Indexing Uncertain Data for Supporting Range Queries*

Rui Zhu, Bin Wang, and Guoren Wang

College of Information Science and Engineering, Northeastern University, China
neuruizhu@gmail.com, {binwang,wanggr}@ise.neu.edu.cn

Abstract. Probabilistic range query is a typical and a fundamental problem in probabilistic DBMS. Although the existing solutions provide a good performance, there are some shortages that are needed to be overcomed. In this paper, we firstly propose a novel structure called MRST to approximately capture the probability density function of uncertain object. Through considering the gradient of the probability density function, MRST could provide uncertain object with strong pruning power and consume fewer space cost. Based on characters of MRST, we also design an efficient algorithm to access MRST. We propose a novel index named R-MRST to efficiently support range query on multidimensional uncertain data. Its has a strong pruning power. At the same time, it has a lower cost both in space and dynamic update. Theoretical analysis and extensive experimental results demonstrate the effectiveness of the proposed algorithms.

1 Introduction

Recently, many emerging applications over uncertain data are attracting a wide attention of researchers [3]. The causes of the uncertainty are greatly different in various applications. For example, in a habitat monitoring system, due to the impreciseness of sensing devices [5], the data obtained are often noisy. As another example, in moving objects tracking [6], the location information of objects collected by a GPS system may not be exact due to the delay on data updating. Among a large number of queries, range query is the most fundamental and important operation in managing uncertain data [1].

A probabilistic range query is to find out the objects that appear in the query region with the probability at least θ(the probabilistic threshold of the query). Since such a computation involves the expensive and complex integral [4] [2], the *filter-refinement* is preferable. In the filtering phase, the probabilistic objects, which must(or not) be the query results, are quickly filtered without proceeding the complex integral. For the objects that cannot be filtered, in the refinement phase, the integral has to be done to verify the answers. Thus, the key of optimizing a prob-range query is to provide, as tight as possible, a bound for flittering with a small cost.

Several indexes have been proposed to answer the queries on uncertain data. The key idea is pre-computing the summary [8] of each object's PDF (short for probability density function), augmenting existing index techniques to organize summary, and

* The work is partially supported by the National Basic Research Program of China (973 Program) (No. 2012CB316201,2011CB302200-G), the National Natural Science Foundation of China (Nos. 61322208, 61272178, 61129002), the Doctoral Fund of Ministry of Education of China (No. 20110042110028), and National High Technology Research and Development 863 Program of China (GrantNo.2012AA011004).

F. Li et al. (Eds.): WAIM 2014, LNCS 8485, pp. 72–83, 2014.

then using the summary for filtering.[8]. One of the most popular index named U-tree employs the PCR(short for probabilistically constrained region) technique to summary the PDF of the uncertain object. However, PCR could not provide the uncertain object with a strong pruning/validating power, and the dynamic update cost of U-Tree is high (detailed in Section2).

Another two popular indexes UI-tree [7] and UD-tree [8] employ the partition technique to summary the PDF of uncertain object. Using the partition technique, the summary of an object could provide it with a stronger pruning/validating ability than the PCR-based [4] summary. However, it still has room for improving. The partition does not fully consider the gradient of PDF. And they both consume too much space cost (e.g., given a 62K data, the index size is 20M).

Contributions: In this paper we study the problem of answering prob-range queries on uncertain data. The contributions are as follows:

Firstly, we propose a novel summary called *MRST*(multi-resolution summary tree) to approximately capture the PDF of uncertain object. The MRST fully considers the gradient of PDF and more effectively captures an object's PDF. It has a more powerful filtering ability and consumes lower space cost. We propose a novel algorithm to access the MRST. Through using the key idea of greedy algorithm, this algorithm could reduce the computational cost as much as possible.

Secondly, we propose a new index called R-MRST to organize the summary of objects. R-MRST augments the R-tree technique. The filtering ability of R-MRST's node is as strong as that of U-Tree, but it has the lower cost both in space and dynamic update.

The rest of this paper is organized as follows: Section 2 gives related work and the problem definition. Section 3 proposes the MRST. Section 4 proposes R-MRST that is used to effectively indexing uncertain data. Section 5 evaluates the proposed methods with extensive experiments. Section 6 is the conclusion and the future work.

2 Related Work and Problem Definition

In Section 2.1, we review the existing indexing approaches. In Section 2.2 we formally define the problem of probabilistic range query on uncertain data. Table 1 summaries the mathematical notations used in the paper.

