Effectively Indexing the Uncertain Space

Ying Zhang, *Member*, *IEEE*, Xuemin Lin, Wenjie Zhang, Jianmin Wang, and Qianlu Lin

Abstract—With the rapid development of various optical, infrared, and radar sensors and GPS techniques, there are a huge amount of multidimensional uncertain data collected and accumulated everyday. Recently, considerable research efforts have been made in the field of indexing, analyzing, and mining uncertain data. As shown in a recent book [2] on uncertain data, in order to efficiently manage and mine uncertain data, effective indexing techniques are highly desirable. Based on the observation that the existing index structures for multidimensional data are sensitive to the size or shape of uncertain regions of uncertain objects and the queries, in this paper, we introduce a novel R-Tree-based inverted index structure, named UI-Tree, to efficiently support various queries including range queries, similarity joins, and their size estimation, as well as top-k range query, over multidimensional uncertain objects against continuous or discrete cases. Comprehensive experiments are conducted on both real data and synthetic data to demonstrate the efficiency of our techniques.

Index Terms—Uncertain, index, range query, partition.

1 Introduction

Managing and mining uncertain data have various applications covering data cleaning, sensor data analysis, moving objects tracking, information retrieval, crime fighting, 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. With the wide application of GPS equipment and various optical, infrared, and radar sensors, a large amount of uncertain data are collected and accumulated. Efficiently managing and analyzing large volumes of uncertain data becomes a main challenge in the database research community. A number of issues have been addressed including modeling [33], managing, and mining uncertainty. Various types of probabilistic queries and uncertain data mining approaches have been studied, such as query evaluation [10], [15], indexing [36], top-k queries [25], skyline queries [31], similarity joins [27], nearest neighbor query [28], and data clustering [3].

Range search over uncertain data is important in query processing and data mining, which have many applications. As an example, a server monitors a set of taxis equipped with GPS and location information of each taxi is sent back to the server every 5 minutes. Based on this periodically updated location information and other factors like velocity constraint, at each time stamp, the location of a taxi is within a circle until next update arrives. The server may issue queries like "find taxis which are currently within 10 kilometers from the city tower." Since the location is not exact, a taxi may satisfy this query partially, as shown in Fig. 1. The gray circles represent uncertain region of taxis while the transparent circle is the query region. While A definitely satisfies the query, B and C *probably* satisfy it, which means that they are within the query region with a *probability*. This probability can be intuitively computed based on the intersection between one uncertain region and the query region, also the specific probability density function (PDF) information inside each taxi's uncertain region. Results with low probability values are often of no interest to users and a probability threshold is sometimes given beforehand to return results with probability no less than this threshold only.

Continuing with the example of monitoring taxis in Fig. 1, in some cases, specific identification of taxis is not necessary and only aggregate information is required, such as "how many taxis are currently inside city?" Taxis with uncertain region partially inside city will also be considered probabilistically; Ranking based on probability is another way to handle possible results besides the threshold-based fashion. To schedule the taxis, the server may retrieve 10 available taxis satisfying "within distance at most 5 km from Four Seasons Hotel" with the highest probability. Such a query is called a probabilistic top-k range query.

Range search is also a key component in the *filtering* phase of many queries in mining uncertain data such as spatial similarity join and k nearest neighbor query. Particularly, the spatial similarity join is essential to identify pairwise similar objects represented by uncertain multidimensional data. And k nearest neighbor query plays an important role in the study of spatial clustering and machine learning.

There are two types of techniques for indexing uncertain data with arbitrary PDF. The first type is R-Tree-based index [11], [26], [27], [29], [34]. More specifically, the uncertain region of multidimensional uncertain objects is grouped by R-Tree, where each data unit is the minimum bounding rectangle (MBR) of a PDF. The drawback of this approach is that the uncertain region of an object is considered as an atomic unit, which leads to a poor performance for the probabilistic threshold-based queries when individual MBRs are large. The second type of index is based on *probabilistically constrained regions* (PCRs) [9], [36]. The uncertain region of an object is partitioned with respect to a set of probability values.

- *Y. Zhang, X. Lin, W. Zhang, and Q. Lin are with the University of New South Wales, Sydney, NSW 2052, Australia. E-mail: {yingz, lxue, zhangw, qlin}@cse.unsw.edu.au.*
- *J. Wang is with the School of Software, Tsinghua University, 100084 R.R. China. E-mail: jimwang@tsinghua.edu.cn.*

Manuscript received 28 Mar. 2009; revised 9 Aug. 2009; accepted 28 Sept. 2009; published online 30 Apr. 2010.
Recommended for acceptance by R. Cheng, M. Chau, M. Garofalakis, and J.X. Yu.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-2009-03-0213.
Digital Object Identifier no. 10.1109/TKDE.2010.77.

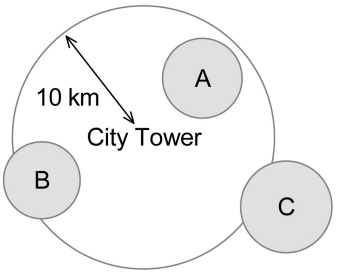


Fig. 1. Taxis within 10 km from the City Tower.

Partitioning results are then organized into an R-Tree style structure named U-Tree. U-Tree significantly outperforms *uncertain region*-based index by utilizing probability thresholds in range query processing. While U-Tree supports the range queries with rectangular regions aligning the dimensional axes of data space well, it may not always provide a good support to range queries with nonrectangular regions or rectangular regions not aligning to axes. Details and analysis of existing indexes will be introduced in Section 2.

Motivated by these facts, in this paper, we study the problem of indexing uncertain data to support queries that require efficient range query processing. Contributions can be summarized as follows:

- A space-efficient index structure for organizing multidimensional uncertain objects, UI-Tree, is proposed. UI-Tree can support arbitrary PDF of uncertain objects.
- We develop efficient solutions for various types of queries based on UI-Tree, including range query, size estimation of range query, probabilistic top-k range query, and similarity join.
- We provide rigorous analysis to estimate the filtering capacity of UI-Tree.
- Extensive experiments over real and synthetic data sets are conducted to demonstrate the efficiency and scalability of UI-Tree compared with other state-of-the-art techniques.

The rest of the paper is organized as follows: We formally define the problem and provide background information in Section 2. Section 3 presents the UI-Tree index structure. Section 4 applies UI-Tree to support different types of queries. Results of comprehensive performance studies are discussed in Section 5. Finally, Section 6 concludes the paper.

2 Preliminary

In Section 2.1, we first formally define the model of multidimensional uncertain objects and queries studied in the paper. These are followed by the problem statement of this paper. Existing indexing approaches are reviewed in Section 2.2. Table 1 summaries the mathematical notations used throughout the paper.

2.1 Problem Definition

Points referred in this paper, by default, are in d-dimensional numerical space $D = \{D_1, \ldots, D_d\}$, where D_i denotes the ith dimension. A multidimensional uncertain object U in our paper can be regarded as a point whose location might appear at some locations with certain probabilities. Each possible appearance of the object is regarded as an instance of the uncertain object. Whenever there is no ambiguity, for instance, u, we use u and $u.p$ to represent the location(point) of the instance and its appearance probability, respectively. For presentation simplicity, we use "uncertain object" to represent "multidimensional uncertain object."

TABLE 1
The Summary of Notations

Notation	Definition
U,V	uncertain objects
$\mathcal{U}$, $\mathcal{V}$	set of uncertain objects
n	the number of uncertain objects in data set
Q (Q_r)	range query (query region)
θ	probabilistic threshold
$P_{app}(Q,U)$	the appearance probability of U w.r.t Q
l	number of partitions for each uncertain object
$w(w_p, w_{list})$	a *word* (probability value, posting list)
$A(w)(A(Q))$	the area of $w_{mbr}(Q_r)$
$WA(w)$	the weighted area of a word
$f_i(f_l)$	fan-out of non-leaf(leaf) node in UI-Tree
$t_{w,U}$	tuple from w_{list} with oid U
$P(t_{w,U})$	probability of tuple $t_{w,U}$
C	Candidate set
m	*merge* factor

An uncertain object can be described either *continuously* or *discretely*. In the *continuous* case, an uncertain object U is described by its PDF $U.pdf$ and uncertain region U_r. The appearance probability of an instance $x \in U_r$ is $U.pdf(x)$ and $\int_{x \in U_r} U.pdf(x)dx = 1$. In the *discrete* case, an uncertain object U consists of a set of instances $u_1, \ldots, u_m$, where u_i appears with probability $u_i.p$ and $\sum_{u \in U} u.p = 1$. For the presentation simplicity, we only discuss the *continuous* cases in the following part of the paper as discrete cases can be easily mapped to *continuous* cases.

Before defining range queries over uncertain data, we first define appearance probability of an uncertain object with respect to the query region. Because of the uncertainty of the location of an object, it may be no longer meaningful to simply declare that it appears or does not appear in the query region. For a given query Q with query region Q_r and uncertain object U, we use $P_{app}(U,Q)$ to represent the probability that U falls in Q_r. $P_{app}(U,Q)$ is defined as follows:

$$P_{app}(U,Q) = \int_{x \in U_r \cap Q_r} U.pdf(x)dx.$$

Usually, query results with low probabilities are of no interest to users. Many queries studied in the literature are accompanied with a user-defined probabilistic threshold θ, which reflects the requirements or confidence level of the user. Following is the definition of probabilistic threshold range query [12], [35]. For presentation simplicity, we use "range query" to denote "probabilistic threshold range query" whenever there is no ambiguity:

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

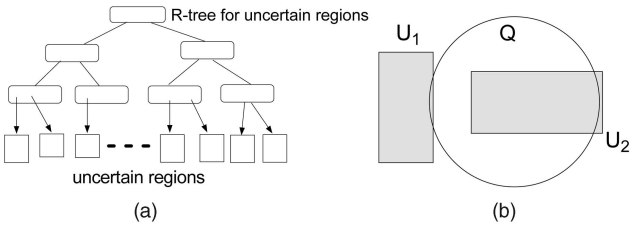


Fig. 2. Uncertain region-based index.

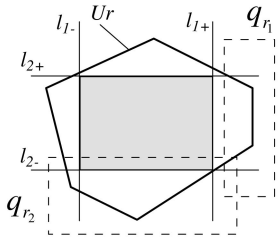


Fig. 3. A 2D PCR (θ).

In some applications, it suffices to get an approximate number of uncertain objects instead of retrieving the uncertain objects qualifying the range query. We call this size estimation of range query.

