheng, S. Prabhakar, and D. V. Kalashnikov. Querying imprecise data in moving object environments. In *ICDE*, 2003.

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

[7] D. V. Kalashnikov, Y. Ma, S. Mehrotra, and R. Hariharan. Modeling and querying uncertain spatial information for situational awareness applications. In *Proc. of ACM GIS*, 2006.

[8] D. V. Kalashnikov, Y. Ma, S. Mehrotra, R. Hariharan, N. Venkatasubramanian, and N. Ashish. SAT: Spatial Awareness from Textual input. In *Proc. of EDBT*, 2006.

[9] D. V. Kalashnikov, S. Prabhakar, and S. Hambrusch. Main memory evaluation of monitoring queries over moving objects. *Distributed and Parallel Databases*, 15(2):117–135, Mar. 2004.

[10] D. V. Kalashnikov, S. Prabhakar, S. Hambrusch, and W. Aref. Efficient evaluation of continuous range queries on moving objects. In *Proc. DEXA Conference*, September 2–6 2002.

[11] S. Mehrotra, C. Butts, D. V. Kalashnikov, N. Venkatasubramanian, and et el. CAMAS: A citizen awareness system for crisis mitigation. In *ACM SIGMOD*, 2004.

[12] S. Mehrotra, C. Butts, D. V. Kalashnikov, N. Venkatasubramanian, R. Rao, G. Chockalingam, R. Eguchi, B. Adams, and C. Huyck. Project RESCUE: challenges in responding to the unexpected. *SPIE*, 5304:179–192, Jan. 2004.

[13] J. Nievergelt, H. Hinterberger, and Sevcik. The grid file: An adaptable, symmetric multi-key file structure. In *ECI'81.*

[14] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

[15] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, 2005.