2.1 Related Work

In recent years, many effective indexes have been proposed to answer prob-range query on the uncertain data. The PCR-based index named U-Tree (and U-catalog-Tree) is proposed by Tao et al [4]. The problem of U-Tree is that the filter ability of PCR is not strong, and the dynamic update cost is high. Given a set of objects O, U-Tree constructs a group of PCRs for every object, and employs the R-tree technique for organizing them. For simplicity, we introduce U-Tree in the 2-dimension space. As is depicted in Fig 1(a), given an object o and a probability threshold $\theta(0 < \theta < 0.5)$(eg.0.2), o.PCR(θ) is constructed as follows: 2 lines in each dimension are computed. In the horizontal dimension, o has the probability θ to occur on the left(right) side of line $l_1(l_2)$. In the vertical dimension, $l_3(l_4)$ is computed in the similar way with $l_1(l_2)$. o.PCR(0.2) is the

Table 1. The Summary of Notations

Notation	definition
o	probabilistic object
$o.pdf(x)$	probability density function of o
o_r	probability region of o
q_r	the search region of the query
q_p	the probability threshold of the query
θ	probability threshold
$o(i)$	the subregion i of o
$PBD(o,i)$	probability bound difference of o(i)
$app(o,i)$	the likelihood of o falling in $o(i)$
$app(o,q)$	the likelihood of o falling in q_r
$app(q,i)$	the likelihood of o falling in $q_r \cap o(i)$
$lb(o,i)(ub(o,i))$	the maximal(minimal) probability density in $o(i)$
$S(o,i)$	the area of $o(i)$
$ZS(o,i)$	the blank(and zero-pdf) area of $o(i).MBR$
$o(i).MBR$	the MBR bounding $o(i)$
$S(q,i)$	the area of $q_r \cap o(i).MBR$

rectangle bound by these four lines. Given a prob-query q with $q_p \leq \theta$(q_p denotes the threshold of q), o.PCR(0.2) is used for pruning/validating if $q_p \geq \theta$. As is depicted in Fig 1(a), q_1, q_2, q_3 and q_4, we assume that their query threshold are all 0.2. Given q_1, o could be pruned because $o.PCR(0.2)$ does not intersect with q_r(short for the query region). On the other hand, given q_2, o could be validated because q_r completely contains the part of the left, upper, down border of $o.MBR$ and l_3. However, the pruning/validating ability of PCR is not powerful if q_r overlaps with an objects but can not contain d-1 dimension planes of an object in a d-dimension space. For example, o obeys uniform distribution. Obviously, o is the query result of q_3. o is not the query result of q_4. However, they can not prune(or validate) o because no filter pruning can be used to prune/validate them. As another problem, the dynamic update cost of U-tree is high. In Fig 1(b), because every object uses a group of PCR to summary its PDF, the node of U-tree also has to use a group of MBRs for bounding these PCRs. Obviously, the cost of maintaining these boundaries is much higher than that of R-Tree once the dynamic update happens.

Zhang et al proposed UI-Tree(and UD-tree) for indexing uncertain objects [7]. The filtering ability of them are stronger than that of U-Tree. However, the space cost of them are all high. To construct the summary of each object's PDF, the key idea of UI-Tree is partitioning the uncertain region of every object, pre-computing the appearance probability of the partitioned sub-region, and using R-tree technique to organize these sub-regions. Given a prob-range query, UI-tree retrieves the sub-regions that overlap with the query region, finds the corresponding objects, and then computes the lower and upper bounds of $app(o,q)$(short for the appearance probability that o lies in the query region). Specifically, given an object, if a subregion $o(i)$ is contained in q_r, $app(o,i)$ (short for the appearance probability that o lies in $o(i)$) contributes to both lower and upper bound of $app(o,q)$. Similarly, if a subregion $o(j)$ overlaps with q_r, $app(o,j)$

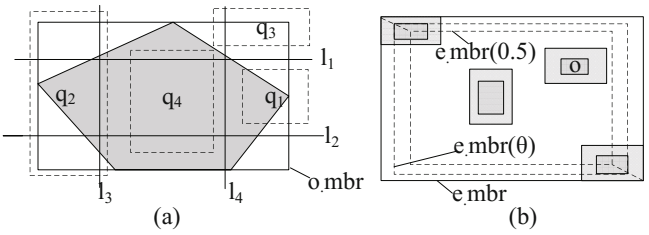


Fig. 1. Answering Prob-range Queries Using U-tree

contributes to the upper bound of $app(o, q)$. Then o may be validated (pruned) based on the lower (upper) bound of $app(o, q)$. Although UI-Tree has the stronger pruning ability than U-Tree, its space cost is too high. In the other hand, the partition do not reflect the PDF's gradient, and the filtering algorithm does not consider the intersection area between the query region and the subregions.