Definition 2 (Size Estimation of Range Query). *For a given set of uncertain objects $\mathcal{U}$ and a range query Q, estimate the number of uncertain objects $U \in \mathcal{U}$ with $P_{app}(U, Q) \geq \theta$, where θ is the user-specified probabilistic threshold and $0 < \theta \leq 1$.*

To handle results with low appearance probability $P_{app}(U, Q)$, ranking the objects based on $P_{app}(U, Q)$ and returning top-k results only is another method besides probabilistic threshold-based approach. Following is the problem definition:

Definition 3 (Top-k Range Query). *For a given set of uncertain objects $\mathcal{U}$ and a range query Q, a top-k range query retrieves k uncertain objects $U \in \mathcal{U}$ with highest $P_{app}(U, Q)$.*

Efficient processing of joins often relies on fast execution of range query in the filtering phase. The problem of distance-based spatial similarity join over uncertain data is introduced in [27]. Following is a formal definition of this problem in a probabilistic threshold fashion. It is referred as "similarity join" when there is no ambiguity:

Definition 4 (Probabilistic Threshold Similarity Join). *For two given sets of uncertain objects $\mathcal{U}$ and $\mathcal{V}$, retrieve all pairs of (U, V), where $U \in \mathcal{U}$ and $V \in \mathcal{V}$, such that*

$$\int_{x \in U_r} \int_{y \in V_r \wedge |x-y| \leq \gamma} U.pdf(x) \times V.pdf(y) dy dx \geq \theta.$$

γ and θ are predefined distance and probabilistic threshold, respectively.

Problem statement. In this paper, we aim to build an efficient index to support various queries that rely on efficient processing of range query. The index supports uncertain objects with arbitrary PDFs and is not sensitive to the size and shape of the query regions.

2.2 Related Work

In this section, we first briefly describe and analyze two types of indexing structures supporting uncertain objects with arbitrary PDFs, *R-Tree-based index* and *PCR-based index*. Then we introduce the *inverted index* technique and its application in indexing uncertain objects. In the end is a brief introduction of other existing techniques.

***R*-Tree-based index**. R-Tree family [21] is tree data structures which are similar to B-Tree, but are used for spatial access methods in which a set of points or rectangles is recursively grouped. Each intermediate entry of R-Tree is represented as an MBR, which is the *minimal bounding rectangle* of the entry that tightly bounds all the data in the subtree. R-Tree can efficiently support the range query because it can prune or validate a group of objects at intermediate entries. Moreover, the construction of R-Tree aims to maximize the chance of pruning/validating R-Tree entries for the range query as well.

A simple way to index the uncertain objects is to organize their uncertain regions with existing indexing approaches like R-Tree [27], [26], [34], [11], [29]. Fig. 2a illustrates the basic idea of the uncertain region-based indexing, where the uncertain regions of the uncertain objects are indexed by R-Tree. It is simple and performs well if the uncertain regions of objects are very small regarding the query region size. As the uncertain region is considered as an atomic unit in the index, without further exploring the detailed information, it cannot tell whether or not an uncertain object satisfies the query when uncertain region overlaps range query. Such an index inherently limits the filtering capacity of the index and is not suitable to the probabilistic threshold-related queries. As shown in Fig. 2b, for a given query Q and probabilistic threshold $\theta = 0.5$, we cannot prune U_1 although intuitively $P_{app}(U_1, Q)$ should be small. Similarly, U_2 cannot be validated either. Consequently, the performance of the index is poor when the size of the uncertain region is not small.

PCR-based index. PCRs-based indexes make use of the detailed information about PDF of uncertain objects to enhance the filtering capacity. It is introduced by Tao et al. [35], [36] to support the range query on uncertain objects in a multidimensional space, where the PDF of the uncertain object might be arbitrary functions. PCR is a general version of *x-bounds*, which aims to index one-dimensional uncertain data [12].

In [35], [36], an uncertain object U is modeled by its PDF $U.pdf(x)$ and uncertain region U_r. For a given probabilistic threshold θ, corresponding $U.pcr(\theta)$ can be employed for pruning and validating purpose. $U.pcr(\theta)$ is constructed as follows: As shown in Fig. 3, 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 shadowed region in Fig. 3 forms $U.pcr(\theta)$. A series of theorems is proposed to take advantage of $U.pcr(\theta)$ to prune or validate U regarding θ. As shown in Fig. 3, suppose both range queries q_1 and q_2 have the same probability threshold

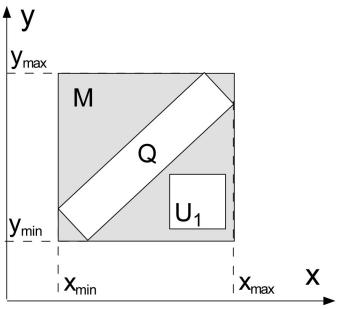


Fig. 4. Irregular query.

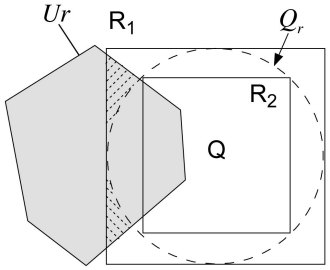


Fig. 5. Circle query.

$\theta \leq 0.5$ and with query regions q_{r_1} and q_{r_2}, respectively. U can be pruned regarding q_1 because q_{r_1} does not intersect with $U.pcr(\theta)$. On the other hand, U can be validated with respect to q_2 since q_{r_2} completely contains U_r below l_{2-}. As it is infeasible to keep all $U.pcr(\theta)$ for any $\theta \in (0, 1]$, a finite number of PCRs are precomputed to facilitate the range query process in [35], [36]. 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 an uncertain object. The main difference is the splitting process in which U-Tree focuses on optimizing the filtering capacity of the PCRs in the intermediate node.

In order to prune or validate an uncertain object, U-Tree needs to project the query region to each dimension as shown in Fig. 4. This loses the spatial "clustering" information, which, in turn, might severely impair the filtering capacity of the PCR technique. As shown in Fig. 4, if the query region Q_r is a rectangle which does not align the x and y axes, then there is no difference between Q_r and M (shadowed rectangle) regarding the pruning ability to uncertain object U_1. It implies that we cannot prune U_1 for query Q regardless of the probabilistic threshold value, even though they do not intersect each other at all. Query Q in Fig. 4 is not uncommon in real applications. For instance, it could be a buffer query [32] based on a segment of roads or rivers, which is a popular query in many Geographic Information System (GIS) applications [32]. Another case is illustrated in Fig. 5, where the range query is a circle. As suggested in [36], two rectangles R_1 and R_2 are utilized for pruning and validation, respectively. This inherently weakens the filtering capacity of U-Tree. As in Fig. 5, U-Tree loses its pruning capacity in the striped areas. As we know, the range query with a circle region is very popular in distance-based queries. Moreover, it is essential for the spatial similarity joins.

Inverted index. In information technology, an inverted index maps from content, such as a word, to its locations in a database file or a document to support full text search. Each word is allocated with a set of posting entries *(docID, offset/frequency)*, which is sorted by the offset or frequency. Inverted index techniques are employed in [1], [30], [34] for indexing uncertain objects in specific applications with assumption or constraints on objects' PDFs or types. The R-Tree and inverted index techniques are employed in the problem of keyword searching on spatial database [23], [19] as well in which the keyword occurrence and the document location are considered. The problem they studied is inherently different from ours.

There are also some studies on indexing uncertain objects, which focus on special cases of objects' PDF or particular data types. For instance, in [6], [7], Böhm et al. study range queries with the constraint that PDF of uncertain objects is *Gaussian* distribution. Managing uncertain trajectories [16], existentially uncertain data [14], uncertain categorical data [34], and vague spatial objects [38] have been separately studied. Aggarwal and Yu [1] study the problem of indexing high-dimensional uncertain data with the assumption that the PDF of the uncertain object on each dimension is independent of others. An index structure called *UniGrid* is proposed to efficiently support the similarity and range query on a selected subset of dimensions. In [30], Ma et al. propose solutions for efficient retrieval of uncertain spatial point data, where the location information is derived from the free text by *spatial expressions*. With an assumption that the space is partitioned by a *virtual grid* with limited number of cells and a region (region of an uncertain object and region of the query) either occupies a whole cell or does not intersect with it at all, a grid index named *U-grid* is built for efficient spatial query processing.

Motivated by the above analysis of existing indexing techniques, we aim to develop a partition-based index structure such that the spatial "clustering" information can be kept and the filtering capacity is less sensitive to the shape of query region. Moreover, the structure should be space efficient and support arbitrary PDF.

3 UI-Tree Index

Based on the analysis of existing index structures for multidimensional uncertain objects, we develop an R-Tree-based inverted index technique, which is based on the partitions of uncertain objects. Section 3.1 introduces the motivation of our index structure and some important index building criteria. Then we describe the details of the index structure and its maintenance algorithms in Sections 3.2 and 3.3, respectively.

3.1 Index Building Criteria

As discussed in Section 2, since R-Tree-based techniques do not capture any details of the PDF of uncertain objects, the performance is poor when the size of the uncertain region is not very small. Although the PCR-based technique makes use of the PDF information by precomputing the *probabilistically constrained regions*, the spatial "clustering" information of the instances is lost because the computation is based

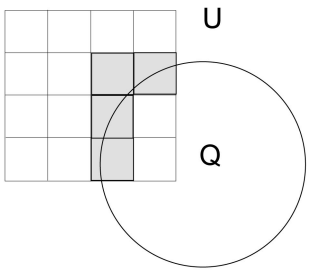


Fig. 6. Motivation.

on projection on each dimension. So, it is sensitive to the shape of the queries. Based on these observations, instead of building index structure against the uncertain region or PCR, we construct the index based on the partitions of the uncertain objects such that the spatial "clustering" information of instances of an uncertain object is well preserved. Following is the motivation of our UI-Tree technique. Note that our analysis focuses on the range query as it is fundamental to other queries studied in the paper.

Suppose that we partition each uncertain object into l disjointed groups $\{g_i\}$ such that for any instance $x \in U_r$, x is contained by one and only one group. And each group g_i consists of the object identity, probability, and MBR of the group, which are denoted by $g.oid$, $g.p$, and $g.mbr$, respectively. Note that the probability of the group is the accumulation of the probability of all instances within that group. The advantage of the partition is immediate. As shown in Fig. 6, suppose that the uncertain object U is partitioned into 16 groups $\{g_1, g_2, \ldots, g_{16}\}$ and each group has probability $\frac{1}{16}$. Let $LP_{app}(U,Q)$ and $UP_{app}(U,Q)$ denote the lower and upper bounds of the appearance probability of uncertain object U regarding query Q, then we have $LP_{app}(U,Q) = \frac{2}{16}$ and $UP_{app}(U,Q) = \frac{6}{16}$.

