Effectively Indexing the Multi-Dimensional Uncertain Objects for Range Searching

Ying Zhang, Wenjie Zhang, Qianlu Lin, Xuemin Lin*

The University Of New South Wales

{yingz, zhangw, qlin, lxue }@cse.unsw.edu.au

ABSTRACT

The range searching problem is fundamental in a wide spectrum of applications such as radio frequency identification (RFID) networks, location based services (LBS), and global position system (GPS). As the uncertainty is inherent in those applications, it is highly demanded to address the uncertainty in the range search since the traditional techniques cannot be applied due to the inherence difference between the uncertain data and traditional data. In the paper, we propose a novel indexing structure, named U-Quadtree, to organize the uncertain objects in a multi-dimensional space such that the range searching can be answered efficiently by applying filtering techniques. Particularly, based on some insights of the range search on uncertain data, we propose a cost model which carefully considers various factors that may impact the performance of the range searching. Then an effective and efficient index construction algorithm is proposed to build the optimal U-Quadtree regarding the cost model. Comprehensive experiments demonstrate that our technique outperforms the existing works for range searching on multi-dimensional uncertain objects.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms

Algorithm, Performance

Keywords

Indexing, mutidimensional uncertain objects, range query

1. INTRODUCTION

In recent years, the database community has witnessed the increasing amount of research on uncertain data modeling and processing, due to its importance in many important applications such as data cleaning, radio frequency identification (RFID) networks, location based services (LBS), global position system (GPS), sensor data analysis, economic decision making and market surveillance. Common causes of uncertainty in these applications include data randomness and incompleteness, limitation of measuring equipment, delay or loss of data updates and privacy preservation.

The range searching problem is fundamental in the above applications. Below are two examples.

In some warehouse management systems, the RFID tags are attached to the items and their current locations can be obtained by RFID readers. Since the RFID reading may be noisy due to the sensitivity of the low cost readers to various environmental factors such as interference from nearby mental objects and contention among tags [23], the location of an object is modeled as a multi-dimensional uncertain object as shown in Fig 1, where an object A is represented by an uncertain region A_r and the probabilistic density function (PDF) $A.pdf$. This implies that an object may appear at the locations within its uncertain region with probabilities described by its PDF. In some scenerios, a manager of the warehouse may want to identify the objects within a search region e.g., the objects covered by a fire sprinkler (the circular region r_q in Fig. 1) or the objects within a particular stock keeping area. As shown in Fig. 1, it is meaningless to tell if the object A appears within search region r_q or not. Instead, the notation of *appearance probability* is proposed in [21] to capture the likelihood of A falling in r_q. As the manager may only be interested in the objects with a sufficiently large chance to fall in the search region, the system will report the objects with probability at least θ appearing in r_q where the probabilistic threshold θ is determined by the manager.

Another example application of range search on uncertain data is the location based services (LBS). The location of a mobile user can be described as an uncertain object, because the location may be derived based on the nearest contour lines of user's possible location regarding the nearby towers [4]. Based on the user's location information, the range search can help a supermarket to send advertising coupons via SMS to potential customers who are likely to be in the vicinity of the supermarket [19].

Motivated by the above applications, in the paper we study the problem of probabilistic threshold range query on multi-dimensional uncertain data; that is, given a set $\mathcal{U}$ of multi-dimensional uncertain objects and a search region r_q, report the objects with *appearance probability* at least θ. This is formally defined in Section 2.

Challenges. A straightforward approach for this problem

* Corresponding Author

EDBT 2012, March 26–30, 2012, Berlin, Germany.

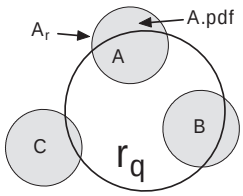


Figure 1: Example of range search on uncertain objects

is to calculate the *appearance probability* of each individual object and then report the ones whose *appearance probabilities* exceed the given probabilistic threshold. This is cost-inhibitive because the computation of the *appearance probability* may be very expensive because of the integral calculation or I/O costs involved. Therefore, all of the existing works [16, 20, 21, 25] on range search against multi-dimensional uncertain data with arbitrary PDF follow the *filtering-and-verification* paradigm such that, with the help of the indexing structures, many objects are *filtered* at a reasonable *filtering* cost without explicit calculation of their *appearance probabilities*. The key idea of the existing techniques is as follows: a summary (See details in Section 2.2 and Section 3.1) of the PDF is pre-computed for each uncertain object to approximately capture the distribution of its PDF, and the summaries are organized by augmenting existing index techniques (e.g., R-Tree). For a given search region r_q, the lower and upper bounds of the *appearance probability* can be derived at a cheap cost for each uncertain object. Then an uncertain object U may be *filtered* in two ways: (i) U is *pruned* if the upper bound of the *appearance probability* is smaller than the given probabilistic threshold θ, or (ii) U is *validated* (*qualified*) if the lower bound of the *appearance probability* is not less than θ. Only the objects survived the *filtering* phase need to be *verified*, i.e., explictly computing their *appearance probabilities*.

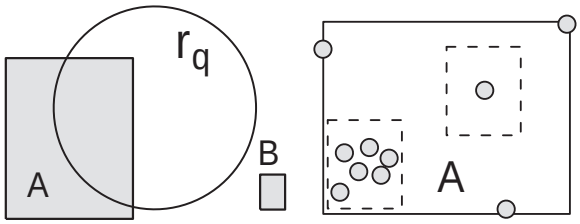


Figure 2: Region size

Figure 3: Diff. Densities

Motivation. Intuitively, the more resources (i.e., larger summary size in terms of space usage) assigned to the summary of an uncertain object U, the tighter the lower and upper *appearance probability* bounds of U and hence the lower expected *verification* cost of U. On the other hand, the *filtering* cost, which is usually dominated by the index I/O delay, increases with the storage overhead of the summaries. Therefore, the key of the index construction is to find a **good trade-off** between *filtering* cost and *verification* cost such that the overall cost is minimized. Based on the above observations, following are three important factors which may affect the resource allocation policies of the summaries during the index construction.

(i) Uncertain region size. Suppose there are two uncertain objects A and B as shown in Fig. 2 where the uncertain region size of A is much larger than that of B, intuitively we should allocate more resources to the PDF summary of A because, with much higher chance, A will survive the *filtering* phase and hence invoke *verification* cost. In an extreme case when the uncertain region of B is a point, it is meaningless to assign extra resources to B besides the location of the point.

(ii) Verification cost. The *verification* costs of the uncertain objects may vary due to different $PDFs$ and I/O costs involved. Intuitively, more resources should be allocated to the objects with higher *verification* costs. Moveover, even if all objects have the same *verification* cost, the *ratio* of the index I/O delay and the *verification* cost per object should be considered. This is because the higher the *verification* cost per object, the more expensive *filtering* cost (i.e., larger index size) we can afford since the gain of a successful validation/pruning is more significant.

(iii) Density of the PDF. As depicted in Fig. 3, the distribution of the possible locations of an uncertain object A may be uneven. Regarding the resource allocation for a particular uncertain object, intuitively the dense part of the object should have the priority because it contributes more to the *appearance probability* than the sparse part. This implies that we should carefully consider the density when building the summary of an uncertain object.

As discussed in Section 2, some of the existing works [21, 25] do consider the density factor during the index construction. However, the factors of uncertain region size and *verification* cost are neglected. This is because the same amount of resources in terms of the space usage are allocated to the summary of each uncertain object in the existing works. Consequently, this "equality strategy" may inherently limit the performance of the indexing techniques.

Motivated by these facts, in the paper we propose an effective and efficient indexing technique for the range search against the uncertain objects, in which above factors are carefully considered according to our quantitative analysis of the cost model for the range search. Our contributions can be summarized as follows.

- To facilitate the range search, we propose a novel technique called U-Quadtree to effectively index the multi-dimensional uncertain objects with arbitrary $PDFs$.
- Assume that the query distribution is known, we propose a cost model to quantitatively analyze the performance of the range search. Then we develop an efficient optimal U-Quadtree construction algorithm guided by the cost model. We also discuss how to construct the U-Quadtree when the query distribution is unknown.
- Comprehensive experiments demonstrate the efficiency of the U-Quadtree technique.

The remainder of the paper is organized as follows. We formally define the problem of range search and introduce the related work in Section 2. Section 3 presents the U-Quadtree structure and range search algorithm, followed by a cost model. Section 4 develops an optimal U-Quadtree construction algorithm based on the cost model. Results of comprehensive performance studies are presented in Section 5. Finally, Section 6 concludes the paper.

2. PRELIMINARY

In this section, we first formally define the problem of range search on multi-dimensional uncertain objects. Then

we introduce the related works. Table 1 summaries notations frequently used throughout the paper.

Notation	Definition
U ($\mathcal{U}$)	uncertain object (a set of uncertain objects)
n	the number of uncertain objects
r_q	range search region
$P_{app}(U, r_q)$	the *appearance probability* of U regarding r_q
θ	probabilistic threshold
c, $l(c)$	a cell of the U-Quadtree, the level of c
h	the height of the U-Quadtree
f	entry page capacity
$\varphi(c)$	cell c and its descendent cells
$\chi(c)$	child cells of the cell c
$x \in c$	instance x is *contained* by cell c
t_U	the verification cost of U
$P_c(c)(P_o(c))$	containing (overlapping) probability of c
$F_c(V_c)$	the filtering (verification) factor of c
S_U, S_U^*	summary of U, optimal summary of U
$\omega(S_U)$	the cost of the summary S_U
$\omega(e)$	the cost of the entry e
$x \looparrowright S_U(c)$	instance x is assigned to cell c in S_U

Table 1: The summary of notations.