2.2 Problem Definition

Given a multidimensional probabilistic object o in the d-dimension space, it is described either continuously or discretely. In the continuous case, an object has two attributes: o_r and $o.pdf(x)$. The o_r is a d-dimension uncertainty region, where o may appear at any locations with certain probabilities. The $o.pdf(x)$ is the probability of o appearing at location x. In the discrete case, o is represented by a set of sampled points x_1, x_2,. . .,x_m, and o occurs at location x_i with probability $x_i.p$. Given a query region q_r, we use $app(o,q)$ to represent the likelihood of o falling in the query region q_r. $app(o,q)$ is also calculated by two cases. In the continuous case:

$$app(o, q) = \int_{o_r \cap q_r} o.pdf(x)dx \tag{1}$$

where $o_r \cap q_r$ denotes the intersection of o_r and q_r, and o is a result if $p_{app}(o,q) \geq \theta$ (query probability threshold). In the discrete case:

$$app(o, q) = \sum_{i=1}^{n2} o.pdf(x_i) / \sum_{i=1}^{n1} o.pdf(x_i) \tag{2}$$

where n_1 is amount of the sampled points in o_r, and n_2 is the amount of the sampled points falling into $o_r \cap q_r$.

Definition 1. *(***Probabilistic Range Query***). Given a set of probabilistic objects O and a range query q, the probabilistic range query retrieves all probabilistic objects $o \in O$ with $app(o, q) \geq \theta$, where θ is the probabilistic threshold and $0 \leq \theta \leq 1$.*

3 Effectively Summarizing Uncertain Data

In this section, we propose a novel summary called *MRST*(multi-resolution summary tree) to capture the PDF of uncertain data. It provides uncertain data with strong pruning/valiating ability through considering the gradient of PDF. At the same time, *MRST* consumes less space cost than the state of art approaches. In the following part, we discuss how to construct and access MRST respectively. In the last part of this section, we employ the bit-vector technique to both reduce the space cost and computational cost.

3.1 A Tight Probabilistic Bound For Filtering

In this section, we introduce how to provide the object with a tight bound. It is the guide of the summary construction.

We firstly discuss how to provide each sub-region $o(i)$ with a tight bound. Given an object o, a sub-region $o(i)$ and a query q, if q_r overlaps with $o(i).MBR$, Equation 3 and Equation 4 show the probabilistic lower-bound and upper-bound of o lying in $o_r \cap q_r$ respectively. Obviously, by fully considering the intersection area between $o(i).MBR$ and q_r, even if our partition is as the same as that of UD-Tree and UI-Tree, the probabilistic bound proposed in this paper is tighter.

$$lb_{app}(q,i) = lb(o,i) \times (max(0, S(q,i) - ZS(o,i))) \tag{3}$$

$$ub_{app}(q,i) = min(ub(o,i) \times S(q,i), app(o,i)) \tag{4}$$

where $app(o,i)$ represents the likelihood of o falling in $o(i)$. The $lb(o,i)(ub(o,i))$ denotes the maximal(minimal) probability density in $o(i)$. $ZS(o,i)$ represents the blank (and zero-pdf) area of $o(i)$. $lb_{app}(o,i)$ ($ub_{app}(o,i)$) denotes the lower-bound (or upper-bound) of the probability o lying in $q_r \cap o(i)$.

Property 1. Given an object o and a query q, when q_r overlaps with o's subregion $\bigcup_{i=1}^{i=n_1} o(i)$, the $lb_{app}(o,q) = \sum_{i=1}^{i=n_1} lb_{app}(q,i)$ and $ub_{app}(o,q)$ is $\sum_{i=1}^{i=n_1} ub_{app}(q,i)$.

$lb_{app}(o,q)(ub_{app}(o,q))$ denotes the lower-bound(upper-bound) of the probability o lying in q_r. For each object o, $ub_{app}(o,q)$-$lb_{app}(o,q)$ is to evaluate whether the bound is tight enough. According to Equation 3, Equation 4 and Property 1, the following conditions should be satisfied for the tighter bound: (i) $ub(q,i) - lb(q,i)$ should be relatively small; (ii) the amount of subregions should be relatively small.

3.2 Effective Summary Construction Using Multi-Resolution Technique

In this section, we employ the multi-resolution technique to construct the summary (called MRST). The MRST could provide the uncertain object with a more effective partition and a tighter probabilistic bound. Now, we formally define the PBD(short for probability bound difference) which is used as the criterion of construction.