Clearly, the larger the number of partitions, the better filtering capacity since the gap between $LP_{app}(U,Q)$ and $UP_{app}(U,Q)$ comes from the accumulated probabilities of the groups overlapping with query region Q_r. However, in order to construct an index with decent filtering capacity, we need to partition each uncertain objects into a certain number of groups. This number might be large, and hence, prevent the use of this approach in the applications with a large number of uncertain objects. So we consider to merge some groups of uncertain objects such that the index size can be reduced. Following is the first criterion for our index structure:

Index building criterion 1. In order to control the size of the index, we need to develop algorithm to merge the groups of partitions from uncertain objects.

As shown in Fig. 7a, suppose that six uncertain objects U_1, U_2, ..., U_6 are partitioned into groups by the dashed lines. Fig. 7b illustrates the merge result of these groups, denoted by $w_1, w_2, \ldots, w_{10}$. We call w_i a "word" in the spatial space. Suppose that w is the *word* constructed from a set of groups $\{g_i\}$, then w_{mbr} is the minimal bounding rectangle of all $g_i.mbr$s. Clearly, we prefer a w_{mbr} with small size. Fig. 7b illustrates the w_{mbr} created by merging MBRs of m groups. Assume that dimensionality of the space is d and the domain of each dimension is normalized to $[0,1]$ and let g_j and w_j denote the average length of $g.mbr$ and w_{mbr} at jth dimension, where $1 \le j \le d$. According to the analysis in [37], if there are totally $l \times n$ groups whose locations are *independent* of each other and every m of them whose MBRs are close to each

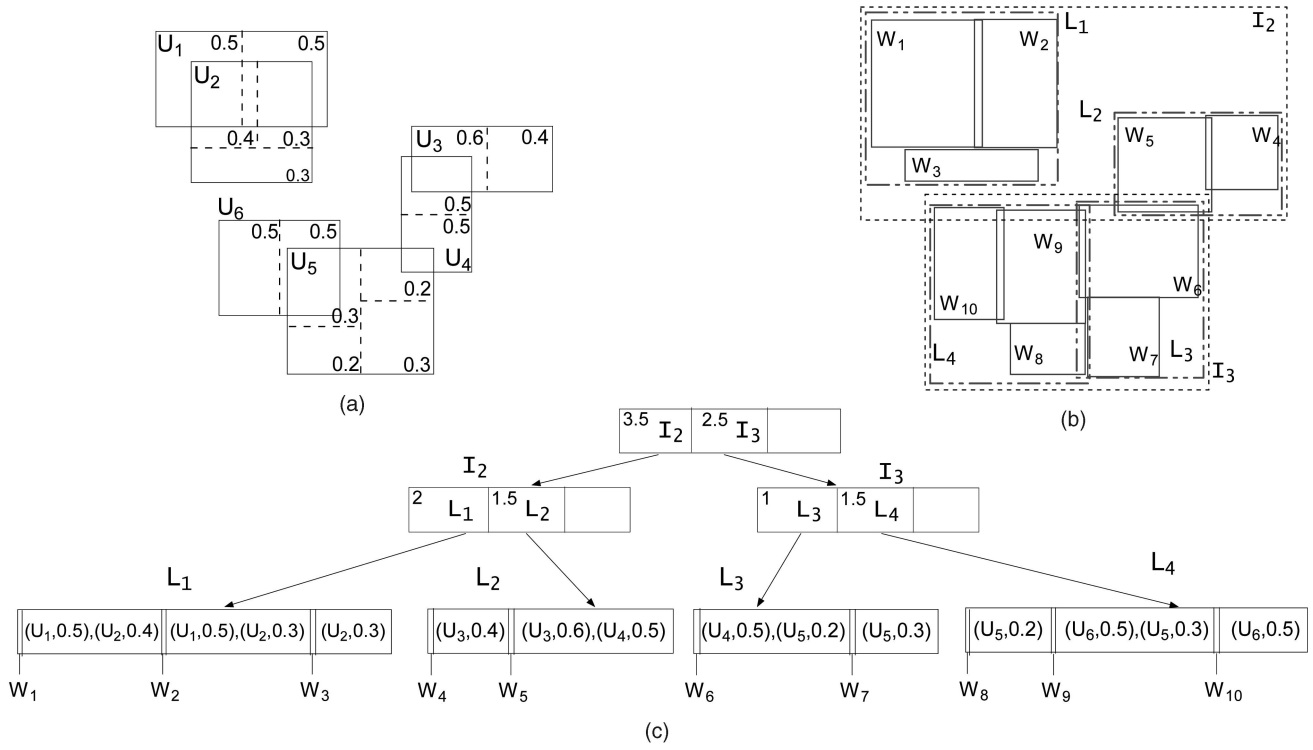


Fig. 7. R-tree-based inverted index.

other is merged to form k *words* where $k = \frac{l \times n}{m}$, we have $w_j = g_j + \triangle$, where

$$\triangle = \frac{m^{\frac{1}{d}} - 1}{(n \times l)^{\frac{1}{d}}}. \quad (1)$$

For the given m and l, (1) implies that the average length of w_{mbr} on each dimension decreases with the number of uncertain objects, which is confirmed in our experiment.

In order to distinguish groups from different uncertain objects and their probability values during the query processing, we have the second index building criterion as follows:

Index building criterion 2. The object identity and probability values for each group should be kept after it is merged.

The *inverted index* technique is employed to meet this criterion. For each word w, a posting list denoted by w_{list} is maintained to keep the object identity and probability of groups merged to w. w_{list} consists of a set of tuples $t(oid, p)$, where oid and p are the object identity of the group and its probability, respectively. The tuples in w_{list} are sorted in a decreasing order by the probability values. And w_p is the total probability of the tuples in w_{list}.

We use $t_{w,U}$ to denote a tuple that belongs to the posting list of *word* w with object identity U. $P(t_{w,U})$ denotes the probability of $t_{w,U}$. Note that the probability values of groups from the same uncertain object will be accumulated in the list. Suppose that we construct an index structure based on a set of words $\{w_i\}$ such that each group of the uncertain objects belongs to one and only one *word*. For uncertain object U, $\mathcal{W}(U)$ represents the set of *words* such that for any $w \in \mathcal{W}(U)$, there is a posting tuple $t_{w,U}$ in w_{list}. For presentation simplicity, we say $\mathcal{W}(U)$ is the *words* contained by U. And if U does not contain w, we have $P(t_{w,U}) = 0$. Let $\mathcal{W}$ and $|\mathcal{W}|$ denote all words in the space and its size. A small $|\mathcal{W}|$ implies that more groups are merged, and therefore, a small index size. On the other hand, the size of the posting list for each *word* increases with the number of partitions for each uncertain object. Therefore, we can control the size of the index by $|\mathcal{W}|$ and number of partitions for each uncertain object.

Theorem 1 indicates that we can compute the appearance probability bounds based on the inverted index of $\mathcal{W}$.

Theorem 1. *For a given query Q and an uncertain object U, let $\mathcal{W}_{con}$ ($\mathcal{W}_{over}$) denote the* words *in $\mathcal{W}(U)$ whose MBRs are contained (overlapped) by Q_r ($\mathcal{W}_{con} \cap \mathcal{W}_{over} = \emptyset$). We have*

$$LP_{app}(U, Q) = \sum P(t_{w,U}), where\ w \in \mathcal{W}_{con},$$
$$UP_{app}(U, Q) = \sum P(t_{w,U}), where\ w \in \mathcal{W}_{over} \cup \mathcal{W}_{con}.$$

Proof. For any instance $x \in U_r$ but $x \notin Q_r \cap U_r$, suppose that x is allocated to group g in the partition of uncertain object U. Let w denote the *word* g belonging to, it is immediate that $w \notin \mathcal{W}_{con}$ which implies that there is no instance $x \notin Q_r \cap U_r$ contributes to the lower bound computation. Similarly, for any instance $x \in Q_r \cap U_r$, it will contribute to the upper bound computation. So, the correctness of the lemma follows. □

Theorem 1 implies that the tightness of the bounds is affected by the *words* that overlap with query region Q_r. The probability of a *word* overlaps query region depends on the area of the MBR of w, denoted by $A(w)$, and the contribution to uncertainty is related to w_p, which is accumulated probabilities of all tuples in its posting list. So, we need to consider not only the area of MBRs of each *word* but also its probability value. For each *word* w, we use $WA(w)$ to represent $A(w) \times w_p$, called "weighted area" of the *word*. The third index building criterion is as follows:

Index building criterion 3. For the effectiveness of the index, given the number of *words* k, we want to create k *words* for the partitioned groups of uncertain objects such that $\sum_{1 \le i \le k} WA(w_i)$ is minimized.

Take the probabilistic threshold range query as an example, we further explain the intuition of this criterion based on the following lemma:

Lemma 1. *Suppose that $\mathcal{W}$ is constructed for uncertain object set $\mathcal{U}$. Then for a range query Q, we assume the probabilistic threshold θ is randomly chosen from $(0, 1]$ and the probability of each* word *overlapping with Q_r is independent of each other, which is denoted by $P_{over}(w)$. Then the expected size of candidate set C is as follows if Theorem 1 is applied for appearance probability computation:*

$$|C| = \sum_{w \in \mathcal{W}} P_{over}(w) \times w_p.$$

Proof. Let $\mathcal{W}_{over}(U, Q)$ denote the set $\{w\}$ such that w_{mbr} overlaps Q_r and $t_{w,U} \in w_{list}$. Given Q and $\mathcal{W}_{over}(U, Q)$, according to Theorem 1, $UP_{app}(U) = LP_{app}(U) + \sum P(t_{w,U})$, where $w \in \mathcal{W}_{over}(U, Q)$. Then U will be *validated* or *pruned* if $LP_{app}(U) \ge \theta$ or $UP_{app}(U) < \theta$. As θ is randomly chosen from $(0, 1]$, $LP_{app}(U) \ge 0$ and $UP_{app}(U) \le 1$, this implies that U will be kept in C with probability $\sum P(t_{w,U})$. So we have

$$|C| = \sum_{w \in \mathcal{W}(U)} \sum_{U \in \mathcal{U}} P(t_{w,U}) \times P_{over}(w)$$
$$= \sum_{w \in \mathcal{W}} P_{over}(w) \times w_p.$$

Recall that $P(t_{w,U}) = 0$ if U does not contain *word* w. □

As $P_{over}(w)$ depends on $A(w)$ and smaller $A(w)$ implies smaller chance to overlap with Q_r, we assume that $P_{over}(w) = A(w) \times c$, where c is a constant derived from Q_r. Then, it is immediate that we need to minimize $\sum_{1 \le i \le k} WA(w_i)$ for a small $|C|$ which is the measurement of filtering capacity of the index. Although the assumption of the independence of $P_{over}(w)$ among *words* and the existence of constant c is not practical in real world, this example does provide some insights for the index building criterion 3.