2.1 Problem Definition

A point (instance) x referred in the paper, by default, is in a d-dimensional numerical space. In the paper, an uncertain object is represented by its possible locations (points) and the probability it may appear at each location. Particularly, an uncertain object can be described either *continuously* or *discretely*. In the *continuous* case, an uncertain object U is described by its *probability density function*(PDF), denoted by $U.pdf$, and its uncertain region U_r. The probability of U appearing at location x is $U.pdf(x)$ and we have $\int_{x \in U_r} U.pdf(x)dx = 1$. Note that we assume $PDFs$ of the uncertain objects are mutually independent and various objects may have different $PDFs$ and uncertain regions. In some applications, the PDF of the uncertain object may not be available and hence an uncertain object is represented by a set of sampled points (*discrete* case); that is, an uncertain object consists of a set of instances (points) u_1, u_2, ..., u_m , where U occurs at location u_i with probability u_{ip} and $\sum_{u \in U} u_p = 1$. For presentation simplicity, the "object" referred in the rest of the paper is the "multi-dimensional uncertain object" unless otherwise specified.

In the paper, U_{mbb} denotes the minimal bounding box (MBB) of the instances of an object U. For a point p and a region r, $p \in r$ means that p is *contained* by r. For any two regions r_1 and r_2, r_1 **contains** r_2 (r_2 is **contained** by r_1), denoted by $r_2 \subseteq r_1$, if $r_1 \cup r_2 = r_1$. We say r_1 **overlaps** r_2 (r_2 is **overlapped** by r_1) if $r_2 \not\subseteq r_1$ and $r_1 \cap r_2 \neq \emptyset$.

For a given search region r_q, we use $P_{app}(U, r_q)$ to represent the likelihood of U falling in r_q, called *appearance probability* of U regarding r_q, which is formally defined below.

For *continuous* cases,

$$P_{app}(U, r_q) = \int_{x \in U_r \cap r_q} U.pdf(x)dx \quad (1)$$

For *discrete* cases,

$$P_{app}(U, r_q) = \sum_{u \in U \ \wedge \ u \in r_q} u_p \quad (2)$$

For presentation simplicity, we concentrate on the *discrete* case in the paper. Nevertheless, all techniques developed in the paper can be immediately applied to *continuous* case.

Below is the definition of "probabilistic threshold range query", which is abbreviated to "range query" in the rest of the paper.

Definition 1. ***Probabilistic Threshold Range Query.*** *Given a set $\mathcal{U}$ of uncertain objects and a search region r_q, the probabilistic threshold range query retrieves all objects $U \in \mathcal{U}$ with $P_{app}(U, r_q) \geq \theta$ where θ $(0 < \theta \leq 1)$ is the user specified probabilistic threshold.*

Example 1. *In Fig 4, suppose objects have four instances each and all instances have the same occurrence probability* (0.25)*. According to Equation 2, we have $P_{app}(A, r_q) = 0.5$, $P_{app}(B, r_q) = 0.75$ and $P_{app}(C, r_q) = 0.25$. For the given range query q with search region r_q and the probabilistic threshold* 0.5*, objects A and B will be reported.*

Problem Statement.
In this paper we investigate the problem of probabilistic threshold range query over multi-dimensional uncertain objects with arbitrary $PDFs$. Particularly, we aim to develop effective indexing mechanism to facilitate the range query processing.

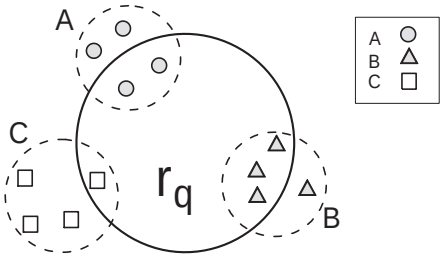


Figure 4: Appearance Probability

2.2 Related Work

In this subsection, we first present four indexing techniques supporting range search on multi-dimensional uncertain objects with arbitrary $PDFs$, including *R-Tree*, *U-Tree*, *UI-Tree* and *UP-Index*. Then we briefly introduce other related indexing techniques for uncertain data.

***R*-Tree.** In R-Tree approach [16, 20, 17], the MBB of an object serves as the summary of its PDF, and MBBs are organized by R-Tree. An uncertain object U can be *validated* if its MBB is contained by r_q, i.e., $U_{mbb} \subseteq r_q$, regardless of the value of the probabilistic threshold θ. Similarly, U is *pruned* if U_{mbb} does not intersect r_q, i.e., $U_{mbb} \cap r_q = \emptyset$. This approach is simple and performs well if the uncertain region sizes are much smaller than r_q. However, as the MBB cannot further explore the PDF of an uncertain object, the filtering capacity of the index is poor when the size of the uncertain region is not small. As shown in Fig. 5, for a given search region r_q, we cannot *prune* A regarding any probabilistic threshold θ although intuitively $P_{app}(A, r_q)$ should be small. Similarly, B cannot be *validated* either.

***U*-Tree.** The PDF summary of an object in U-Tree is a finite set of probabilistically constrained regions (PCRs), which is introduced by Tao *et al.* in [21]. PCR is a general version of *x-bounds* which aims to index one dimensional uncertain data [9]. For a given θ $(0 \leq \theta \leq 0.5)$, the PCR of an object U regarding θ, denoted by $U.pcr(\theta)$, is constructed as

follows. As shown in Fig. 6, in each dimension, two lines are calculated. In the horizontal dimension, U has the probability θ to occur on the left side of line l_{1-}, also probability θ to occur on the right side of line l_{1+}. Similarly, l_{2-} and l_{2+} are calculated in the vertical dimension. The shaded region in Fig. 6 is the geometric representation of $U.pcr(\theta)$. Then we can take advantage of $U.pcr(\theta)$ to *prune* or *validate* U regarding θ and r_q. For instance, as shown in Fig. 6, suppose θ is the probabilistic threshold for two search regions r_q^1 and r_q^2. U can be *pruned* regarding r_q^1 because r_q^1 does not intersect $U.pcr(\theta)$. On the other hand, U can be *validated* with respect to r_q^2 since all instances below l_{2-} are contained by r_q^2. As it is infeasible to keep all $U.pcr(\theta)$ for any $\theta \in [0, 0.5]$, a finite number of PCRs are pre-computed for each object and the lower and upper *appearance probability* bounds can be derived. Based on the PCRs of the uncertain objects, U-Tree is built up in a similar way with R-Tree where each entry in a leaf node corresponds to the $PCRs$ of an uncertain object.

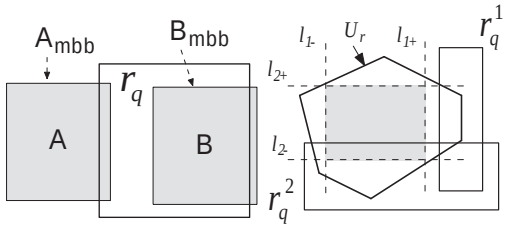


Figure 5: MBB

Figure 6: PCR

UI-Tree. The PDF summary of an object U in [25] is a set of groups which are disjointed partitions of its PDF based on a KD-Tree. Given a search region r_q, we can derive the lower and upper bounds of $P_{app}(U, r_q)$ based on the topological relationships between the groups and r_q. Specifically, groups contained by r_q contribute to both lower and upper bounds of the *appearance probability* since all instances in these groups are contained by r_q. With similar rationale, groups *overlapped* by r_q only contribute to upper bound. Then an object U may be *validated* (*pruned*) based on the lower (upper) bound of $P_{app}(U, r_q)$. For the space efficiency, the groups of the uncertain objects may be merged such that a set of groups from different objects can share the same boundary, namely "word" in [25]. The identifications of the related objects and their corresponding probability mass are kept in each "word". Then UI-Tree is constructed in a similarly way with R-Tree where each entry of a leaf node is a "word".

UP-Index. Recently, Angiulli *et al.* [3] develop a pivot based indexing mechanism for uncertain data in general metric space. For a given pivot point p and an object U, the PDF summary of U is the histogram of the distance distribution regarding p and U. The upper bound of $P_{app}(U, r_q)$ can be derived based on the *reverse triangle inequality* according to the histogram and the distance between the centre of r_q and the pivot point p. Then an object can be pruned based on the upper bound derived. To enhance the pruning power, a set of pivot points are employed in [3]. The advantage of UP-Index is that it can support distance based range query in general metric space. Nevertheless, as shown in our empirical study, its performance is not competitive under our problem setting because : (i) an object cannot be *validated* in [3] because UP-Index cannot derive the lower bound of $P_{app}(U, r_q)$, and (ii) for any range search the whole index is scanned to prune objects, and the index size is usually large for a decent pruning capacity.

	R-Tree [16]	*U-Tree [21]*	*UI-Tree [25]*	*UP-Index [3]*	*U-Quadtree*
Region size	×	×	×	×	✓
Verification	×	×	×	×	✓
Density	×	✓	✓	✓	✓

Table 2: Summary of the indexing techniques

Table 2 illustrates the properties of the indexing techniques regarding three factors mentioned in Section 1. Clearly, R-Tree based approach considers none of the factors since only one MBB is allocated for the PDF summary of each object. U-Tree, UI-Tree and UP-Index techniques consider the density factor because they attempt to evenly distribute the accumulated probabilities in the summary of an object U, which implies that more resources are allocated for the dense parts of U. However, the same amount of resources in terms of the space usage are allocated to the summary of each object in these techniques. Consequently, they cannot effectively address different uncertain region sizes and *verification* costs during the index construction. In Section 4, we show that U-Quadtree technique proposed in the paper carefully considers all three factors.