Definition 2. *(*PBD*). Given a sub-region $o(i)$ of an object o, $PBD(o,i)$ = $(ub(o,i) - lb(o,i)) \times S(o,i)$.*

Given an object o, its corresponding MRST is constructed in the following two steps: they are *spilt* and *shrink*. The split is to partition the subregions where the probability density changes dramatically. The procedure is that we recursively partition the object region o_r until the PBD of each sub-region is less than λ. And then, we use a quad-tree to temporarily organize this split result. After spilt, the probability density in each subregion $o(i)$ changes smoothly, and $ub(o,i) - lb(o,i)$ may be small enough. Obviously, the bound provided by MRST is tighter. For example, in Fig 2(a), the shadow region is the object region o_r bounded by a MBR, and the blank region may be seem as the subregion of o_r with a zero-pdf. Fig 2(g) is designed to show the PBD of each subregion. According to Fig 2(g), because the $PBD(o,A)$ and $PBD(o,C)$ are less than λ(=0.1 in this section), we stop splitting them. Because $PBD(o,B)$ and $PBD(o,D)$ are more than λ, we subdivide them into four parts respectively. The Fig 2(b) is the result of spilt, and Fig 2(c) shows the corresponding quad-tree.

After the spilt, the shrink is done to merge the subregions where the probability density of them are roughly the same. Given two subregions $o(i)$, $o(j)$ of o, they are merged if PBD(o,$i+j$)$\leq \lambda$. We access the quad-tree in the post-order. We firstly merge the leaf nodes within the same subtree. Then, we merge the leaf nodes among different subtrees. Specifically, in each subtree, the leaf node with the minimal $app(i,o)$ is selected as the candidate node(eg,. d_1, b_1, A and C). Given two candidate node u and v from different nodes, if PBD(o,$u+v$)$\leq \lambda$, they are merged. According to Fig 2(g), b_1, because PBD(o,b_1+b_2+b_4) $\leq \lambda$,b_1, b_2 and b_4 can be merged. The Fig 2(d) shows the result of merging the nodes from the same subtree, where b_1 and C are merged. The Fig 2(f) is the finally $MRST$.

After constructing the MRST of an object, an interesting result is that if the probability density of a sub-region is dramatically changing, it has a fine partition; otherwise, it has a coarse partition. By this property, it guarantees that the MRST could more effectively reflect the gradients of the PDF, and the amount of subregions is relatively small(shown in experiment). In addition, because the $PBD(o,i)$ of each subregion $o(i)$ is also relatively small, MRST could provide the object with a tight bound. We could build a cost model to find the optimal λ that need to consider both the filtering ability and I/O cost. A similar method was proposed in [8]. Due to the limitation of space, we do not discuss it.

3.3 Accessing the Summary of Uncertain Data

In this section, we propose Algorithm 1 to efficiently access the summary of uncertain data. Algorithm 1 employs the key idea of greedy algorithm. The Algorithm 1 uses a field called $d(q,i)$ to determine the accessing order of the nodes in MRST so as to early terminating the accessing of MRST as much as possible.

$$d(i,q) = min(u(i,o) \times S(q,i), app(o,i)) - lb(i,o) \times (max(0, S(q,i) - ZS(o,i))) \tag{5}$$

Given a query q, an object o and a subregion $o(i)$, if $q.r$ overlaps with $o.MBR$, we access the MRST of o to check whether o is a result of q. The $d(q,i)$ is computed through Equation 5. Obviously, the larger the $d(q,i)$ is, the greater it contributes

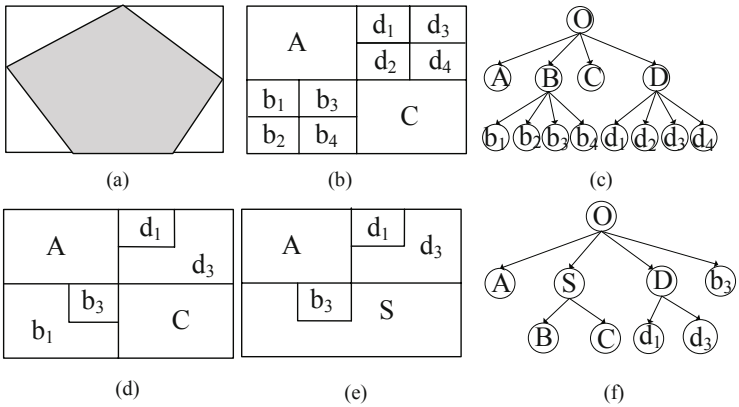


Fig. 2. Constructing MRST

to $ub_{app}(o)$-$lb_{app}(o)$, and the corresponding $o(i)$ should be prior accessed. Compared with the traditional accessing method such as preorder traversal and inorder traversal, introducing this field to control the nodes accessing order is more efficiently to compute the bound.