As a special case of the optimization problem in criterion 3 where all $w_i.p = 1$ and $k = 2$ is equivalent to the bipartition problem with measurement of area [24] which is NP-hard, we have to find some heuristics to solve this problem. Started with $n \times l$ *words*, each of which corresponds to a group of the uncertain objects where n is the number of uncertain objects and l is the number of partitions for each uncertain object, we can create the index with a greedy heuristic such that the total "weighted area" is minimized at each step in which one

word is merged. Nevertheless, this approach is infeasible to our problem as the computational complexity of the algorithms is cubic to $n \times l$ and multiscan of the groups is required, which leads to large number of IO operations.

In order to incrementally maintain the index structure in an efficient way, we employ R-Tree technique for the index construction because of its good support for spatial clustering [22]. Another important reason to apply R-Tree technique is because it can efficiently support a wide range of spatial queries. And this meets the fourth index building criterion.

Index building criterion 4. To efficiently support spatial queries which essentially depend on range query, the *words* should be well organized such that for the given query region Q_r, the *words* from $\mathcal{W}_{con}(Q)$ and $\mathcal{W}_{over}(Q)$ can be retrieved in an efficient way, where $\mathcal{W}_{con}(Q)$ ($\mathcal{W}_{over}(Q)$) denotes $w \in \mathcal{W}$ which is contained(overlapped) by query region Q_r.

To address the four index building criteria proposed, in the following part, we introduce the R-Tree-based inverted index technique for uncertain objects, named UI-Tree.

3.2 UI-Tree Structure

UI-Tree index is a depth-balanced tree structure similar to R-Tree [21] as illustrated in Fig. 7c and each node corresponds to a disk page. In this paper, we use I, L, and w to represent the nonleaf node, leaf node, and *word*, respectively. Each entry of the leaf node is a *word* with its posting list, represented by (w_{mbr},w_{list}). Note that for space efficiency, w_p will be computed on the fly based on posting entries in w_{list}. A set of entries is organized by a leaf node and the minimal bounding rectangle of the leaf node tightly contains the MBRs of the *words*. Note that for the index maintenance efficiency, if a *word* w occupies more than one page due to the large size of w_{list}, we simply create a new *word* w' to take half of the posting entries. As we have to keep a certain number of *words* for a decent filtering capacity, usually the size of w is not large. Because we aim to minimize the sum of $WA(w)$, the total probability of *words* in child entries is kept for each node to facilitate the tree structure maintenance. The nonleaf node of the UI-Tree is exactly the same as that of R-Tree except that the probability value is kept in its entry at parent node. Note that we do not keep any object identity information on the nonleaf node.

Suppose that the average size of each *word* is s_w and then the average node capacity (fan-out) of the leaf node is $\lfloor \frac{PageSize}{s_w} \rfloor$, denoted by f_l. The node capacity of nonleaf node is denoted by f_i. And the height h of a UI-Tree with k *words* is as follows:

$$h = 2 + \left\lceil \log_{f_i} \frac{k}{f_i \times f_l} \right\rceil. \quad (2)$$

If we regard the leaf node as a data entry, the UI-Tree corresponds to an R-Tree with $\frac{k}{f_l}$ data entries and an extra level for leaf nodes. Then, the (2) is immediate [18].

3.3 Index Maintenance

In this section, we first introduce the UI-Tree structure maintenance algorithms including uncertain object partition, insertion, and deletion.

3.3.1 Uncertain Object Partition

Before inserting an uncertain object into UI-Tree, we need to partition the uncertain object into l groups such that any instance $x \in U_r$ belongs to one and only one group. Ideally, we want to find l groups such that the sum of $A(g.mbr) \times g.p$ is minimized. As the partition is conducted on every uncertain object, the partition algorithm must be very efficient in terms of CPU time and number of IOs. If each uncertain object is already organized by some hierarchical tree structures such as R-Tree [21] and $Quad$-Tree [20], we can directly choose the intermediate node as the group since the instances of the uncertain object are naturally clustered. Otherwise, we employ a partition approach similar to kd-Tree [4]. Starting with one group which is the uncertain region of the uncertain object, we recursively partition the groups into two parts with the same probability value along a particular dimension chosen in a round-robin order. Suppose that the depth of the partition is d_p, then it comes up with $l = 2^{d_p}$ groups. For the *discrete* case, the partition procedure has time complexity of $O(d_p \times n_i)$, where n_i is the number of instances in the uncertain object. Recall that an instance of the uncertain object in *discrete* case corresponds to a possible occurrence of the uncertain object. This is because in each iteration, we can first find the median value of a set of n elements on the selected dimension with time complexity $O(n)$ [13] and then separate the groups into two parts with one scan. As to the *continuous* case, we can find the median value based on the cumulative density functions (CDFs) of the uncertain object and the partition cost is depended on CDF.

3.3.2 Insertion

The insert operation of UI-Tree is similar to R-Tree except that the probability value of the node is considered and we need to merge *words* to reduce the space. We can regard the node in the UI-Tree as a virtual *word* with empty posting list. Then, we redefine the area of the node as its "weighted area," and all of the operations in R-Tree which are related with area computation are updated such as *choose Leaf* and *node splitting* in UI-Tree.

In order to incrementally maintain the UI-Tree with limited space, we need to merge *words*. Let $w = merge(w_1, w_2)$ be the merged *word* from w_1 and w_2. Note that w_{mbr} is the minimal bounding rectangle of $w_{1_{mbr}}$ and $w_{2_{mbr}}$, while w_{list} consists of posting tuples of w_1 and w_2 in which tuples with the same object identity are merged. To measure the loss of information caused by merging two *words*, we define the *similarity* of two *words* w_1 and w_2 based on w:

$$sim(w_1, w_2) = \frac{1}{WA(w) - WA(w_1) - WA(w_2)}. \quad (3)$$

Note that we have $sim(w_1, w_2) = \infty$ when $w_1 = w_2$.

Clearly, we prefer to merge *words* with high similarity according to our index construction criteria. For a given k which is the maximal number of *words* the UI-Tree will maintain, we first randomly choose $\frac{k}{l}$ uncertain objects and partition them into k groups to build up the UI-Tree. The merge operation is not considered at this stage, so the procedure is the same as that of R-Tree except that the "weighted area" is considered. After this, we start to control the number of *words* by merging similar *words*. Note that a group g from the partition of U can be regarded as a *word* with one posting entry. Algorithm 1 illustrates the details of

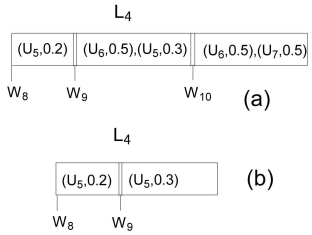


Fig. 8. Update.

the insertion algorithm. The flag $startmerge$ is set to $false$ before the UI-Tree construction.

After choosing the leaf node, the insertion procedure is simple if the merge stage does not start. Otherwise, we need to merge the most similar pair of *words* among *words* in L and g. Suppose that the most similar pair of current words and their similarity are kept in each leaf node denoted by L_{w_1}, L_{w_2}, and sim_L, respectively. We will merge g with the most similar *word* in the leaf node if their similarity value is greater than sim_L. Otherwise, L_{w_1} and L_{w_2} are merged, and g is inserted as a new *word*. Note that the split might be invoked as well after merging two *words* since a new posting entry is created although the number of *words* remains the same. After insertion, we need to update the related information (e.g., MBR, probability) on leaf nodes and its parents nodes. Following the index example in Fig. 7c, Fig. 8a demonstrates how the leaf node L_4 is updated after inserting a new uncertain object U_7.

For presentation simplicity, we use $m = \frac{l \times n}{k}$ to measure the *words* compression ratio of UI-Tree, named *merge factor*. Since the splitting procedure is quite complicated, we omit this part. Then the cost for an uncertain object insertion is $O(l \times (h \times f_i \times d + f_l^2 \times d))$ in the worse case, as the leaf node choosing takes time $O(f_i \times d)$ to find most similar subnodes at each level and the worst time complexity between Lines 5 and 14 is $O(f_l^2 \times d)$. And it takes time $O(f_i \times d)$ to adjust the nodes at each level.

Algorithm 1: ***Insertion***(UI, U)

```
Input : UI : the UI-Tree,
        U : uncertain object to be inserted
1  G := l groups from partition of U;
2  for each g ∈ G do
3      choose Leaf node L for g ;
4      if startmerge = false then
5          insert g to L as a new word ;
6          if number of words in UI reach k then
7              startmerge := true ;
8      else
9          w := the word most similar with g in L;
10         if sim(w, g) > sim_L then
11             w := merge(w, g);
12         else
13             L_w1 := merge(L_w1, L_w2) ;
14             Insert g to L as a new word ;
15             Update sim_L, L_w1, L_w2 ;
16     Adjust the UI-Tree by propagating changes;
```

3.3.3 Deletion

For uncertain object U to be deleted, we first descendantly find all *words* $\{w\}$, which include posting tuples $t_{w,U}$. Then, all tuples are removed from their corresponding posting list. The *words* with empty posting list are removed from the UI-Tree, which is the same as R-Tree. Based on the index example in Fig. 7c, Fig. 8b shows how the leaf node L_4 of the index is updated after deleting the uncertain object U_6. The delete operation is simple and efficient. However, it suffers from its inability to perform adjustment of MBR of the *word* if some posting tuples are removed. Because the cost of "shrinking," the MBR of a *word* w is expensive as we have to reload the uncertain objects which contribute to the w_{list}. Consequently, the filtering capacity of the UI-Tree might degrade if there are frequent deletions. Nevertheless, the UI-Tree is efficient in many of the real applications in which there are no frequent updates.

4 Query Processing

In this section, we introduce how to efficiently process various queries based on the UI-Tree proposed in Section 3. Section 4.1 presents our range query algorithm and related analysis. Then, we study the size estimation of the range query in Section 4.2, followed by the top-k range query and similarity join in Sections 4.3 and 4.4.