Others. There are also some studies on indexing multi-dimensional uncertain objects which focus on specific cases of objects' $PDFs$ and queries. For instances, in [5, 6, 8, 13], Böhm *et al.* study range queries with the constraint that $PDFs$ of uncertain objects follow *Gaussian* or *uniform* distributions. Assuming $PDFs$ of the objects are either histograms or more complex ones such as *Gaussian* or piecewise algebraic, in [1] Agarwal *et al.* provide thorough theoretical analysis on range search on uncertain data. Managing uncertain moving objects [24] and uncertain categorical data [20] have been separately studied. Aggarwal *et al.* [2] study the problem of indexing high dimensional uncertain data with the assumption that the $PDFs$ of the uncertain object on each dimension are independent to others. Assuming the space is partitioned by a *virtual grid* with limited number of cells, Ma *et al.* [18] propose solutions for efficient retrieval of uncertain spatial point data where the location information is derived from the free text by *spatial expressions*. Recently, Kinura *et al.* [15] propose a primary indexing technique named UPI to speed up the query processing on uncertain data by clustering the heap files, in which U-Tree [21] technique is used as a building block to index uncertain objects with arbitrary $PDFs$. In [17], Lian *et al.* propose a generic framework to index uncertain data. Their main focus is how to tackle the local correlations among uncertain objects, and their indexing technique falls in the R-Tree category.

3. U-QUADTREE

In the paper, our index structure is based on the quadtree because it is a flexible data structure in the sense that we can adaptively build summaries of the objects so that the overall cost of range searching can be minimized regarding the cost model proposed in the paper. Section 3.1 formally introduces U-Quadtree structure, and Section 3.2 presents

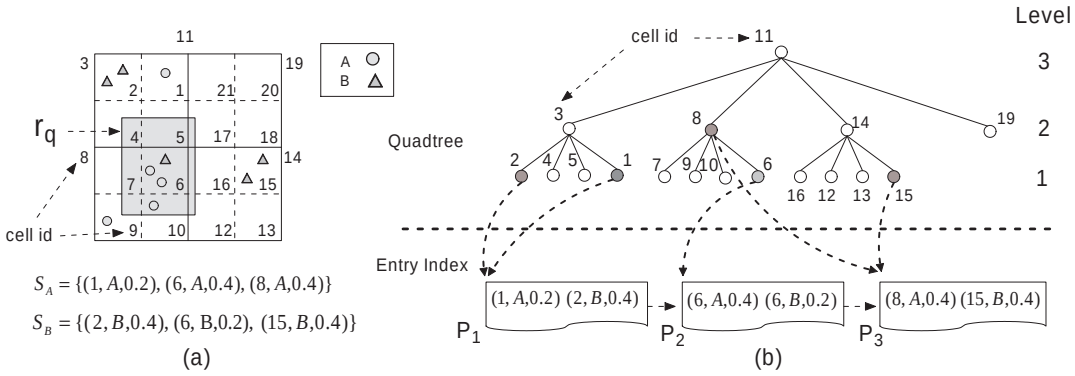


Figure 7: U-Quadtree

the U-Quadtree based range search algorithm. Section 3.3 proposes the cost model for the range search on U-Quadtree.

3.1 U-Quadtree Structure

A quadtree [12] is a space partitioning tree data structure in which a d-dimensional space is recursively subdivided into 2^d regions (cells). Due to its simplicity and regularity, the quadtree technique has been widely applied in many applications. In the paper, we focus on 2-dimensional space and all techniques developed can be immediately applied to higher dimensional spaces.

Given a quadtree, a **summary** of an object U is defined as follows.

DEFINITION 2 (S_U). *A **summary** S_U of an object U regarding a quadtree consists of a set of entries $\{e\}$ where each entry e is a triplet $(e.c, e.o, e.p)$ where $e.c$ and $e.o$ represent the identification (id) of the cell and the object associated with the entry, and $e.p$ $(0 < e.p \leq 1)$ is the probability mass of the instances assigned to this entry (i.e., the cell $e.c$). For any instance $x \in U$, x must be assigned to **exactly one** cell c (entry e) where $x \in c$ $(e.c)$ and hence $\sum_{e \in S_U} e.p = 1$.*

In Fig. 7(a), objects A and B have 5 instances each and all instances have the same occurrence probability (0.2). The height of the quadtree (h) is 3 and the ids of the cells are labeled. We may have $S_A = \{(1, A, 0.2), (6, A, 0.4), (8, A, 0.4)\}$ and $S_B = \{(2, B, 0.4), (6, B, 0.2), (15, B, 0.4)\}$. Note that the summary of an object **is not unique** as an instance x can be assigned to any cell which *contains* x. For instance, an alternative of S_A could be $\{(1, A, 0.2), (8, A, 0.8)\}$ in which 4 instances of A are assigned to cell 8 on level 2. Section 4 will investigate how to efficiently build optimal summaries.

As shown in Fig. 7(b), entries of the objects are organized based on a quadtree, named U-Quadtree, which consists of two parts:

- **Entry Index** (UQ_E): a B^+ tree used to keep entries of the objects in the secondary memory, where the key of each entry is its cell *id*. Similar to [14], we assume the *id* of a cell is its Hilbert code [11] generated in a recursive way such that the cells with close spatial proximity are likely to be allocated to the same or adjacent pages in UQ_E. Particularly, a leaf node of UQ_E is called the *entry page* (e.g., P_1, P_2 and P_3 in Fig. 7(b)) and f denotes its capacity (i.e., the maximal number of entries in an entry page).
- **Quadtree** (UQ_T): a pointer-based quadtree with height h. For each cell (node) c, let P be the first entry page in which there is an entry e with $e.c = c$. We keep the address of P as the pointer of cell c. As shown in Fig. 7(b), a gray cell (node) of UQ_T implies the pointer of the cell is not empty, i.e., there is at least one entry on it. Note that we do not need to keep a cell in UQ_T if none of its descendent cells including itself contains any entry.

3.2 U-Quadtree Based Range Search

The following theorem indicates that we can derive the lower and upper bounds of $P_{app}(U, r_q)$, denoted by $LP_{app}(U, r_q)$ and $UP_{app}(U, r_q)$ respectively, based on the topological relations between r_q and cells associated with S_U.

THEOREM 1. *Given an object summary S_U and a search region r_q, let C_1 (C_2) denote the cells in S_U which are contained (overlapped) by r_q, we have*

$$LP_{app}(U, r_q) = \sum e.p, \text{ where } e \in S_U \text{ and } e.c \in C_1$$

$$UP_{app}(U, r_q) = \sum e.p, \text{ where } e \in S_U \text{ and } e.c \in C_1 \cup C_2$$

PROOF. For any point $x \in c$, we have $x \in r_q$ if c is *contained* by r_q. It is immediate that $P_{app}(U, r_q) \geq \sum e.p$ where $e \in S_U$ and $e.c \in C_1$. Given a point $x \in c$, if c is not *contained* or *overlapped* by r_q, then we have $x \notin r_q$. This implies $P_{app}(U, r_q) \leq 1 - \sum e.p$ where $e \in S_U$ and $e.c \notin C_1 \cup C_2$. Because $\sum_{e \in S_U} e.p = 1$, we have $P_{app}(U, r_q) \leq \sum e.p$ where $e.c \in C_1 \cup C_2$. Therefore, the theorem holds. □

EXAMPLE 2. *In Fig. 7(a), given a search region r_q (shaded region), according to Theorem 1 we have $LP_{app}(A, r_q) = 0.4$ and $UP_{app}(A, r_q) = 0.8$ because the cell 6 is contained by r_q and the cell 8 is overlapped by r_q. Consequently, A is pruned if the probabilistic threshold θ equals 0.9, and A is validated if $\theta = 0.3$. We need to verify A if $\theta = 0.5$ since we cannot prune or validate the object A.*

Algorithm 1 illustrates the details of the range search following the *filtering-and-verification* paradigm. Line 2 retrieves a set $\mathcal{J}$ of non-empty cells which are *contained* or *overlapped* by r_q. Let $\mathcal{D}$ denote the entry pages $\{P\}$ associated with those cells, i.e., $\mathcal{D} = \{P | \exists e \in P \text{ s.t. } e.c \in \mathcal{J}\}$. Regarding the example in Fig. 7, $\mathcal{D} = \{P_2, P_3\}$. As we access entry pages in sequence, the total number of I/O incurred in Line 4 is $|\mathcal{D}|$. According to Theorem 1, we can come up with the lower and upper bounds of the *appearance probabilities* of the objects. Note that any unvisited object has *appearance probability* zero. It is immediate that we can *validate* an object U if $LP_{app}(U, r_q) \geq \theta$ (Line 12). Similarly an object U is *pruned* if $UP_{app}(U, r_q) < \theta$ (Line 14). In Line 17, we only need to *verify* the remaining objects which are not *pruned* or *validated* during the *filtering* phase.

Algorithm 1: ***Range Query***(UQ, r_q, θ)

Input : UQ : the U-Quadtree of a set $\mathcal{U}$ of objects,
r_q : search region,
θ : probabilistic threshold
Output : $\mathcal{R}$: objects $U \in \mathcal{U}$ with $P_{app}(U, r_q) \geq \theta$

```
1  R := ∅; C := ∅ ;
2  J := all non-empty cells contained or overlapped by r_q ;
3  D := entry pages associated with c ∈ J ;
4  for each entry e ∈ D do
5      U ← e.o ;
6      if e.c ∈ J then
7          UP_app(U, r_q) := UP_app(U, r_q) + e.p ;
8          if cell e.c is contained by r_q then
9              LP_app(U, r_q) := LP_app(U, r_q) + e.p ;
10 for each visited object U do
11     if LP_app(U, r_q) ≥ θ then                    /* validation */
12         R := R ∪ U ;
13     else
14         if UP_app(U, r_q) ≥ θ then                /* pruning */
15             C := C ∪ U;
16 for each U ∈ C do
17     if P_app(U, r_q) ≥ θ then                     /* verification */
18         R := R ∪ U;
19 return R
```

3.3 Cost Model

The cost of Algorithm 1 consists of four parts:

1. C_s(Line 2), the cost to retrieve the cells *contained* or *overlapped* by r_q , which is $O(\frac{4^h-1}{3})$ in the worst case.
2. C_{index} (Line 4), the cost to load entry pages in $\mathcal{D}$, which equals $|\mathcal{D}| \times t_{io}$ where t_{io} is the delay of an I/O access.
3. C_{prob} (Lines 5-9), the cost to compute lower and upper bounds of the *appearance probabilities*. As a hash table is employed to maintain the objects visited, $C_{prob} = \sum_{U \in \mathcal{J}} |S_U|$ where $\mathcal{J}$ represents the visited objects and $|S_U|$ is the number of entries in S_U.
4. C_{verify} (Line 17), the cost to *verify* the remaining objects in $\mathcal{C}$ where $C_{verify} = \sum_{U \in \mathcal{C}} t_U$, where t_U denotes the *verification* cost of U.