As shown in Algorithm 1, we firstly access the root of MRST, compute $lb_{app}(o)$ and $ub_{app}(o)$ according to Equation 3 and Equation 4. If o can not be pruned(or validated), we initialize the array L(line 2-6). After initializing L, the following things are repeatedly done to compute the probabilistic bound. Firstly, we find the node e whose corresponding $d(i, q)$ is maximal in L. Secondly, based on e, we tighten the bound: (i) eliminate the contribution of e_i to $ub_{app}(o)$ and $lb_{app}(o)$ (line 8 to 10); (ii) access every children of e_i to compute the new bound according to by Equation 3 and Equation 4,and property 1. Thirdly, if o is not still filtered, we update L: we insert the children e_{ij} of e_i into L, when e_{ij} satisfies the conditions that (i) e_{ij} also has children; (ii) the corresponding subregion of e_{ij} overlaps with q_r.

After accessing the MRST of an object, o is validated if the lower-bound of $app(o, q)$ is more than q_p. Also, o is pruned if the upper-bound of $app(o, q)$ is less than q_p. If o can not be pruned/validated, we have to use the integral to check whether o is the result.

3.4 Efficient Summary Storage

Now, we discuss how to efficiently store the MRST. The MRST stores three types of information to capture the PDF of a given object. Given an object o and a subregion $o(i)$, they are the probabilistic information (eg,.$app(o, i)$, $lb(o, i)$ and $ub(o, i)$), location information, blank area information, and the hierarchical relationship between parent and its children. Because too many information has to be stored, we employ the bit vector to compress MRST as much as possible.

Firstly, we use a m-bits vector to represent the probabilistic information, and its domain is 2^m. As the tradeoff between the degree of accuracy and the space, given an object o and a sub-region $o(i)$, we use 6 bits to express $app(o, i)$, where the domain is 0 to 63. $app(o, i) = 0.2$, it is expressed by $\lfloor 0.2 \times 63 \rfloor$ =12(001100). We use 4bit to

express $lb(o,i)$(also $ub(o,i)$), where the domain is 0 to 15. Secondly, we use a n-bits bit vector to express $o(i)$'s location information.

Specifically, given an object o, we use a MBR to bound it. Next, we could use a "virtual grid" with a $2^n \times 2^n$ resolution to partition the MBR. Lastly, the "virtual coordinate" expressed by bit vector is used to express $o(i)$'s location information. For example, using a 7-bits vector, the resolution of the grid is 128×128. The left-bottom(right-upper) coordinates are described by the cell Id. As shown in Fig 2, the "virtual coordinate" of node d_1 is expressed by (64,111) and (80,127). The area information depends on the resolution of the "virtual grid".

Algorithm 1. Accessing MRST

Input: MRST, o, probabilistic range query, q, Node e
; **Output**: lower-bound, lb; upper-bound, ub
; $ub_{app}(o) \leftarrow min(1, ub(o) \times S(o))$;
$lb_{app}(o) \leftarrow max(0, S(o) - ZS(o)) \times lb(o)$;
if $ub_{app}(o) < p_q \vee lb_{app}(o) \geq p_q$ **then**
    return;
Insert(L,o, d(q,o),$ub_{app}(o)$, $lb_{app}(o)$);
while *Empty(L)≠ true* **do**
    Node e=PopFront(L);
    $ub_{app}(o) \leftarrow ub_{app}(o) - e.ub_{app}$*(o,i)*;
    $lb_{app}(o) \leftarrow lb_{app}(o) - e.lb_{app}$*(o,i)*;
    for i *from* 0 *to* $e.Len$ **do**
        $ub_{app}(o) \leftarrow ub_{app}(o) + min(1, ub(o,i) \times S(o,q))$;
        $lb_{app}(o) \leftarrow lb_{app}(o) + max(0, S(o,1) - ZS(o,i)) \times lb(o,i)$;
    if $ub_{app}(o) < p_q \vee lb_{app}(o) \geq p_q$ **then**
        return;
    else
        for i *from* 0 *to* $R.Len$ **do**
            if $q_r \cap o(i).r \neq \emptyset \wedge q_r \cap o(i) \neq o(i).r$ **then**
                Insert(L,R(i,o), d(q,i),$ub_{app}(o,i)$, $lb_{app}(o,i)$);
return ;

For example, as shown in Fig 2, base on the "virtual grid", because the area of d_1 is 32×32=1024 and half on d_1 is blank, the blank area of d_1 is 512(10000000). Finally, we use a static array to organize the nodes in MRST. We use k-bits vector to express "offset+len" so as to describe the hierarchical relationship between the parent and its children. As shown in Fig 2, D is a interval node that has two children d_1 and d_3, where the offset is 3(11),and len=2(10).