4.1 Range Query

In this section, we present a detailed searching algorithm for the probabilistic threshold range query based on the UI-Tree. For the given query Q, we descend the tree from the root in a manner similar to the R-Tree. All data entries(*words*) which are contained or overlapped by Q_r are retrieved, denoted by $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$, respectively. For each uncertain object U appeared in the posting lists of the *words*, we use U_{low} and U_{upper} to represent the lower and upper bounds for $P_{app}(U, Q)$ that can be computed based on $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$ according to Theorem 1. Then, all uncertain objects that cannot be filtered are kept in a candidate list C for *verification*. Algorithm 2 describes the range searching procedure.

The posting tuples of the list are visited in sequential order as shown in Fig. 10. A pointer, denoted by r_w, is employed to record currently visited tuple in the posting list w_{list}. We refer a tuple as *current* tuple in the posting list if it is recorded by the pointer. Let P_{max} denote the total sum of probability values of all current tuples from $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$. As the posting tuples are sorted in a decreasing order by their probability values, we can safely prune unseen uncertain objects once $P_{max} < \theta$. A maximal heap H is employed to maintain the pointers of the posting lists sorted by their probability values such that P_{max} can be reduced in a greedy way.

The total cost of the range query is $C_R + C_{cand} + C_{ver}$. Specifically, C_R is the cost for retrieving leaf nodes containing *words* in $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$, which is the same as the R-Tree range search. Let t_n denote the total number of posting tuples in $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$. The candidate set computation cost, denoted by C_{cand}, is $O(t_n \times \log w_n)$ in the worst case as the heap maintenance cost is $\log(w_n)$ for each iteration, where w_n is the number of *words* in $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$. C_{ver} is the cost for *verification* including exact appearance probability computation and some extra IO cost for loading uncertain objects.

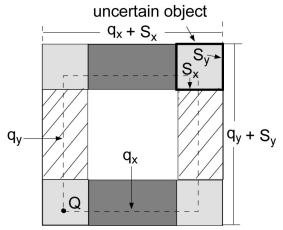


Fig. 9. Example.

Note that we do not discuss details of *verification* as the focus of the paper is to develop efficient index technique to reduce the number of candidates of the queries.

Algorithm 2: ***Range Query***(UI, Q, θ)

Input : UI : the UI-Tree over uncertain objects set $\mathcal{U}$,
Q : query with region Q_r,
θ : Probabilistic threshold
Output: R : uncertain objects with $P_{app}(U, Q) \geq \theta$

$\mathcal{L}$:= all leaf nodes in UI contained or overlapped by Q_r;
$\mathcal{W}_{con}$:= *words* with w_{mbr} contained by Q_r from $\mathcal{L}$;
$\mathcal{W}_{over}$:= *words* with w_{mbr} overlaps with Q_r from $\mathcal{L}$;
$C := \emptyset$; $R := \emptyset$;
$H := \emptyset$; $P_{max} := 0$;
for each first posting tuple $t \in \mathcal{W}_{con} \cup \mathcal{W}_{over}$ **do**
  $r_w := t$; $P_{max} := P_{max} + t_p$;
  put r_w into H ;
while $H \neq \varnothing$ and $P_{max} \geq \theta$ **do** /* filtering */
  Remove top pointer r_w from H ;
  t := the posting tuple r_w referred ;
  U := the uncertain object with identity $t.oid$;
  if U is not *pruned* or *validated* **then**
    if t is from $\mathcal{W}_{con}$ **then** /* contain */
      $U_{low} := U_{low} + t_p$;
      $U_{upper} := U_{upper} + t_p$;
    else /* overlap */
      $U_{upper} := U_{upper} + t_p$;
    if $U_{upper} + P_{max} < \theta$ **then**
      U is pruned;
    else if $U_{low} > \theta$ **then**
      $R := R \cup U$;
  if t is not the last tuple **then**
    Let r_w point to next tuple; put r_w into H;
  update P_{max} ;
Refine the C by visiting remainding tuples ;
for each $U \in C$ **do** /* verification */
  if $P_{app}(U, Q) \geq \theta$ **then**
    $R := R \cup U$;
return R

Estimate the filtering capacity. In the following part, with some uniformity assumptions, we analyze the performance of the range query by estimating the number of candidates since it reflects the filtering capacity of the index. For presentation simplicity, we assume that the domain sizes of all dimensions are between $[0, 1]$ and the dimensionality is 2. Suppose the uncertain regions of the uncertain objects are

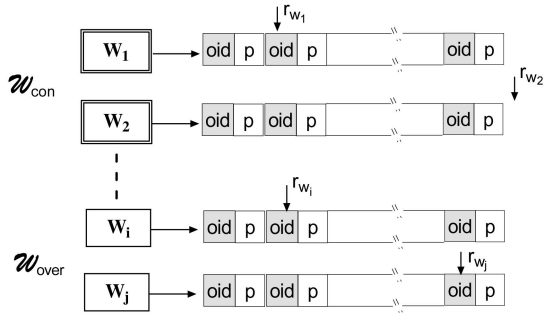


Fig. 10. Query example.

regular rectangles whose instances follow the $uniform$ distribution and the probabilistic thresholds are randomly chosen from $(0, 1]$. As shown in Fig. 9, we assume that the query region is larger than the uncertain region of uncertain object on each dimension. Let s_x and s_y denote the average lengths of the rectangle on x and y dimensions, respectively. A query Q has a regular rectangle region with length q_x and q_y, which is issued with a randomly selected center.

First, we assume that there is no merge operation during the index construction ($m = 0$). Let the probability $p_{cx} + p_{cy} + p_n$ represent the probability that Q_r overlaps U_r. Specifically, $p_{cx} = 2 \times (q_x - s_x) \times s_y$ is the probability of Q_r covering U_r at x dimension and $p_{cy} = 2 \times (q_y - s_y) \times s_x$ represents the probability of Q_r covering U_r at y dimension. While $p_n = 4 \times s_x \times s_y$ denotes the probability that Q_r overlaps U_r but does not cover U_r in any dimension. As shown in Fig. 9, when the left bottom corner of the query Q falls in the light gray rectangles with total area size $4 \times s_x \times s_y = p_n$, Q_r overlaps U_r but does not cover x or y dimension of U_r. Similarly, p_{cx} and p_{cy} correspond to the total area of dark gray rectangles and rectangles with strike lines.

Theorem 2 evaluates the expected candidate size for Q for $m = 0$.

Theorem 2. *Let C denote the set of uncertain objects in the candidate set in Algorithm 2. For proof simplicity, we assume that P_{max} is not considered in Algorithm 2. Suppose the uncertain region of each uncertain object is partitioned into $n_x \times n_y$ cells with same size, then the average size of C can be estimated by $n \times (\frac{p_{cx}}{n_y} + \frac{p_{cy}}{n_x} + p_n \times \frac{(n_x + n_y)}{2 \times n_x \times n_y})$, where n is the number of uncertain objects.*

Proof. According to the description of Algorithm 2, an uncertain object U contributes to C if and only if U_r overlaps Q_r and $U_{low} < \theta \leq U_{upper}$. Let O_r denote the region such that Q_r overlaps U_r when the center of Q, denoted by q_c, falls in O_r. Since the probabilistic threshold θ is randomly chosen from $(0, 1]$, the probability of $U \in C$ is

$$P_{U \in C} = \int_{x \in O_r} pdf(x) \int_0^1 f(\theta, x) d\theta dx$$
$$= \int_{x \in O_r} pdf(x) D(x) dx,$$

where $pdf(x)$ is the probabilistic density function of x. We have $f(\theta, x) = 0$ if $\theta > U_{upper}$ (being *pruned*) or $\theta \leq U_{low}$ (being *validated*), otherwise, $f(\theta, x) = 1$. We use $D(x)$ to represent the difference between U_{upper} and U_{low} when q_c locates at position x. According to Theorem 1, we have $D(x) = \sum_{w \in \mathcal{W}_{over}(U)} P(t_{w,U})$. For instance, $D(x)$

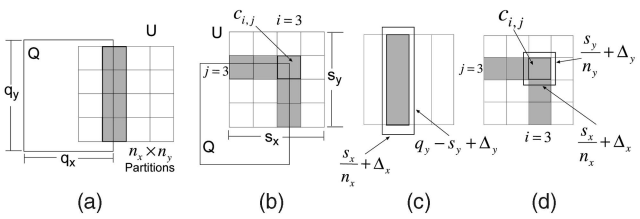


Fig. 11. Filtering capacity evaluation.

corresponds to the accumulated probability values of the shaded cells in Figs. 11a and 11b.

As we assume that the rectangle region of the query is larger than that of the uncertain object in every dimension, there are three possible cases in which Q_r *overlaps* U_r:

- $E1$: Q_r covers U_r in y dimension but not in x dimension.
- $E2$: Q_r covers U_r in x dimension but not in y dimension.
- $E3$: Q_r does not cover U_r in any dimension.