The dominant costs of the range search are the index I/O cost (C_{index}) and the verification cost (C_{verify}). In our empirical study, they contribute more than 98% of the cost for the range search. Therefore, we only consider C_{index} and C_{verify} in our cost model, which represent the *filtering* cost and *verification* cost of the Algorithm 1 respectively.

Suppose the query distribution $D_{\mathcal{Q}}$ is known (e.g., query log is available), below we define the *containing probability* and *overlapping probability* of a cell c regarding $D_{\mathcal{Q}}$.

DEFINITION 3. ***Containing probability*** $P_c(c)$ *and* ***Overlapping probability*** $P_o(c)$. *Suppose r_q in Algorithm 1 is randomly chosen from a query distribution $D_{\mathcal{Q}}$, the containing (overlapping) probability, namely $P_c(c)$ ($P_o(c)$), of a cell c is the likelihood of c being contained (overlapped) by r_q.*

Learn $P_c(c)$ and $P_o(c)$. Given a query distribution $D_{\mathcal{Q}}$, we can randomly retrieve n_q search regions from $D_{\mathcal{Q}}$ and record the number of times in which the cell c is *contained* or *overlapped*, denoted by $n_c(c)$ and $n_o(c)$ respectively. Then we have $P_c(c) = \frac{n_c(c)}{n_q}$ and $P_o(c) = \frac{n_o(c)}{n_q}$.

Now, we estimate the expected *filtering* cost and *verification* cost assuming the query distribution is known.

Index I/O cost. For an entry page P loaded in Algorithm 1, we say an entry e in P is *non-redundant* regarding r_q if $e.c$ is *contained* or *overlapped* by r_q, and e is *redundant* otherwise. For instance, in Fig. 7(b) the entry $e(8, A, 0.4)$ ($e(15, B, 0.4)$) in P_3 is a non-redundant (redundant) entry regarding r_q. In the paper, f_n denotes the average number of non-redundant entries for the entry pages loaded. An entry page will be loaded in the main memory if any of its associated cell is *contained* or *overlapped* by r_q. The expected number of index I/Os incurred by an entry e is $(P_o(e.c) + P_c(e.c)) \times \frac{1}{f_n}$ since all non-redundant entries in the same entry page share the I/O latency. We can estimate the expected index I/O cost, namely $E(C_{index})$, as follows because PDFs of the objects are mutually independent.

$$E(C_{index}) = |\mathcal{D}| t_{io} = \sum_{U \in \mathcal{U}} \sum_{e \in S_U} (P_o(e.c) + P_c(e.c)) \times \frac{t_{io}}{f_n} \quad (3)$$

It is immediate that there is no redundant entry regarding r_q in an entry page P if all entries in P are from the same cell, which is likely when the number of uncertain objects is large. Moreover, since entries are ordered by the Hilbert codes (*ids*) of their corresponding cells, an entry page is likely to be shared by the cells with close spatial proximity. Consequently, the number of redundant entries is small because the cells in the same page are likely to be *contained* or *overlapped* by r_q at the same time. Therefore, we set f_n to f in the paper and Equation 3 can be rewritten as follows. Recall that f is the capacity of an entry page.

$$E(C_{index}) = \sum_{U \in \mathcal{U}} \sum_{e \in S_U} (P_o(e.c) + P_c(e.c)) \times \frac{t_{io}}{f} \quad (4)$$

Verification Cost. In Algorithm 1, an object U will be *verified* if U is not *pruned* or *validated*, i.e., $LP_{app}(U, r_q) < \theta \leq UP_{app}(U, r_q)$. Suppose the probabilistic threshold θ is randomly chosen from $(0, 1]$, U will be *verified* with probability Δ, where $\Delta = UP_{app}(U, r_q) - LP_{app}(U, r_q)$. According to Theorem 1, this implies that only the entries whose associated cells *overlapped* by r_q contribute to the *verification* cost, and hence $\Delta = \sum_{e \in S_U} P_o(e.c) \times e.p$. Then based on the PDF independence assumption we have the following estimation for the expected *verification* cost, denoted by $E(C_{verify})$.

$$E(C_{verify}) = \sum_{U \in \mathcal{U}} \sum_{e \in S_U} P_o(e.c) \times e.p \times t_U \quad (5)$$

Based on Equation 4 and Equation 5 we come up with the expected cost, denoted by $E(C)$, of the range query.

$$E(C) = \sum_{U \in \mathcal{U}} \sum_{e \in S_U} ((P_o(e.c) + P_c(e.c)) \times \frac{t_{io}}{f} + P_o(e.c) \times e.p \times t_U) \quad (6)$$

REMARK 1. ***About*** t_U. *As shown in [21, 25], the dominant cost of the verification comes from the sampling of the PDF for the continuous case or the I/O cost incurred for the discrete case. The sampling cost can be estimated based on the required accuracy [21] (i.e., the number of samples), while the I/O cost can be estimated by the number of pages occupied by the object.*

In the paper, we say a U-Quadtree is **optimal** regarding the cost model if $E(C)$ is minimized. In Section 4, we investigate how to efficiently construct the optimal U-Quadtree regarding the cost model so that the overall cost of the range search can be minimized.

4. BUILD U-QUADTREE

A nice property of the cost model is that we can build the optimal U-Quadtree by optimizing the cost of the summary for each individual object. In Section 4.1, we first propose an algorithm to compute the optimal summary of an object U based on the dynamic programming technique. Then Section 4.2 introduces the U-Quadtree maintenance algorithms. Section 4.3 discusses how to construct the U-Quadtree when the query distribution is unknown.

4.1 Optimization of S_U

For presentation simplicity, we use F_c and V_c to denote the **filtering factor** and **verification factor** of cell c regarding U where $F_c = (P_o(c) + P_c(c)) \times \frac{t_{io}}{f}$ and $V_c = P_o(c) \times t_U$. Note that in the subsection we assume the query distribution is known and hence $P_c(c)$ and $P_o(c)$ are computed. In the paper, $\omega(e)$ denotes the cost of an entry e where $\omega(e) = F_{e.c} + V_{e.c} \times e.p$ if $e.p > 0$, and otherwise $\omega(e) = 0$. Note that e is a dummy entry if $e.p = 0$. The cost of a summary S_U, denoted by $\omega(S_U)$, is $\sum_{e \in S_U} \omega(e)$. We say a summary of U is **optimal**, denoted by S_U^*, if it has the minimal cost among all possible summaries of U regarding the quadtree.

In this subsection, we develop an efficient algorithm to construct the optimal summary of an object.

We first introduce some important properties. The following property is immediate based on the definition of the quadtree.

PROPERTY 1. *For any two cells c_1 and c_2 in a U-Quadtree, c_1 is a descendant cell of c_2 if and only if $c_1 \subset c_2$, where $c_1 \subset c_2$ implies $c_1 \subseteq c_2$ and $c_1 \neq c_2$.*

Given two cells c_1 and c_2 with $c_1 \subset c_2$, if c_1 is *overlapped* by r_q then c_2 is *overlapped* as well. This implies $V_{c_1} \leq V_{c_2}$. With similar rationale, we have $F_{c_1} \leq F_{c_2}$. Without loss of generality, we assume the following property holds.

PROPERTY 2. *Given two cells c_1 and c_2, $V_{c_1} < V_{c_2}$ and $F_{c_1} < F_{c_2}$ if $c_1 \subset c_2$.*

Given a summary S_U and a cell c, $S_U(c)$ denotes the instances of U assigned to cell c in S_U. We say a cell c is *empty* regarding S_U if $S_U(c) = \emptyset$, otherwise c is *occupied*. For an instance $x \in U$, $x \leftarrow S_U(c)$ denotes the fact that x is assigned to c in S_U. Clearly, $x \leftarrow S_U(c)$ implies $x \in c$, but **not vice versa**. The following theorem implies that once an instance x is assigned to a cell c in S_U^*, all descendant cells of c which contain x must be *empty*.

THEOREM 2. *Given the optimal summary S_U^* and two cells c_1 and c_2 where $c_1 \subset c_2$, for any two instances x and y where $x \in c_1$ and $y \in c_1$, we have $y \not\leftarrow S_U^*(c_2)$ if $x \leftarrow S_U^*(c_1)$.*

PROOF. The proof is by contradiction. Let e_1 and e_2 denote two entries of S_U^* on c_1 and c_2 respectively. Suppose we have $x \leftarrow S_U^*(c_1)$ and $y \leftarrow S_U^*(c_2)$. This implies both cells are not *empty*. Let $X = \omega(e_1) + \omega(e_2) = F_{c_1} + V_{c_1} \times e_1.p + F_{c_2} + V_{c_2} \times e_2.p$. Since $y \in c_1$, we can move y from c_2 to c_1, which leads to anther summary of U denoted by S'_U. Let e'_1 and e'_2 be two entries of S'_U on c_1 and c_2 respectively. The costs of e'_1 and e'_2, denoted by Y, is less than or equal to $F_{c_1} + V_{c_1} \times (e_1.p + y_p) + F_{c_2} + V_{c_2} \times (e_2.p - y_p)$. Since $c_1 \subset c_2$, we have $V_{c_1} < V_{c_2}$ according to the Property 2 and hence $Y < X$. As the costs of other entries in S'_U remain unchanged, we have $\omega(S'_U) < \omega(S_U^*)$, which is against the assumption. □

Following Property is immediate based on Theorem 2.