Another advantage of data compression is that we could use the bit-operations to do the above operations shown in algorithm 1. Due to the limitation of space, we do not discuss how to store the node in MRST, and how to access MRST using bit-operations.

4 Indexing uncertain data

In this section, we propose an index called *R-MRST* to organize the MRST of uncertain data. Its pruning ability is roughly the same with the other indexes such as U-Tree, but cost of dynamic update and space are much lower than them.

As is discussed in Section 2.1, it is unworthy to store too much probabilistic information in each node(leaf or interval). For example, given a leaf node based on U-Tree, although using a group of MBRs to bound its children's PCR could obtain a tighter boundary, as shown in Fig 1(b), the shrunken degree of the boundary is relatively small, and it causes both a high space cost and high dynamic update cost. The other indexes such as UI-Tree and UD-Tree also have the similar problem.

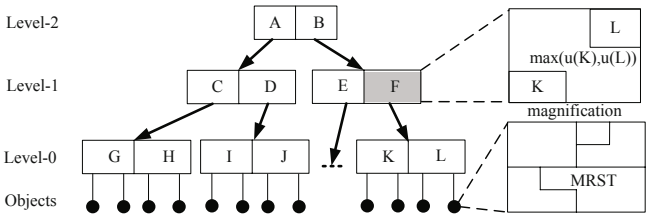


Fig. 3. The Framework of R-MRST

Based on the above analysis, we propose the *R-MRST*. As shown in Fig 3, it is the framework of *R-MRST*. It is similar with R-Tree. Due to the limitation space, we mainly discuss how to maintain the probabilistic information in each node of R-MRST, and how to use it for pruning according to Property 2.

Property 2. Given a prob-range query q and a node e of R-MRST, the intersection area between $e.MBR$ and q_r is S. If $S \times ub(e) < q_p$, e can be pruned.

For each node e in R-MRST, we maintain the maximal probability density called $ub(e)$ among all the objects in the subtree of e. Given a query q, if q_r overlaps with the MBR of e, we employ Property 2 for pruning. Although the pruning method seems simple, as shown in Fig 1, it also could prune the node whose MBR's margin overlaps with the query region. Thus, it is suitable for processing range query over uncertain data, and both the space cost and update cost are low.

Query on R-MRST: Given a prob-range query q, the search starts from the root of R-MRST, and eliminates its entries according to Property 2. For each remaining entry, we retrieve its child node, and perform the above process recursively until a leaf node is reached. For an object o encountered, we attempt to filter it through accessing its MRST. For the object o which can not be filtered, in the refinement phase, we use the integral to check whether o is the result of q.

Dynamic Update Algorithm: Compared with R-tree, the update method of our index is roughly the same. The difference is the maintenance of $ub(e)$. Specifically, given a leaf node e, when a newly arrived object o inserts into e, the following cases cause the updating of $ub(e)$. (i) $ub(o) \geq ub(e)$; (ii) the number of objects in e exceeds to the bucket size, and causes e spilt. In the first case, we set $ub(e)$=$ub(o)$. In the second case,

if e is split into e_1 and e_2, we compute $ub(e_1)$ and $ub(e_2)$. When a object o leaves e, the following cases cause the $ub(e)$ updating. (i) $ub(o)$ is maximal probability density among all the object in e, in this case, we select the new $ub(e)$ from e. (ii) if two node e_1 and e_2 are merged into e, the $ub(e)$ is max($ub(e_1)$, $ub(e_2)$). After updating $ub(e)$, we access the parent $e^{'}$ of e to check whether the $ub(e^{'})$ need to be updated. If so, we update $ub(e^{'})$ and continuous access the upper-level node until no interval node should be updated.

5 Experimental Evaluation

This section experimentally evaluates the efficiency of the proposed techniques. The R-MRST will be compared with U-Tree (a classic technique) and UD-Tree, where U-Tree is a classic index and UD-Tree is the most advanced index technology presently.