Case $E1$ is illustrated in Fig. 11a. According to the uniformity assumptions and (4), we have

$$P_{E1} = p_{cy} \times \frac{1}{n_x},$$

where P_{E1} is probability of occurring of $E1$ and $U \in C$, p_{cy} is the occurrence probability of $E1$ and $p_{cy} = 2 \times (q_y - s_y) \times s_x$. Similarly, we have $P_{E2} = p_{cx} \times \frac{1}{n_y}$ and $p_{cx} = 2 \times (q_x - s_x) \times s_y$. For case $E3$, Fig. 11b illustrates one of its four subcases in which the upper right corner of Q_r falls in U_r. Based on (4) and uniformity assumptions, we have

$$\begin{aligned} P_{E3} &= 4 \times \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} \frac{p_n}{4 \times n_x \times n_y} \times (i + j - 1) \times \frac{1}{n_x \times n_y} \\ &= p_n \times \frac{(n_x + n_y)}{2 \times n_x \times n_y}, \end{aligned}$$

where p_n denotes the occurring probability of $E3$, which is $4 \times s_x \times s_y$. Since $P_{U \in C} = P_{E1} + P_{E2} + P_{E3}$, according to the uniformity assumptions, the expected size of C is

$$E(C) = n \times P_{U \in C} + 0 \times P_{U \notin C} = n \times P_{U \in C},$$

where n is the number of uncertain objects. □

Once the merge procedure is involved, let Δ_x and Δ_y denote the average increment of the MBRs of the partitions on x and y dimensions, respectively. The following theorem evaluates the estimated candidate size for Q. Clearly, the more partitions of the uncertain objects are merged, the larger Δ values, and hence, less filtering power:

Theorem 3. *Suppose that a UI-Tree is constructed based on the partitions of the uncertain objects and the uncertain region of each uncertain object is partitioned into $n_x \times n_y$ cells with same size, then the expected size of C can be estimated by $n \times (\frac{p_{cx}}{n_y} + \frac{p_{cy}}{n_x} + p_n \times \frac{(n_x+n_y)}{2 \times n_x \times n_y})$, where n is the number of uncertain objects, $p_{cx} = 2 \times (q_x - s_x + \Delta_x) \times (\frac{s_y}{n_y} + \Delta_y) \times n_y$, $p_{cy} = 2 \times (q_y - s_y + \Delta_y) \times (\frac{s_x}{n_x} + \Delta_x) \times n_x$, and $p_n = 4 \times (\frac{s_x}{n_x} + \Delta_x) \times (\frac{s_y}{n_y} + \Delta_y) \times n_x \times n_y$. Δ_x and Δ_y denote the average increment of MBRs of the partitions after merge procedure on x and y dimensions, respectively.*

Proof. As the MBRs of the partitions for the uncertain objects are merged during the index construction, the probability that Q_r overlaps those partitions increases. For instance, as shown in Fig. 11c, the probability that the left boundary of Q_r overlaps the ith column in case $E1$ of Theorem 2 becomes $(q_y - s_y + \Delta_y) \times (s_x + \Delta_x)$. Similarly, Fig. 11d illustrates that the upper right corner of the query region falls in each partition with probability $(\frac{s_x}{n_x} + \Delta_x) \times (\frac{s_y}{n_y} + \Delta_y)$. Following the same rational of Theorem 2, the correctness of the theorem is immediate. □

4.2 Size Estimation of Range Query

Instead of retrieving the uncertain objects qualifying the range query, it suffices to get the approximated number of uncertain objects in some applications. One of the important observations in the field of multidimensional selectivity estimation is that although the whole data set is unlikely to follow the uniform distribution in real applications, it might be true within a local area. This motivates us to estimate the selectivity of the range query based on the uniformity assumption of the instances in the MBRs of their corresponding *words*. Then instead of keeping lower and upper bounds, the appearance probability of the uncertain object U regarding query Q can be estimated with the following formula:

$$\begin{aligned} P_{app}(U, Q) = &\sum_{w \in \mathcal{W}_{over}} P(t_{w,U}) \times \frac{A(Q_r \cap w_{mbr})}{A(w_{mbr})} \\ &+ \sum_{w \in \mathcal{W}_{con}} P(t_{w,U}). \end{aligned}$$

This implies that we can estimate the size of the range query based on the UI-Tree only. Our experiments demonstrate the effectiveness of the estimation.

4.3 Top-k Range Query

For a given query Q, once we retrieve $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$, the remaining part of the query processing is similar to the traditional top-k computation on distributive inverted indexes [17]. The main difference is that the posting tuples from $\mathcal{W}_{over}$ only contribute to the appearance probability upper bound of uncertain object. In our implementation, we modify Algorithm 2 such that we can safely claim that all of the unseen uncertain objects cannot be top-k answers once $P_{max} \leq p_{lk}$, where p_{lk} is the kth largest U_{low}. After the refinement, all uncertain objects with $U_{upper} < p_{lk}$ can be *pruned* and the ones with $U_{low} \geq p_{uk}$ are *validated* where p_{uk} is the kth largest U_{upper}. Then we need to compute the exact $P_{app}(U, Q)$ for the uncertain objects in the candidate set to decide the top-k result.

4.4 Similarity Join

As the algorithm for probabilistic threshold similarity join is lengthy, we only introduce the outline of the algorithm. Let $\mathcal{U}$ and $\mathcal{V}$ denote two sets of uncertain objects. The distance and probabilistic threshold are represented by γ and θ, respectively. First, we retrieve all pairs of *words* w_s, w_r based on the traditional spatial join algorithm [8] such that $|w_{s_{mbr}} - w_{r_{mbr}}|_{min} \leq \gamma$, where $|w_{s_{mbr}} - w_{r_{mbr}}|_{min}$ denotes the

minimal euclidean distance between $w_{s_{mbr}}$ and $w_{r_{mbr}}$. These pairs are classified into two sets: $\mathcal{W}^*{}_{in}$ and $\mathcal{W}^*{}_{part}$. For any $(w_s, w_r) \in \mathcal{W}^*{}_{in}$, $|w_{s_{mbr}} - w_{r_{mbr}}|_{max} \leq \gamma$, which implies that the distance between any pair of instances of uncertain objects in w_s and w_r is smaller than or equal to γ. Other pairs belong to $\mathcal{W}^*{}_{part}$. Let $LP(U \bowtie_\gamma V)$ and $UP(U \bowtie_\gamma V)$ denote the lower and upper bounds for the similarity between uncertain objects U and V. Clearly, pairs from $\mathcal{W}^*{}_{in}$ contribute to the computation of $LP(U \bowtie_\gamma V)$ and $UP(U \bowtie_\gamma V)$, while the ones from $\mathcal{W}^*{}_{part}$ only contribute to $UP(U \bowtie_\gamma V)$. Similar to Theorem 1, we have the following formulas for *verification* and *pruning*, respectively:

$$LP(U \bowtie_\gamma V) = \sum_{(w_s, w_r) \in \mathcal{W}^*{}_{in}} P(t_{w_s,U}) \times P(t_{w_r,V}),$$

$$UP(U \bowtie_\gamma V) = \sum_{(w_s, w_r) \in \mathcal{W}^*{}_{in} \cup W^*_{part}} P(t_{w_s,U}) \times P(t_{w_r,V}).$$

The correctness of the formulas is immediate based on the same rationale of Theorem 1. To compute the candidate efficiently, a pointer is employed for each posting list in a similar manner with Algorithm 2. All of the pairs are organized by a maximal heap sorted by the probability values of their corresponding posting entries. Let PS and PR denote the pointers in $\mathcal{U}$ and $\mathcal{V}$, respectively. Let PS_{max} and PR_{max} keep the maximal possible probability for unseen uncertain objects from $\mathcal{U}$ and $\mathcal{V}$, respectively. Clearly, once $PS_{max} \times PR_{max} < \theta$, the pair (U, V) does not belong to the candidate set if U and V are unvisited.

5 Performance Evaluation

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

- UI**-Tree:** The R-Tree-based inverted index technique proposed in Section 3 and four query algorithms presented in Section 4.
- U**-Tree:** The U-Tree technique presented in [36]. The implementation is publicly available.
- R**-Tree:** The uncertain region-based R-Tree technique. The implementation of similarity join is based on the join strategy proposed in [27]. As there is no existing work on the size estimation of range query and top-k range query on uncertain objects with arbitrary PDF, the R-Tree technique is also employed as baseline algorithm because it can be regarded as a special case of UI-Tree in which every uncertain objects is a *word*.

In our experiment, the uncertain region of the uncertain object is a circle or sphere with radius r_u varying from 50 to 500 with default value 100. Suppose that the PDF of an uncertain object is described by 400 instances (*discrete* case), which follow two popular distributions *Normal* and *Uniform*. The *Normal* serves as default distribution with standard deviation $\frac{r_u}{2}$. Specifically, we use the *constrained normal* distribution such that the possible location of the instances is restricted in the uncertain region. Instances might be loaded into memory once an uncertain object is required for *verification*. For *verification* efficiency, same as [5], the instances of an uncertain object are organized by an aggregate R-Tree, where aggregate value of a R-Tree node is the accumulation of the probabilities of its child instances. Note that each instance corresponds to one bin in [5] since we consider the *discrete* case in the experiment.

TABLE 2
System Parameters

Notation	Definition (Default Values)
r_q	the radius of query (1000)
r_u	the radius of uncertain object region(100)
n	number of uncertain object
θ	probabilistic threshold ($\in (0, 1]$)
k	k value in top-k query
m	merge factor (12)
l	number of partitions(32)

Two real spatial data sets, *CA* and *US*, are employed to represent the center of the uncertain regions. They contain 62 K and 200 K two-dimensional points representing locations in Los Angeles and the United States, respectively. We also generate synthetic data set *3D* with dimensionality 3 and size 200 K in which the centers and instances of uncertain objects are uniformly distributed. All dimensions are normalized to domain $[0, 10000]$ and *CA* with *constrained normal* distribution is employed as the default data set. To study the similarity join between two sets of uncertain objects, two synthetic data sets, named $2d$ 10K and $2d$ 1K, are created in which the centers of uncertain objects follow *Uniform* distribution and the instances follow *constrained normal* distribution.

A workload for range query and its size estimation query consists of 200 queries in our experiment. Same as [36], the region of a range query Q is a circle or sphere with radius r_q. r_q varies from 500 to 1,500 with default value 1,000. The centers of the queries are randomly chosen from the centers of the target uncertain objects. Note that the query regions in a workload share the same r_q. In order to avoid favoring particular θ value, we randomly choose the probabilistic threshold $\theta \in (0, 1]$ for each query. Instead of specifying the probabilistic threshold θ, the value k varying from 50 to 250 is used for the top-k query.

As all of the algorithms investigated in the paper follow the *filtering and verification* frameworks, the cost of the query is largely dependent on the candidate size as the *verification* is expensive in terms of IO and CPU time. So the average candidate size of the queries is employed as the most important performance measurement in our experiments. In addition, the average number of IOs and false positives are recorded as well as the average query response time.

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 2.40 GHz dual CPU and 4 GB memory running Debian Linux. The disk page size is fixed to 4,096 bytes. In order to achieve a good filtering capacity, the catalog size of U-Tree is set to 9 for CA and *US*, and 10 for $3D$ as suggested in [35].

Table 2 lists the parameters which may potentially have an impact on our performance study. In our experiments, all parameters use default values unless otherwise specified.

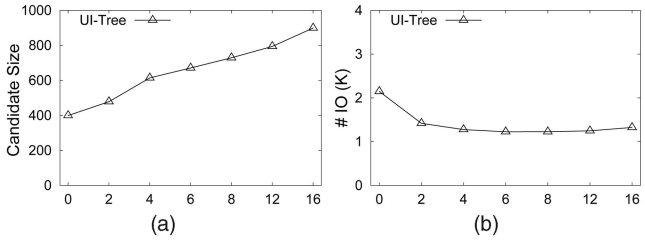


Fig. 12. Index evaluation against diff. m. (a) Candidate size versus m. (b) IO versus m.