PROPERTY 3. *Let $c_{1,i}$ be a cell on level 1, for any two instances x and y where $x \in c_{1,i}$ and $y \in c_{1,i}$, x and y will always be assigned to the same cell in S_U^*; that is, $x \leftarrow S_U^*(c)$ implies $y \leftarrow S_U^*(c)$ and vice versa.*

Property 3 indicates that for the optimal summary construction, an object can be regarded as l *virtual* instances $\{u_i\}$ where u_i corresponds to the instances contained by $c_{1,i}$; that is, $u_i = \{x | x \in U \text{ and } x \in c_{1,i}\}$. In Fig. 8, 10 instances of U have the same occurrence probability (0.1) and they correspond to 4 *virtual* instances. In S_U^*, each *virtual* instance u must be assigned to **exactly one** cell which contains u, and there are at most h target cells for u. Consequently, a straightforward solution for the optimal summary construction is to enumerate all possible choices of each *virtual* instance, which takes $O(l^h)$ time where l is the number of *virtual* instances.

To further improve the efficiency of the optimal summary construction, we develop an efficient algorithm based on the dynamic programming technique. Below, we start with a motivating example.

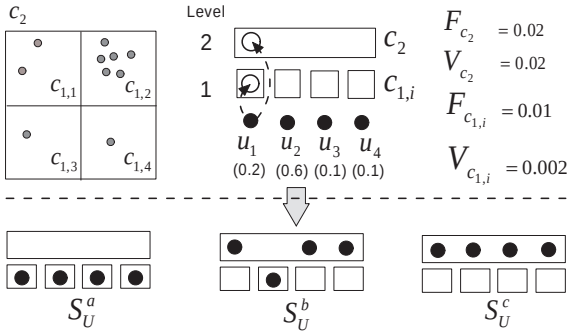


Figure 8: Motivating Example

Motivating Example. In Fig. 8, we assume that $F_{c_2} = 0.02$, $V_{c_2} = 0.02$, $F_{c_{1,i}} = 0.01$, and $V_{c_{1,i}} = 0.002$ for any $i \in [1, 4]$. We show three possible forms of S_U where $S_U^a = \{(c_{1,1}, 0.2), (c_{1,2}, 0.6), (c_{1,3}, 0.1), (c_{1,4}, 0.1)\}$, $S_U^b = \{(c_2, 0.4), (c_{1,2}, 0.6)\}$ and $S_U^c = \{(c_2, 1.0)\}$. Note that we ignore object *ids* in the entries since all instances are from U in this subsection. Let $e_1 = (c_2, 0.4)$ and $e_2 = (c_{1,2}, 0.6)$, we have $\omega(S_U^b) = \omega(e_1) + \omega(e_2) = F_{c_2} + V_{c_2} \times 0.4 + F_{c_{1,2}} + V_{c_{1,2}} \times 0.6 = 0.0392$. Similarly, we have $\omega(S_U^a) = 0.042$ and $\omega(S_U^c) = 0.04$, and hence S_U^b wins out. This implies that some virtual instances can be pushed to cells at higher levels so that the cost of the summary is reduced. we use X ($X \neq \emptyset$) to denote the set of virtual instances which are pushed to c_2, and S_U^X is the corresponding summary. Compared with S_U^a in which none of the virtual instance is pushed to c_2, the cost reduced by S_U^X, denoted by Δ_X, is as follows.

The cost of e'_2 is *zero* if y is the only instance assigned to c_2 in S_U^*.

$$\Delta_X = \omega(S_U^a) - \omega(S_U^X) = (\sum_{u_i \in X} (F_{c_{1,i}} - (V_{c_2} - V_{c_{1,i}}) \times u_{ip})) - F_{c_2} \quad (7)$$

According to Equation 7, we can see the advantage of pushing instances in X to c_2 is that these instances can share the *filtering* cost (F_{c_2}), and hence reduce their *filtering* costs if $F_{c_2} < \sum_{u_i \in X} F_{c_{1,i}}$. On the other hand, the disadvantage is that the *verification* cost involved is increased since $V_{c_2} > V_{c_{1,i}}$. We use $\Delta(u_i)$ to denote the gain of pushing the virtual instance u_i to c_2 without considering the *filtering* cost on c_2, where $\Delta(u_i) = F_{c_{1,i}} - (V_{c_2} - V_{c_{1,i}}) \times u_{ip}$. Recall that u_{ip} is the occurrence probability of u_i. Particularly, we have $\Delta(u_1) = 0.0064$, $\Delta(u_2) = -0.0008$, $\Delta(u_3) = \Delta(u_4) = 0.0082$. Let $g(X) = \sum_{u \in X} \Delta(u)$ denote the total gains of X, we have $\Delta_X = g(X) - F_{c_2}$. It is worthwhile to push all virtual instances in X to c_2 if $g(X) > F_{c_2}$ because the gain can pay off the *filtering* cost on c_2 (F_{c_2}), and hence $\omega(S_U^X) < \omega(S_U^a)$. To minimize the cost of S_U^X (i.e., maximize the $g(X)$), we should push all virtual instances $\{u\}$ with $\Delta(u) > 0$; that is, $X = \{u_1, u_3, u_4\}$ and $g(X) = 0.0228 > F_{c_2}$, which corresponds to the summary S_U^b. If we have $g(X) < F_{c_2}$ for any $X \neq \emptyset$, none of the virtual instances should be pushed to c_2 because $\omega(S_U^X) > \omega(S_U^a)$, which is the case if we set $F_{c_2} = 0.03$ in Fig. 8.

Before presenting the optimal summary construction algorithm, we first introduce some important notations.

DEFINITION 4 (S_c : **summary of cell** c). *Let $\varphi(c)$ represent the cell c and its descendant cells, the summary of a cell c, denoted by S_c, is a set of entries associated with cells in $\varphi(c)$ such that **any** instance contained by c must be assigned to **exactly one** cell in $\varphi(c)$.*

In the paper, $\omega(S_c)$ corresponds to the sum of the costs of the entries in S_c, i.e., $\omega(S_c) = \sum_{e \in S_c} \omega(e)$. Specifically, we use S_c^* to represent the optimal summary of c which has the smallest $\omega(S_c)$ value. Let $l(c)$ denote the level of a cell c in UQ_T, S_c^* corresponds to S_U^* if $l(c) = h$.

We say an instance x is **pushed up above** c if $x \in c$ and x is assigned to a cell at higher level than c. For instance, in Fig. 8 u_1 is pushed up above $c_{1,1}$ in S_U^b and S_U^c. The following property indicates that all instances pushed up above c in S_U^* must be assigned to the same cell. The correctness of the Property is immediate based on Theorem 2.

PROPERTY 4. *If an instance x is pushed up above c in S_U^*, i.e., $x \in c$ and $x \nrightarrow S_U^*(c_j)$ where $c \subset c_j$, then $y \nrightarrow S_U^*(c_j)$ for any instance y pushed up above c in S_U^*.*

Property 4 is essential to our optimal summary construction algorithm, because it suggests that for each c we can evaluate the possible gain of pushing up above c against its **individual** ancestor cell, i.e., cells $\{c_j\}$ with $c \subset c_j$ according to Property 1. Consequently, for each ancestor cell c_j of c ($c \subset c_j$), we use S_{c,c_j} to denote the summary of c regarding c_j which is defined as follows.

DEFINITION 5 (S_{c,c_j} : **summary of** c **regarding** c_j). *The summary of a cell c regarding c_j where $c \subseteq c_j$, denoted by S_{c,c_j}, is a set of entries associated with the cells in $\varphi(c)$ and c_j such that **any** instance contained by c must be assigned to **exactly one** cell in $\varphi(c)$ **or** c_j.*

In the paper, $\omega(S_{c,c_j})$ corresponds to the total costs of the entries in S_{c,c_j} **without** considering the *filtering* cost of the entry on c_j (denoted by e_{c_j}), i.e., $\omega(S_{c,c_j}) = (\sum_{e \in (S_{c,c_j} - e_{c_j})} \omega(e)) + V_{c_j} \times e_{c_j}.p$. Moreover, we use $\Delta(S_{c,c_j})$ to denote the **gain of** c **regarding** c_j, where $\Delta(S_{c,c_j}) = \omega(S_c^*) - \omega(S_{c,c_j})$. In the paper, S_{c,c_j}^* denotes a set of entries with the smallest $\omega(S_{c,c_j})$ value (i.e., the largest $\Delta(S_{c,c_j})$ value), which is named the optimal summary of c regarding c_j. Recall that the main difference between S_{c,c_j} and S_c is that the instances contained by c can be pushed to c_j in S_{c,c_j} and the cost of S_{c,c_j} ($\omega(S_{c,c_j})$) does not include the *filtering* cost on c_j. Note that we have $\Delta(S_{c,c_j}^*) \geq 0$ because S_c is a special case of S_{c,c_j} where none of the instances contained by c is assigned to c_j.