Two real spatial data sets LB and CA are employed to represent the center of probabilistic regions, which has been used as the test data set such as [8] [4]. They contain 53k and 62k two-dimension points representing locations in Long Beach and Los Angeles respectively. In addition, three synthetic data sets containing 128k/256k/512k two-dimension points are employed. In our experiments, the region of probabilistic data is a rectangular with side-length varying from 100 to 500 and the default value of the side-length is 200. In this paper, we call the half of side-length as radius. Because it is unfair to select uniform distribution as the $o.pdf(x)$, we use two other common distributions: poisson distribution and normal distribution. In the default case, all dimensions are normalized to domain [0,10000] and LB with constrained normal distribution is employed as the default data set. A workload contains 100 queries in our experiment. The region of the queries are a rectangular with r_q varying from 500 to 1500. In our experiments, we randomly choose the probabilistic threshold $\theta \in (0,1]$ for each query. R-MRST was implemented in C++. Experiments are run on a PC with i3-core and 4 GB memory.

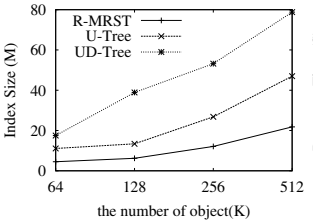
5.1 Index Construction

Firstly, we compare the index size among R-MRST, U-Tree and UD-Tree. Secondly, we compare the constructing time among these three indexes. Thirdly, we compare the space cost of summary based on these three indexes. The experiment is employed in different data sets. One of them is based on the CA. Another one is a synthetic data set with 200k 2-dimension data. Since UD-Tree can not work when the PDF is in the continual case, we use the sampled points to simulate the PDF of an object.

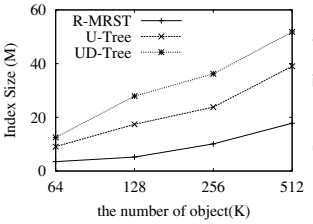
The Fig. 4(a) to Fig. 4(b) uses the synthetic data set. In the Fig. 4(a), the storage cost of R-MRST is less than that of both U-Tree and UD-Tree. Fig. 4(b) shows the space cost of MRST. As we see, ours performs best of all.

5.2 Query Performance

In this section, we evaluate the query performance. In the first group of experiments, we evaluate the performance of the R-MRST, UD-Tree [8] and U-Tree against different r_q.

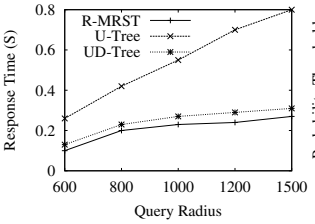


(a) Index Size

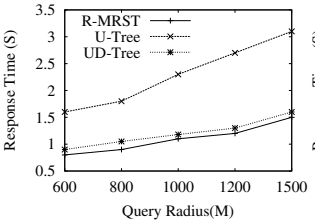


(b) Summary Size

Fig. 4. Index size Based on Different Data Sets

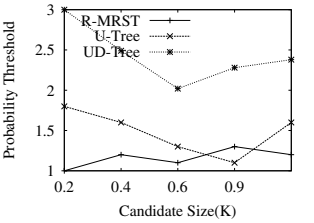


(a) Candidate Size

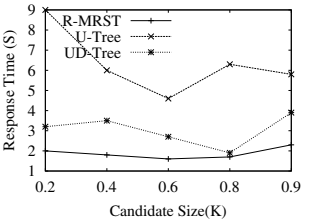


(b) CPU Time

Fig. 5. Cost vs.diff R_u



(a) Candidate Size



(b) CPU Time

Fig. 6. Cost vs. diff θ

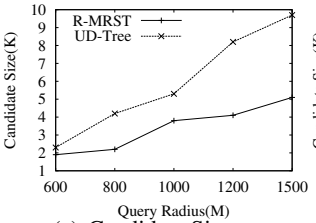
The parameters of the experiments are same as the previous one. Firstly, we evaluate the ability of pruning/validating. In the Fig. 5(a), the candidate size of UD-Tree and R-MRST are roughly the same which both perform better than U-Tree. Secondly, we evaluate the response time. In the Fig. 5(b), R-MRST performs best of all.

In the second group of experiments, we evaluate the performance of the R-MRST, UD-Tree and U-Tree against different threshold θ. The θ varies from 0.1 to 0.9, and the other parameters are default. The experiment content are the same as the first group. The Fig. 6(a)-Fig. 6(a) are the results of the experiments. In the Fig. 6(a), R-MRST performs best of all.