TABLE 3
Index Size Comparison

	CA	*US*	$3D$
UI-Tree	15 (M)	49 (M)	80 (M)
U-Tree	15 (M)	49 (M)	86 (M)

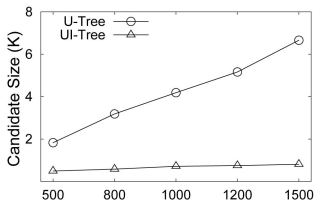


Fig. 13. Diff. r_q.

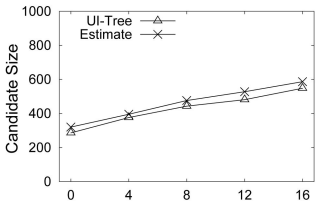


Fig. 14. Diff. m.

5.1 Index Construction Evaluation

In this section, we evaluate the performance of UI-Tree construction algorithm. The size of the UI-Tree depends on the number of uncertain objects n, merge factor m, and the number of partitions per uncertain object l. Clearly, smaller m and larger l will lead to better filtering capacity but more index space. For comparison convenience, we fix l to 32 and vary m to tune the index size such that it is similar to that of U-Tree. Under the default setting, Fig. 12a shows that the filtering capacity of the UI-Tree slowly decreases with m. Note that $m = 0$ implies there is no merge operation. Meanwhile, the index size also drops from 62 to 15 M. Fig. 12b reports the average IO cost for range query with default setting, where m varies from 0 to 16. Although the number of candidates is small for small m, it might invoke more IO cost because of the large index size. By default, we set m to 12 for *CA* and *US*, and 6 for $3D$, respectively. So the UI-Tree has similar index size with U-Tree.

Table 3 shows the index sizes of UI-Tree and U-Tree for *CA*, *US*, and $3D$, respectively, which correspond to around 5, 5, and 6 percent of the data sets.

Note that since the U-Tree code from [36] does not employ the memory buffer during the U-Tree construction, the index construction is very slow. So we do not evaluate its construction time in this paper for fairness of comparison. Our index construction algorithm is very efficient, and the average insertion time per object for *CA*, *US*, and $3D$ is around 1, 4, and 4 ms, respectively. More specifically, for *CA*, it totally takes 118 s for partition, and 17 s (164 s) for insertion without(with) merge operation.

5.2 Query Performance Evaluation

In this section, we first evaluate the performance of range query. It is followed by size estimation, top-k range query, and similarity join. As U-Tree is the state-of-the-art technique for range query on multidimensional uncertain objects with arbitrary PDF, it is employed as baseline algorithm to evaluate our UI-Tree-based range query algorithm.

In order to confirm the observation of Fig. 2a in Section 2.2, we construct a set of queries whose query regions are rectangles with length $2 \times r_q$ and width 300. The center of the query is randomly chosen from *CA* data and we rotate the rectangle around its center to a random angle between 0 and 2π. Fig. 13 demonstrates that the filtering capacity of U-Tree degrades significantly with the growth of r_q from 500 to 1,500, while the performance of UI-Tree technique is much more efficient and less sensitive to r_q.

To confirm the effectiveness of filtering capacity estimation, we also conduct filtering capacity evaluation on two-dimensional synthetic data. There are 100 K uncertain objects evenly distributed and the uncertain regions of the uncertain objects are squares with width 200. The instances follow the *Uniform* distribution and query regions are regular rectangle queries with $q_x = q_y = 1{,}600$. We build UI-Tree indices with m varying from 0 to 16. Note that the candidate size estimation for $m = 0$ is based on Theorem 2 while others are based on Theorem 3 since the merge procedure is involved for $m > 0$. Fig. 14 shows that estimated values are close to the real candidate size.

In the third set of experiments, we evaluate the performance of UI-Tree and U-Tree against different data sets (*CA*, *US*, and $3D$) while other system parameters are set to default values. The candidate size, number of false positives, number of IOs, and response time of two techniques are reported in Fig. 15. Due to poor filtering capacity, the candidate size of U-Tree is much larger than that of the UI-Tree especially on *US* as shown in Fig. 15a. A similar observation is found in Fig. 15b, which reports the number of false positives. According to Figs. 15c and 15d, U-Tree and UI-Tree have similar filtering cost in terms of index IO and filtering time. However, due to the large number of candidates as shown in Fig. 15a, the total cost of the U-Tree for range query is much more expensive than that of the UI-Tree in terms of IO and query response time.

To evaluate the impact of the instance distribution of uncertain objects, we report the number of IOs against different instance distributions. For the "arbitrary" distribution, the instances of the uncertain object are created by mapping 400 closest points for given random point in real data set *US* into the uncertain region of the uncertain object. Fig. 16 shows that the performance of the algorithm is not sensitive to these distributions.

The R-Tree-based index technique in [5] can be used to support range query, where the uncertain regions of the

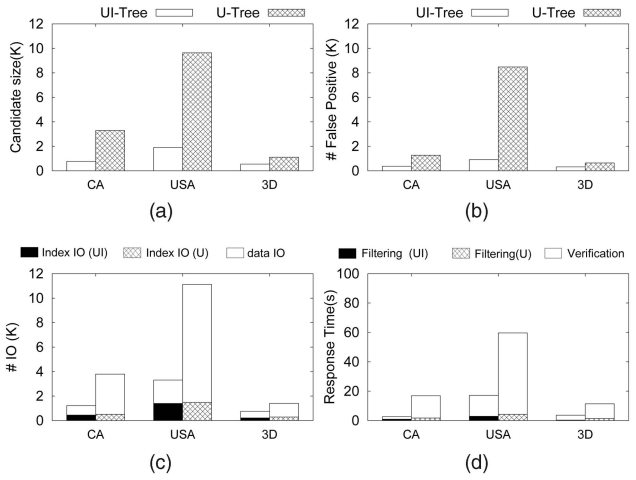


Fig. 15. Performance versus diff. data sets. (a) Candidate size. (b) False positive. (c) IO. (d) Response time.

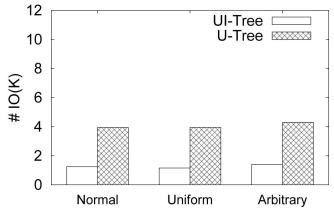


Fig. 16. Diff. distribution.

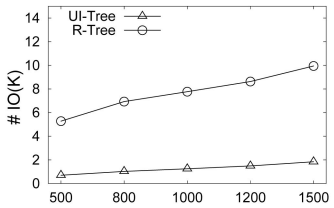


Fig. 17. Diff. r_q.

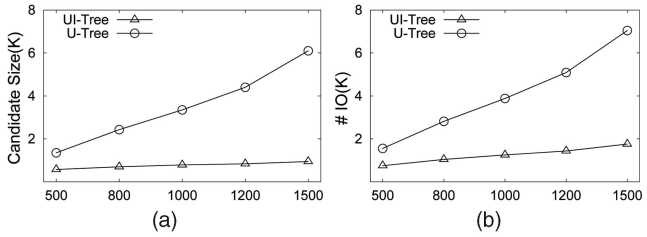


Fig. 18. Performance versus diff. r_q size. (a) Candidate size versus r_q. (b) *IO* versus r_q.

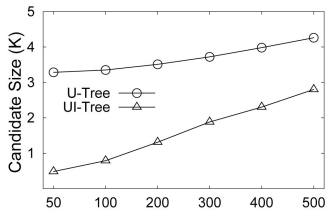


Fig. 19. Diff. r_u.

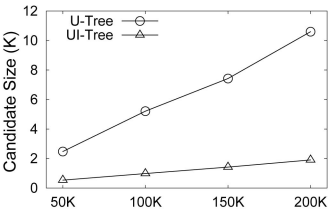


Fig. 20. Diff. n.

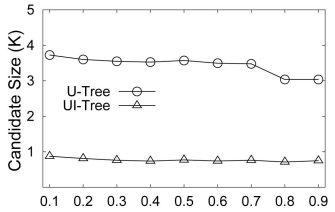


Fig. 21. Diff. θ.

uncertain objects are indexed by a global R-Tree and the PDF of each uncertain object is represented by a set of bins (histograms) organized by an aggregate R-Tree. In our experiment, each instance corresponds to a bin as we consider the *discrete* case. Fig. 17 demonstrates that UI-Tree significantly outperforms the R-Tree-based index technique.

We evaluate the impact of r_q against the candidate size and the number of IOs. Fig. 18 shows that as r_q increases from 500 to 1,500, the performance of U-Tree drops significantly while UI-Tree is more efficient and stable against r_q. In Fig. 18a, the candidate size of U-tree reaches 6 K when $r_q = 1{,}500$, which is six times larger than that of UI-Tree. And the same trend goes to the number of IOs as depicted in Fig. 18b.

We study the scalability of U-Tree and UI-Tree by varying r_u and n. The results are reported in Figs. 19 and 20, respectively. As expected, the number of candidates of both techniques increases with r_u as the larger uncertain region implies more uncertain objects overlapping with query region. In Fig. 20, a uniform sample of 50, 100, and 150 K uncertain objects from *US* and *US* (200 K) are employed to evaluate the impact of n. It is not surprising that the number of candidates goes up when the number of uncertain objects increases. Nevertheless, the growth of UI-Tree is much slower than that of the U-Tree. This is because the filtering capacity of the PCRs of an uncertain object is independent of n. So, the number of candidates goes linearly with n. According to the analysis in Section 3.1, for the fixed merge factor m, a larger n implies a smaller w_{mbr}. Consequently, the number of candidates grows very slowly with n because the filtering capacity of individual *word* improves due to a smaller w_{mbr}.

Fig. 21 demonstrates that the probabilistic threshold does not have much impact on the candidate size of U-Tree and UI-Tree. And the performance of U-Tree slightly improves when θ is large but is still less competitive compared with that of the UI-Tree.

In the last set of experiments, we evaluate the performance of UI-Tree-based query algorithms for size estimation of range query, top-k range query, and similarity join. The R-Tree-based techniques are employed as baseline Algorithms. Let S and $\bar{S}$ denote the exact and estimated number of uncertain objects returned by range query, respectively.

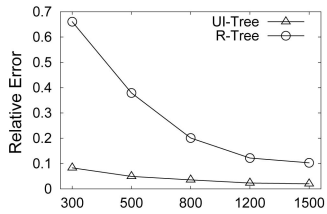


Fig. 22. Size est. versus r_q.