Algorithm 2: ***BuildOptSummary***(UQ_T, U)

Input : UQ_T : the quadtree of the U-Quadtree,
U : the object for summary construction
Output: S_U^* : the optimal summary of U

```
1  Find the minimal cell c' in UQ_T which contains U_mbb;
2  H := l(c') ;
3  for each cell c ⊆ c' on level one do
4      create a virtual instance u regarding c ;
5      S*_c := {(c, u_p)} ;
6      for each cell c_j where c ⊂ c_j ⊆ c' do
7          if u_p × V_{c_j} < ω(S*_c) then
8              S*_{c,c_j} := {(c_j, u_p)} ;
9          else
10             S*_{c,c_j} := S*_c;
11 for current level := 2 to H do
12     for each cell c on current level with c ⊆ c' do
13         χ(c) := child cells of c ;
14         g := Σ_{c_i ∈ χ(c)} Δ(S*_{c_i,c});
15         if g > F_c then
16             S*_c := ⋃_{c_i ∈ χ(c)} S*_{c_i,c};
17         else
18             S*_c := ⋃_{c_i ∈ χ(c)} S*_{c_i} ;
19         for each cell c_j where c ⊂ c_j ⊆ c' do
20             S_{c,c_j} := ⋃_{c_i ∈ χ(c)} S*_{c_i,c_j} ;
21             if ω(S_{c,c_j}) < ω(S*_c) then
22                 S*_{c,c_j} := S_{c,c_j};
23             else
24                 S*_{c,c_j} := S*_c;
25 return S*_{c'}
```

Build Optimal Summary. Algorithm 2 illustrates the construction of S_U^* based on the dynamic programming technique. Line 1 identifies the minimal cell c' which contains the MBB of the object. In Lines 3-10, we create *virtual* instances for all descendant cells of c' on the first level, as well as their corresponding optimal summaries. Lines 11-24 show how to build related optimal summaries of a cells c with $l(c) > 1$. Let $\chi(c)$ denote the child cells of c (Line 13). In Line 14, we first compute the total gains, denoted by g, of the child cells (i.e., $\chi(c)$) regarding c. If the gains can

We have $e_{c_j}.p = 0$ (i.e., e_{c_j} is a dummy entry) if none of the instances is assigned to c_j in S_{c,c_j}.

pay off the *filtering* cost on cell c (i.e., $g > F_c$), we come up with S_c^* by merging its child optimal summaries regarding c (Line 16), otherwise S_c^* consists of the optimal summaries of its child cells (Line 18). Lines 19-24 compute the optimal summaries of c regarding its ancestor cells. Specifically, for each ancestor cell c_j of c with $c_j \subseteq c'$, S_{c,c_j} is the merged result of the optimal summaries of its child cells regarding c_j (Line 20). If there is no gain compared with S_c^* (i.e., $\omega(S_{c,c_j}) \geq \omega(S_c^*)$), S_{c,c_j}^* is set to S_c^* (Line 24). Otherwise, S_{c,c_j} is the optimal summary of c regarding c_j (Line 22).

EXAMPLE 3. *In Fig. 8, we have* $\omega(S_{c_{1,1}}^*) = 0.01 + 0.002 \times 0.2 = 0.0104$, $\omega(S_{c_{1,2}}^*) = 0.0112$, *and* $\omega(S_{c_{1,3}}^*) = \omega(S_{c_{1,4}}^*) = 0.0102$. *Meanwhile, since* $V_{c_2} \times u_1.p = 0.004 < \omega(S_{c_{1,1}}^*)$, *we have* $\omega(S_{c_{1,1},c_2}^*) = 0.004$, $\Delta(S_{c_{1,1},c_2}^*) = \omega(S_{c_{1,1}}^*) - \omega(S_{c_{1,1},c_2}^*) = 0.0064$ *and* $S_{c_{1,1},c_2}^* = \{(c_2, 0.2)\}$. *Similarly, we have* $\omega(S_{c_{1,3},c_2}^*) = \omega(S_{c_{1,4},c_2}^*) = 0.002$, $\Delta(S_{c_{1,3},c_2}^*) = \Delta(S_{c_{1,4},c_2}^*) = 0.0082$ *and* $S_{c_{1,3},c_2}^* = S_{c_{1,4},c_2}^* = \{(c_2, 0.1)\}$. *Since there is no gain by pushing* u_2 *to* c_2 *as* $0.6 \times V_{c_2} > \omega(S_{c_{1,2}}^*)$, *we have* $\omega(S_{c_{1,2},c_2}^*) = \omega(S_{c_{1,2}}^*) = 0.0112$, *and hence* $\Delta(S_{c_{1,2},c_2}^*) = 0$ *and* $S_{c_{1,2},c_2}^* = S_{c_{1,2}}^* = \{(c_{1,2}, 0.6)\}$. *In Algorithm 2,* $g = \sum_{c \in \chi(c_2)} \Delta(S_{c,c_2}^*) = 0.0228$ *and* $g > F_{c_2} = 0.02$. *Therefore, we have* $S_U^* = S_{c_2}^* = \bigcup_{c \in \chi(c_2)} S_{c,c_2}^* = \{(c_{1,2}, 0.6), (c_2, 0.4)\}$ *and* $\omega(S_U^*) = 0.0392$.

Correctness. With similar argument in Theorem 2, regarding all instances assigned to the cells above c', we can move them to c' to reduce the cost since all instances are contained by c'. Therefore, we only need to consider the cells in $\varphi(c')$ during the optimal summary construction.

The following proof is by induction on the levels of U-Quadtree. The correctness of the optimal summaries of the cells on the first level is immediate based on their definitions. Suppose Algorithm 2 is correct regarding cells on the levels lower than k, then for a cell c on level k, we first show the correctness of its optimal summary S_c^*.

Clearly, the cell c is either *empty* or *occupied* in S_c^*. Let S_c^1 denote the merged result of the optimal summaries of its child cells, i.e., $S_c^1 = \bigcup_{c_i \in \chi(c)} S_{c_i}^*$. S_c^1 is a summary of c because, for any instance $x \in c$, there is exactly one child cell $c_i \in \chi(c)$ in which $x \in c_i$ and hence x is assigned to exactly one cell in S_c^1. Moreover, we have $\omega(S_c^1) = \sum_{c_i \in \chi(c)} \omega(S_{c_i}^*)$. Then we show S_c^1 is the optimal summary when c is *empty*. The proof is by contradiction. Suppose there is another summary S_c^2 with $\omega(S_c^2) < \omega(S_c^1)$, where $S_c^2 = \bigcup_{c_i \in \chi(c)} S_{c_i}^2$ and $S_c^2(c) = \emptyset$, then there is a summary $S_{c_i}^2$ where $c_i \in \chi(c)$ such that $\omega(S_{c_i}^2) < \omega(S_{c_i}^*)$ because $\omega(S_c^2) = \sum_{c_i \in \chi(c)} \omega(S_{c_i}^2)$. This is against our assumption since the level of c_i is lower than k and $S_{c_i}^*$ is the optimal summary of c_i. Therefore, S_c^1 is the optimal summary when c is *empty*.

If c is occupied in S_c^*, some instances must be pushed to c in S_c^*. Let S_c^3 denote the merged result of the optimal summaries of the child cells regarding c, i.e., $S_c^3 = \bigcup_{c_i \in \chi(c)} S_{c_i,c}^*$. S_c^3 is a summary of c because any instance contained by c is contained by exactly one of its child cell, say c_i, and any instance $x \in c_i$ must be assigned to exactly one cell in $S_{c_i,c}^*$. With similar rational to the above case where c is empty, we can prove that S_c^3 has the lowest cost among the summaries of c in which c is *occupied*. $g > F_c$ in Line 15 of Algorithm 2 implies that $\omega(S_c^3) < \omega(S_c^1)$; that is, it is worthwhile to occupy the cell c by pushing some instances to c because this can pay off the *filtering* cost of c. Consequently, S_c^3 is the optimal summary when c is *occupied* and hence the correctness of the optimal summaries of the cells on level k holds.

The correctness of the optimal summary of c regarding its ancestor cells can be obtained with similar rationale, we omit the details of the proof due to space limits.

Cost Analysis. In Algorithm 2, it takes $O(h)$ time to locate c' where h is the height of the U-Quadtree. It takes $O(hm)$ time to create the *virtual* instances in a top-down fashion where m is the number of the instances in U. As to each cell c, it costs $O(h)$ time to compute related optimal summaries (S_c^* and S_{c,c_j}^* where $c \subset c_j$). Note that it takes $O(1)$ time to merge child summaries and calculate related costs at Line 16, 18 and 20 because the child summaries share at most one cell (e.g., cell c in Line 16). Therefore, the **time complexity** of Algorithm 2 is $O(h \times m + h \times \frac{4^h - 1}{3})$ in the worst case since the number of cells in UQ_T is bounded by $\frac{4^h-1}{3}$. For a cell c on level s, there are at most 4^{s-1} entries for each optimal summary since c has at most 4^{s-1} descendant cells on the first level. There are at most 4^{h-s} cells on level s and we only need to keep at most $h - s + 1$ related optimal summaries for each cell. Therefore, the space usage for each level is $O(4^{h-1} \times h)$ in the worst case. Since in Algorithm 2 we do not need to keep the summaries of the child cells once the optimal summaries computation of a cell is finished, the **space complexity** of Algorithm 2 is $O(4^{h-1} \times h)$.

Addressing three factors in Section 1. The final part of this subsection shows that three factors mentioned in Section 1 are carefully addressed in Algorithm 2. The resources allocated to an object are measured by the number of entries in its optimal summary. For an object with large uncertain region size, the number of *virtual* instances is large and hence more resources (i.e., the number of entries in S_U^*) may be allocated. The instances in the sparse part of the PDF tends to be pushed to the cells on higher levels in U-Quadtree because the gain of "pushing up" is more significant than that of the instances from the dense part. Therefore, more resources are allocated to the dense part of an object. With same rationale, the instances of the objects with high *verification* cost tend to stay on the lower level cells, and hence lead to a large summary size.