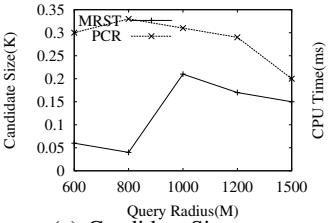
The third group of experiments evaluate the filtering ability and the computational cost of the node in R-MRST. All of parameters are default. Firstly, we count the number of the entry nodes needed to be accessed. In Fig. 7(a), the filtered ability of R-MRST and U-Tree are are roughly the same. Secondly, we record the response time. In the Fig. 7(a), the computational cost of them are roughly the same.

In the forth group of experiments, we study the probability filtering ability of MRST. We count the amount of probabilistic data that should be checked and then calculate the recall $ratio(rr)$ and the response time. The result are reported in the Fig. 8(a)-Fig. 8(a). As expected, MRST has a stronger filtering ability. In the Fig. 8(a), although the computational cost based on MRST is higher than of PCR, the difference of their response time can be accepted.

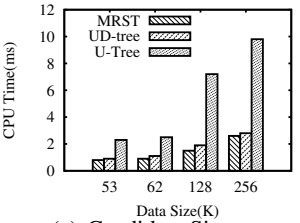
In the last experiments, we compare the performance of R-MRST, UD-Tree, and U-Tree by different data sets. Five data sets (LB,CA and three synthesize) are employed. The number of data points of each data set is 53k, 62k, 128k, 256k and 512k. We use default parameters in these experiments. As expected, R-MRST performs best of all.



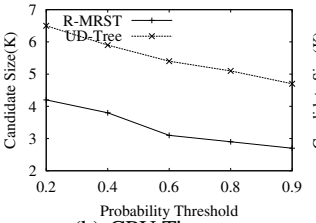
(a) Candidate Size



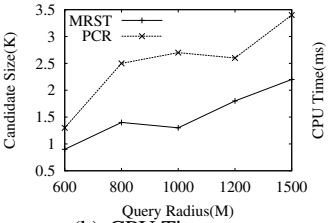
(a) Candidate Size



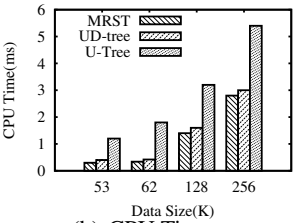
(a) Candidate Size



(b) CPU Time



(b) CPU Time



(b) CPU Time

Fig. 7. S-node vs. U-Tree

Fig. 8. PCR vs.MRST

Fig. 9. Cost vs. diff dataset

6 Conclusions

In this paper, we studied the problem of range query on probabilistic data. Through deep analysis, we proposed an effective indexing technique named R-MRST to manage uncertain data. R-MRST could provided a very tight bound for pruning/validating the objects that overlap(or non-overlap) with the query region in a lower cost. Our experiments convincingly demonstrated the efficiency of our indexing techniques. In the future, we will further study other indexes which are suitable for high-dimensional uncertain data and support probabilistic data update frequently.

References

1. Agarwal, P.K., Cheng, S.W., Tao, Y., Yi, K.: Indexing uncertain data. In: PODS, pp. 137–146 (2009)
2. Kalashnikov, D.V., Ma, Y., Mehrotra, S., Hariharan, R.: Index for fast retrieval of uncertain spatial point data. In: GIS, pp. 195–202 (2006)
3. Lian, X., Chen, L.: Set similarity join on probabilistic data. PVLDB 3(1), 650–659 (2010)
4. Tao, Y., Cheng, R., Xiao, X., Ngai, W.K., Kao, B., Prabhakar, S.: Indexing multi-dimensional uncertain data with arbitrary probability density functions. In: VLDB, pp. 922–933 (2005)
5. Tran, T.T.L., Sutton, C.A., Cocci, R., Nie, Y., Diao, Y., Shenoy, P.J.: Probabilistic inference over rfid streams in mobile environments. In: ICDE, pp. 1096–1107 (2009)
6. Zhang, M., Chen, S., Jensen, C.S., Ooi, B.C., Zhang, Z.: Effectively indexing uncertain moving objects for predictive queries. In: PVLDB, vol. 2(1), pp. 1198–1209 (2009)
7. Zhang, Y., Lin, X., Zhang, W., Wang, J., Lin, Q.: Effectively indexing the uncertain space. IEEE Trans. Knowl. Data Eng. 22(9), 1247–1261 (2010)
8. Zhang, Y., Zhang, W., Lin, Q., Lin, X.: Effectively indexing the multi-dimensional uncertain objects for range searching. In: EDBT, pp. 504–515 (2012)