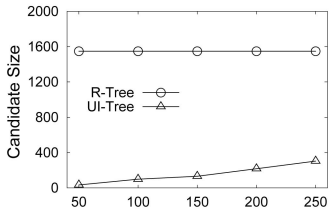


Fig. 23. Top-k (diff. k).

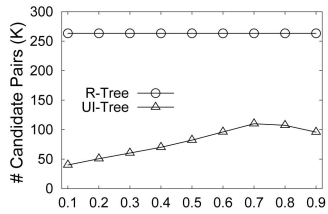


Fig. 24. Similarity join (diff. θ).

Then, we measure the effectiveness of the size estimation query by relative error $|\frac{S-\bar{S}}{S}|$. The average relative error of the queries under default setting is reported in Fig. 22. As expected, the accuracy of UI-Tree technique is much better than that of the R-Tree technique. It is 0.02 when r_q equals 1,500.

We evaluate the number of candidates in top-k queries with k varying from 50 to 250. r_q is set to 300 and Fig. 23 shows that only a small number of candidates are required for further *verification* based on UI-Tree technique. There is no surprise to see that the number of candidates is large and remains unchanged with k for R-Tree technique as it does not capture any details of the PDF of uncertain objects. Similar observation is reported in Fig. 24 in which data sets $2d\ 10K$ and $2d1K$ are joined with different probabilistic threshold values varying from 0.1 to 0.9. The distance is set to 600.

6 Conclusion

In this paper, we investigate the problem of indexing multidimensional uncertain objects with arbitrary PDFs and propose a novel index structure UI-Tree. Combining R-Tree and inverted index techniques, UI-Tree is space effective and well supports different shapes of query regions. UI-Tree serves as a general index, where efficient processing of range queries is desirable. We propose solutions for various types of queries based on UI-Tree, including range query, size estimation of range query, top-k range query, and similarity join. Our extensive experiments demonstrate that UI-Tree outperforms the previous studies. In the future, we will further investigate how to apply the UI-Tree technique to efficiently manage and mining spatial uncertain data, including nearest neighbor query, reverse nearest neighbor query, probabilistic skyline computation, and clustering.

References

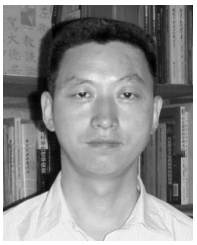
[1] C. Aggarwal and P. Yu, "On High Dimensional Indexing of Uncertain Data," *Proc. Int'l Conf. Data Eng. (ICDE),* 2008.
[2] C.C. Aggarwal, *Managing and Mining Uncertain Data.* Springer, 2009.
[3] C.C. Aggarwal and P.S. Yu, "A Framework for Clustering Uncertain Data Streams," *Proc. Int'l Conf. Data Eng. (ICDE),* 2008.
[4] J.L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Comm. ACM,* vol. 18, no. 9, pp. 509-517, 1975.
[5] G. Beskales, M.A. Soliman, and I.F. Ilyas, "Efficient Search for the Top-k Probable Nearest Neighbors in Uncertain Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* vol. 1, no. 1, 2008.
[6] C. Böhm, M. Gruber, P. Kunath, A. Pryakhin, and M. Schubert, "Prover: Probabilistic Video Retrieval Using the Gauss-Tree," *Proc. Int'l Conf. Data Eng. (ICDE),* 2007.
[7] C. Böhm, A. Pryakhin, and M. Schubert, "Probabilistic Ranking Queries on Gaussians," *Proc. Int'l Conf. Scientific and Statistical Database Management (SSDBM),* 2006.
[8] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient Processing of Spatial Joins Using R-Trees," *Proc. ACM SIGMOD,* 1993.
[9] J. Chen and R. Cheng, "Efficient Evaluation of Imprecise Location-Dependent Queries," *Proc. Int'l Conf. Data Eng. (ICDE),* 2007.
[10] R. Cheng, D.V. Kalashnikov, and S. Prabhakar, "Evaluating Probabilistic Queries Over Imprecise Data," *Proc. ACM SIGMOD,* 2003.
[11] R. Cheng, D.V. Kalashnikov, and S. Prabhakar, "Querying Imprecise Data in Moving Object Environments," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 9, pp. 1112-1127, Sept. 2004.
[12] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J.S. Vitter, "Effcient Indexing Methods for Probabilistic Threshold Queries Over Uncertain Data," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2004.
[13] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms,* second ed. The MIT Press, 2001.
[14] X. Dai, M. Yiu, N. Mamoulis, Y. Tao, and M. Vaitis, "Probabilistic Spatial Queries on Existentially Uncertain Data," *Proc. Int'l Symp. Large Spatio-Temporal Databases (SSTD),* 2005.
[15] N. Dalvi and D. Suciu, "Efficient Query Evaluation on Probabilistic Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2004.
[16] Z. Ding, "Utr-Tree: An Index Structure for the Full Uncertain Trajectories of Network-Constrained Moving Objects," *Proc. Int'l Conf. Mobile Data Management (MDM),* 2008.
[17] R. Fagin, A. Lotem, and M. Naor, "Optimal Aggregation Algorithms for Middleware," *J. Computer and System Sciences,* vol. 66, no. 4, pp. 614-656, 2003.
[18] C. Faloutsos, T.K. Sellis, and N. Roussopoulos, "Analysis of Object Oriented Spatial Access Methods," *Proc. ACM SIGMOD,* 1987.
[19] I.D. Felipe, V. Hristidis, and N. Rishe, "Keyword Search on Spatial Databases," *Proc. Int'l Conf. Data Eng. (ICDE),* 2008.
[20] R.A. Finkel and J.L. Bentley, "Quad Trees: A Data Structure for Retrieval on Composite Keys," *Acta Informatica,* vol. 4, pp. 1-9, 1974.
[21] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD,* 1984.
[22] J. Han and M. Kamber, *Data Mining: Concepts and Techniques.* Diane Cerra, 2001.
[23] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems," *Proc. Int'l Conf. Scientific and Statistical Database Management (SSDBM),* 2007.
[24] J. Hershberger and S. Suri, "Finding Tailored Partitions," *Proc. Symp. Computational Geometry,* 1989.
[25] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach," *Proc. ACM SIGMOD,* 2008.
[26] D.-O. Kim, D.-S. Hong, H.-K. Kang, and K.-J. Han, "Ur-Tree: An Efficient Index for Uncertain Data in Ubiquitous Sensor Networks," *Proc. Int'l Conf. Advances in Grid and Pervasive Computing (GPC),* 2007.

[27] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz, "Probabilistic Similarity Join on Uncertain Data," *Proc. Int'l Conf. Database Systems for Advanced Applications (DASFAA),* 2006.
[28] H.-P. Kriegel, P. Kunath, and M. Renz, "Probabilistic Nearest-Neighbor Query on Uncertain Objects," *Proc. Int'l Conf. Database Systems for Advanced Applications (DASFAA),* 2007.
[29] R. Li, B. Bhanu, C. Ravishankar, M. Kurth, and J. Ni, "Uncertain Spatial Data Handling: Modeling, Indexing and Query," *Computers & Geosciences,* vol. 33, no. 1, pp. 42-61, 2007.
[30] Y. Ma, D.V. Kalashnikov, and S. Mehrotra, "Toward Managing Uncertain Spatial Information for Situational Awareness Applications," *IEEE Trans. Knowledge and Data Eng.,* vol. 20, no. 10, pp. 1408-1423, Oct. 2008.
[31] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic Skylines on Uncertain Data," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2007.
[32] Y. Sadahiro, "Buffer Operations on Spatial Data with Limited Accuracy," *Trans. GIS,* vol. 9, no. 3, pp. 323-344, 2005.
[33] A.D. Sarma, O. Benjelloun, A. Halevy, and J. Widom, "Working Models for Uncertain Data," *Proc. Int'l Conf. Data Eng. (ICDE),* 2005.
[34] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S.E. Hambrusch, "Indexing Uncertain Categorical Data," *Proc. Int'l Conf. Data Eng. (ICDE),* 2007.
[35] Y. Tao, R. Cheng, X. Xiao, W.K. Ngai, B. Kao, and S. Prabhakar, "Indexing Multi-Dimensional Uncertain Data with Arbitrary Probability Density Functions," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2005.
[36] Y. Tao, X. Xiao, and R. Cheng, "Range Search on Multidimensional Uncertain Data," *ACM Trans. Database Systems,* vol. 32, no. 3, article no. 15, 2007.
[37] Y. Theodoridis and T.K. Sellis, "A Model for the Prediction of R-Tree Performance," *Proc. Symp. Principles of Database Systems (PODS),* 1996.
[38] D. Zinn, J. Bosch, and M. Gertz, "Modeling and Querying Vague Spatial Objects Using Shapelets," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2007.

Ying Zhang received the BSc and MSc degrees in computer science from Peking University and the PhD degree in computer science from the University of New South Wales. He is a research fellow in the School of Computer Science and Engineering at the University of New South Wales. His research interests include query processing on data stream, uncertain data, and graphs. He is a member of the IEEE.

Xuemin Lin received the BSc degree in applied math from Fudan University in 1984 and the PhD degree in computer science from the University of Queensland in 1992. He is a professor in the School of Computer Science and Engineering at the University of New South Wales (UNSW). He has been the head of the Database Research Group at the UNSW since 2002. Before joining the UNSW, he held various academic positions at the University of Queensland and the University of Western Australia. During 1984-1988, he studied for the PhD degree in applied math at Fudan University. He is currently an associate editor of the *ACM Transactions on Database Systems*. His current research interests include data streams, approximate query processing, spatial data analysis, and graph visualization.

Wenjie Zhang received the BS and MS degrees in computer science from Harbin Institute of Technology, China. She is currently working toward the PhD degree in the School of Computer Science and Engineering, the University of New South Wales, Australia. Her research focuses on uncertain data management and spatiotemporal indexing techniques. She has published papers in conferences and journals including SIGMOD, VLDB, ICDE, the *Very Large Data Bases Journal*, and the *IEEE Transactions on Knowledge and Data Engineering*.



Jianmin Wang received the BSc and PhD degrees in computer science, respectively, from Peking University in 1990 and Tsinghua University in 1995. He is a professor in the School of Software at Tsinghua University. He is the vice dean of the School of Software. His current research interests include unstructured data management, workflow management, content delivery, digital right management, and performance evaluations.

Qianlu Lin received the BS degree in computer science from the University of New South Wales, Australia. She is currently working toward the PhD degree in the School of Computer Science and Engineering, the University of New South Wales, Australia. Her research focuses on high-dimensional indexing.