4.2 Optimal U-Quadtree Construction

According to the cost model in Section 3, the U-Quadtree is **optimal** if all summaries of the objects are optimal because $E(C) = \sum_{U \in \mathcal{U}} \omega(S_U)$. Therefore, we can easily come up with the optimal U-Quadtree construction algorithm by updating UQ_T and UQ_E based on the entries in S_U^*.

Algorithm 3 illustrates how to insert an object into the U-Quadtree. We first create the optimal summary of U based on Algorithm 2 (Line 1). Then each entry of S_U^* is inserted to the entry index of U-Quadtree (Line 2-8). The procedure is the same as the insertion of B^+ tree except that we need to maintain page links of the cells (Lines 6 and 8).

Cost Analysis. Let n and n_e denote the number of the objects in $\mathcal{U}$ and the average number of entries in the optimal summaries, respectively. According to the analysis of Algo-

Recall that we do not include the *filtering* cost on c (F_c) when computing the cost of the optimal summary of each child cell regarding c (i.e., $\omega(S_{c_i,c}^*)$ for each $c_i \in \chi(c)$).

Entries with the same cell *id* will be merged by accumulating their probability values.

Suppose $x \in c$, it takes $O(1)$ time to identify the child cell of c which contains x.

Algorithm 3: ***Insert***(UQ, U)

Input : UQ : the U-Quadtree,
U : an object in $\mathcal{U}$
Output: UQ : the updated U-Quadtree

```
S*_U ← BuildOptSummary(UQ, U) ;
for each entry e in S*_U do
    find corresponding entry page P based on e.c;
    if P is full then
        Split page P;
        Reset the page links of the cells whose entries
        are moved to the new page ;
    Insert e into corresponding entry page ;
    Set the page link of e.c ;
return UQ
```

rithm 2, the **space complexity** of the optimal U-Quadtree construction algorithm is $O(n \times n_e + 4^{h-1} \times h)$. The dominant cost of Algorithm 3 is the summary construction and B^+ Tree (UQ_E) maintenance. According to [10], the amortized cost of the insertion for B^+ Tree is $O(\log_f n \times n_e)$. Consequently, the **time complexity** of our index construction algorithm is $O(n \times \frac{4^h-1}{3} \times h + n \times n_e \times \log_f(n \times n_e))$.

Deletion. The deletion of an object U from U-Quadtree is straightforward. Firstly we find the cells which are *contained* or *overlapped* by U_{mbb}. Then the entries associated with the cells are removed from UQ_E. Same as the insertion, we also need to maintain the page links of the cells.

4.3 Unknown Query Distribution

When the distribution of the query load is unknown, it is intuitive to construct the U-Quadtree with query uniform assumption. This is arguably the best choice since that, without any priori knowledge, we should assume that all cells at the same level of the U-Quadtree have the same chance to be *contained* or *overlapped* by the search region. In the experiments, we construct the U-Quadtree based on the query uniform assumption and it is shown that our technique can significantly outperform the competitors against various query distributions. Particularly, we assume each query region is a square whose center is uniformly distributed in the space $[0, 1]^d$ and the length l is randomly chosen from $[0, 1]$. With similar argument in [22], we have $P_c(c) + P_o(c) = min((l + w)^d, 1.0)$ where w is the length of cell c. Similarly, $P_o(c) = (l + w)^d - (l - w)^d$ if the $w < l$; Otherwise, $P_o(c) = min((l + w)^d, 1.0)$.

5. PERFORMANCE EVALUATION

We present results of a comprehensive performance study to evaluate the efficiency and scalability of the proposed techniques in the paper. Following algorithms are evaluated.

U**-Tree** The U-Tree technique presented in [21]. Note that the *R-tree* based technique is a special case of U-Tree where the catalog size equals one. As shown in [21], the R-tree technique is significantly outperformed by U-Tree. So we exclude R-tree technique in our empirical study.

UI**-Tree** The UI-Tree technique proposed in [25].

UP The pivot indexing technique proposed in [3].

UD**-Tree** The U-Quadtree technique. By default, the query uniform assumption in Section 4.3 is used to construct U-Quadtree for comparison fairness.

Datasets. Three real spatial datasets, CA, LB and US, are employed to represent the centers of the uncertain regions. They contain $62K$, $53K$ and $200K$ 2-dimensional points representing locations in Los Angeles, Long Beach and the United States respectively which are available at http://www.census.gov/geo/www/tiger and CA is the default dataset. We also generate synthetic dataset where dimensionality various from 2 to 4, named $2D$, $3D$ and $4D$ respectively. There are $200K$ points in each synthetic dataset and they are uniformly distributed. All dimensions are normalized to domain $[0, 10000]$. In our experiment, the uncertain region of the uncertain object is a circle or sphere with expected radius r_u varying from 100 to 300 with default value 200 for 2-dimensional data. r_u is set to 600 and 1200 for 3 and 4 dimensional data respectively. For a given r_u, the radius of the objects are randomly chosen between 0 and $2 \times r_u$. Suppose the PDF of an object is described by m instances which follow two popular distributions *Normal* and *Uniform*, where the expected m varies from 200 to 1000 with default value 400. Given m, the number of instances for each object uniformly distributes between 0 and $2m$. The *Normal* serves as default distribution with standard deviation $\frac{r_u}{2}$. There are totally around $24.8M$ instances for the default dataset (CA) and the maximal number of instances is $80M$ in US dataset.

Workload. A workload for the range query consists of 200 queries in our experiments. Same as [21, 25], the region of a range query r_q is a circle or sphere with radius γ (query distance). By default, γ is uniformly distributed between 500 and 1500 in 2-dimensional space, and γ is set to 1900 and 3500 in 3 and 4 dimensional space respectively. We use $Q(CA)$ to denote the queries whose centers are generated from the centers of the CA datasets. Similarly, we have $Q(LB)$ and $Q(US)$. By default, the query distribution follows the underlying target data distribution. For instance, $Q(CA)$ is the default query distribution for CA dataset. We also create synthetic query distributions whose centers follow the *uniform*, *anti-correlated* and *correlated* distributions based on the generator from [7], denoted by $Q(E)$, $Q(A)$ and $Q(C)$ respectively. Meanwhile, we randomly choose the probabilistic threshold $\theta \in (0, 1]$ for each query. The average response times are recorded to evaluate the overall performance of the range searching algorithms. We also record the average number of disk accesses (index I/O and data I/O), candidate objects and false positives.

All algorithms proposed in this paper are implemented in standard C++ with STL library support and compiled with GNU GCC. Experiments are run on a PC with Intel Xeon 2.40GHz dual CPU and 4G memory running Debian Linux. The disk page size is fixed to 4096 bytes and the capacity of the entry page (f) is set to 512.

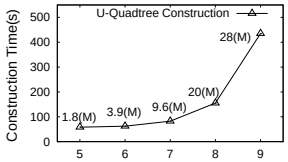


Figure 9: Diff. h

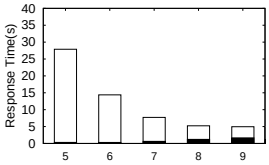
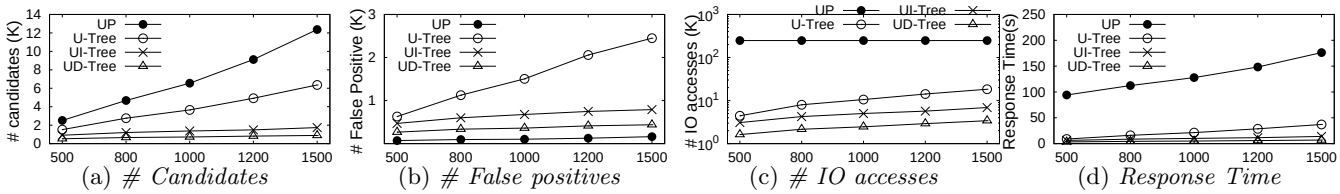


Figure 10: Diff. h

In order to achieve the best overall performance, the catalog size of U-Tree [21] is set to 8, 10 and 12 for 2, 3 and



(a) # *Candidates* (b) # *False positives* (c) # *IO accesses* (d) *Response Time*

Figure 11: Diff. γ

4-dimensional dataset respectively. Similarly, the merge factor m of UI-Tree [25] is set to 8, 4 and 3 respectively. As suggested in [3], we set the histogram size l to 100 and there are $10 \times d$ pivot points for d-dimensional dataset. Table 3 lists parameters which may have an impact on our performance study. In our experiments, all parameters use default values unless otherwise specified.

Notation	Definition (Default Values)
h	height of U-Quadtree (8)
γ	query distance (1000)
r_u	radius of uncertain object region(200)
n	number of uncertain objects ($62K$)
m	number of instances per object (400)
θ	probabilistic threshold ($\in (0, 1]$)

Table 3: System Parameters

5.1 Construction of U-Quadtree

For comparison fairness, in the experiments we construct the U-Quadtree based on the uniform assumption of the query load as discussed in Section 4.3. We set $t_{io} = 1$ and set t_U to the number of pages occupied by the object U since we only need to know the ratio of $\frac{t_U}{t_{io}}$ for the optimal U-Quadtree construction, instead of exact t_{io} and t_U values.

Fig. 9 shows the construction times on CA dataset where h varies from 5 to 9. As expected, the construction time and index size grow with h. The index size is $20M$ when $h = 8$, which is 1.3% of the dataset. The index size of U-Tree, UI-Tree and UP-Index are $13M$, $34M$ and $969M$ respectively. The construction time of UD-Tree is 155 second when $h = 8$, and it takes 120, 332, 1152 seconds for U-Tree, UI-Tree and UP respectively.

Fig. 10 reports the range query response time where the solid bar and empty bar represent the *filtering* time and *verification* time respectively. It is interesting to see that the performance of the algorithm becomes stable once h is sufficiently large ($h \geq 8$). This is because the instances of the objects can be "pushed up" to a proper level by the index construction algorithm. In the following experiments, we set h to 8, 6 and 4 for 2, 3 and 4 dimensional data respectively.

5.2 Evaluate Range Query

In this subsection, we evaluate the performance of the algorithms for range searching.

Impact of query distance (γ). Fig. 11 reports the average query response time, the average number of candidates, false positives and disk accesses of four algorithms (U-Tree, UI-Tree, UP and UD-Tree) against query distance γ. Recall that, by default the query distribution is obtained from the centers of the underlying target objects. Therefore $Q(CA)$ is the query distribution. Results show that although UP Algorithm has the smallest number of false positives, the candidate size is large because UP-Index Algorithm cannot *validate* any object. Moreover, the number of I/O accesses is also large since the whole index is scanned for the object pruning. Therefore, the overall performance of UP-Index Algorithm is not competitive compared with other algorithms and hence we exclude UP-Index Algorithm in the following experiments. Fig. 11 shows that UD-Tree Algorithm always outperforms UI-Tree and U-Tree Algorithms.

Impact of query distribution. Fig. 12 reports the response time of the algorithms on CA dataset where various query distributions are employed. Particularly, the UD-Tree(OPT) Algorithm constructs the U-Quadtree based on each corresponding query distribution (i.e., query distribution is known). Recall that the query uniform assumption in Section 4.3 is employed for UD-Tree Algorithm. It is shown that UD-Tree(OPT) Algorithm can make some performance improvement compared with UD-Tree Algorithm. Results shows that UD-Tree Algorithm significantly outperforms U-Tree and UI-Tree Algorithms under all query distributions, which implies that constructing U-Quadtree based on query uniform assumption is a good choice in practise when the query distribution is unknown.

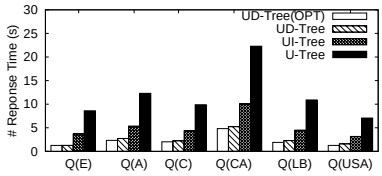
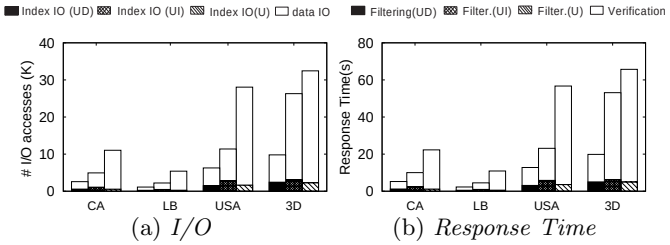


Figure 12: Diff. Query Distributions

Impact of datasets. Fig. 13 reports the number of I/O accesses and the query response time of three algorithms against CA, LB, US and $3D$ datasets. Results show that UD-Tree Algorithms ranks first under all datasets.

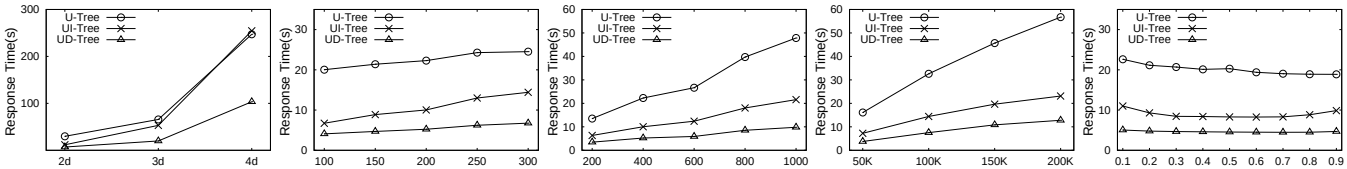


(a) *I/O* (b) *Response Time*

Figure 13: Performance vs diff. datasets

Impact of other parameters. We also study the impact of other parameters which may potentially affect the performance of the algorithms. Specifically, Fig. 14(a), Fig. 14(b), Fig. 14(c) and Fig. 14(d) investigate the scalability of the algorithms against d (dimensionality), r_u (uncertain region size), m (instance size) and n (object size) respectively. As expected, Fig. 14(a) shows that the performances of three algorithms degrade rapidly against dimensionality. Nevertheless, UD-Tree Algorithm is more scalable than UI-Tree and

Note that UP-Index focus on supporting the range search on general metric space, which cannot be achieved by U-Tree, UI-Tree and U-Quadtree techniques.



(a) *Diff.* d (b) *Diff.* r_u (c) *Diff.* m (d) *Diff.* n (e) *Diff.* θ

Figure 14: Impact of various parameters

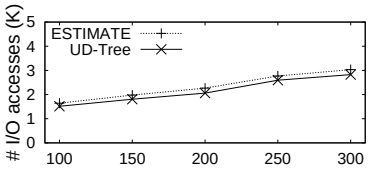


Figure 15: Accuracy Evaluation

U-Tree Algorithms, and UD-Tree Algorithm always outperforms other algorithms under all settings. Note that in Fig. 14(d) we randomly choose $50K$, $100K$ and $150K$ objects from *US* dataset. Finally, Fig. 14(e) shows that the performances of the algorithms are not sensitive to the probabilistic threshold θ.

5.3 Accuracy of the Cost Model

We evaluate the accuracy of the cost model by estimating the number of I/Os in the experiments. Particularly, in the cost model (Equation 6) we set t_{io} and t_U to 1 and the number of pages the object U occupied respectively. Then $E(C)$ corresponds to the expected number of I/O accessed. Fig. 15 verifies the accuracy of the cost model on *CA* dataset where r_u varies from 100 to 300, and search regions are squares with length 4000. The total number of I/O accesses are predicated, and results show the proposed analytical model is quite accurate, the relative error usually being around 8%.

6. CONCLUSION AND FUTURE WORK

To address the uncertainty in various applications, we have developed an effective indexing technique to support range search on multi-dimensional uncertain objects. Based on some insights of the range search we build a cost model which carefully considers various factors which may impact the performance of the range query. An effective and efficient index construction algorithm is proposed based on the cost model. Our experiments convincingly demonstrate the effectiveness and efficiency of our indexing techniques. As a future work, we will investigate how to extend our indexing technique to tackle the correlation among the multi-dimensional uncertain objects.

Acknowledgement. Ying Zhang is supported by ARC DP110104880 and UNSW ECR grant PS27476. Wenjie Zhang is supported by ARC DP120104168 and ARC DE12010 2144. Xuemin Lin is supported by ARC DP0987557, DP110102937 and DP120104168.

7. REFERENCES

[1] P. K. Agarwal, S.-W. Cheng, Y. Tao, and K. Yi. Indexing uncertain data. In *PODS*, 2009.

[2] C. Aggarwal and P. Yu. On high dimensional indexing of uncertain data. In *ICDE 2008*.

[3] F. Angiulli and F. Fassetti. Indexing uncertain data in general metric space. *TKDE*, to appear.

[4] G. Beskales, M. A. Soliman, and I. F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. *PVLDB*, 1(1), 2008.

[5] C. Böhm, M. Gruber, P. Kunath, A. Pryakhin, and M. Schubert. Prover: Probabilistic video retrieval using the Gauss-tree. In *ICDE 2007*.

[6] C. Böhm, A. Pryakhin, and M. Schubert. Probabilistic ranking queries on Gaussians. In *SSDBM 2006*.

[7] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[8] J. Chen and R. Cheng. Efficient evaluation of imprecise location-dependent queries. In *ICDE*, 2007.

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

[10] D. Comer. The ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137, 1979.

[11] C. Faloutsos. Gray codes for partial match and range queries. *IEEE Trans. Software Eng.*, 1988.

[12] R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 1974.

[13] E. Frentzos, K. Gratsias, and Y. Theodoridis. On the effect of location uncertainty in spatial querying. *TKDE*, 21(3):366 –383, 2009.

[14] G. R. Hjaltason and H. Samet. Speeding up construction of pmr quadtree-based spatial indexes. *VLDB J.*, 2002.

[15] H. Kimura, S. Madden, and S. B. Zdonik. Upi: A primary index for uncertain databases. *PVLDB*, 2010.

[16] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Probabilistic similarity join on uncertain data. In *DASFAA*, 2006.

[17] X. Lian and L. Chen. A generic framework for handling uncertain data with local correlations. *PVLDB*, 2010.

[18] Y. Ma, D. V. Kalashnikov, and S. Mehrotra. Toward managing uncertain spatial information for situational awareness applications. *TKDE*, 20(10), 2008.

[19] J. H. Schiller and A. Voisard. *Location-based Services*. Morgan Kaufmann, 2004.

[20] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. E. Hambrusch. Indexing uncertain categorical data. In *ICDE*, 2007.

[21] Y. Tao, X. Xiao, and R. Cheng. Range search on multidimensional uncertain data. *ACM Trans. Database Syst.*, 32(3), 2007.

[22] Y. Theodoridis and T. K. Sellis. A model for the prediction of r-tree performance. In *PODS*, 1996.

[23] T. T. L. Tran, C. Sutton, R. Cocci, Y. Nie, Y. Diao, and P. J. Shenoy. Probabilistic inference over rfid streams in mobile environments. In *ICDE*, 2009.

[24] M. Zhang, S. Chen, C. S. Jensen, B. C. Ooi, and Z. Zhang. Effectively indexing uncertain moving objects for predictive queries. *PVLDB*, 2009.

[25] Y. Zhang, X. Lin, W. Zhang, J. Wang, and Q. Lin. Effectively indexing the uncertain space. *TKDE*, 2010.