Efficient Evaluation of Probabilistic Advanced Spatial Queries on Existentially Uncertain Data

Man Lung Yiu, Nikos Mamoulis, Xiangyuan Dai, Yufei Tao, Michail Vaitis

Abstract—We study the problem of answering spatial queries in databases where objects exist with some uncertainty and they are associated with an *existential probability*. The goal of a *thresholding* probabilistic spatial query is to retrieve the objects that qualify the spatial predicates with probability that exceeds a threshold. Accordingly, a *ranking* probabilistic spatial query selects the objects with the highest probabilities to qualify the spatial predicates. We propose adaptations of spatial access methods and search algorithms for probabilistic versions of range queries, nearest neighbors, spatial skylines, and reverse nearest neighbors and conduct an extensive experimental study, which evaluates the effectiveness of proposed solutions.

Index Terms—H.2.4.h Query processing, H.2.4.k Spatial databases

1 Introduction

Conventional spatial databases manage objects located on a thematic map with 100% certainty. In real-life cases, however, there may be uncertainty about the existence of spatial objects or events. As an example, consider a satellite image, where interesting objects (e.g., vessels) have been extracted (e.g., by a human expert or an image segmentation tool). Due to low image resolution and/or color definitions, the data extractor may not be 100% certain about whether a pixel formation corresponds to an actual object o; a probability E_o could be assigned to o, reflecting the confidence of o's existence. We call such objects *existentially uncertain*, since their exact locations are known and the uncertainty refers only to their existence. As another example of existentially uncertain data, consider emergency calls to a police calling center, which are dialed from various map locations. Depending on the caller's voice, for each call we can generate a spatial event associated with a potential emergency and a probability that the emergency is actual. Events generated from sensors (e.g., smoke detection) can also be regarded as existentially uncertain data because each sensor is associated with a certain location and the existence of each detected event depends on the sensor sensitivity and the background noise. Existential probabilities are also a natural way to model *fuzzy classification* [1]. In this case, the class label of a particular object is uncertain; each label has an existential probability and the sum of all probabilities is 1.

- *M. L. Yiu is with the Department of Computer Science, Aalborg University, DK-9220 Aalborg, Denmark.*
  E-mail: mly@cs.aau.dk
- *N. Mamoulis and X. Dai are with the Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong.*
  E-mail: {nikos,xydai}@cs.hku.hk
- *Y. Tao is with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Sha Tin, New Territories, Hong Kong.*
  E-mail: taoyf@cse.cuhk.edu.hk
- *M. Vaitis is with the Department of Geography, University of the Aegean, Mytilene, Greece.*
  E-mail: vaitis@aegean.gr

Manuscript received XXXX XX, 200X; revised XXXX XX, 200X.

We can naturally define probabilistic versions of spatial queries that apply on collections of existentially uncertain objects. We identify two types of such probabilistic spatial queries. Given a *confidence* threshold t, a *thresholding* query returns the objects (or object pairs, in case of a join), which qualify some spatial predicates with probability at least t. E.g., given a segmented satellite image with uncertain objects, consider a port officer who wishes to find a set of vessels S such that every $o \in S$ is the nearest ship to the port with confidence at least 30%. Another example is a police station asking for the emergencies in its vicinity, which have high confidence. A *ranking* spatial query returns the objects, which qualify the spatial predicates of the query, in order of their confidence. Ranking queries can also be thresholded (in analogy to nearest neighbor queries) by a parameter m. For instance, the port officer may want to retrieve the $m = 10$ ships with the highest probability to be the nearest neighbor of the port.

Previous work on managing spatial data with uncertainty [2], [3], [4], [5], [6], [7] focus on *locationally* uncertain objects; i.e., objects which are known to exist, but their (uncertain) location is described by a probability density function. The rationale is that the managed objects are actual moving objects with unknown exact locations due to GPS errors or transmission delays. In Section 2, we elaborate the fundamental differences (e.g., location, existence, storage, and probability computation) between existentially uncertain objects and locationally uncertain objects, and explain why existing solutions for managing and searching locationally uncertain data are inappropriate for existentially uncertain data. To our knowledge, there is no prior work on existentially uncertain spatial data. Our contributions are summarized as follows:

- We identify the class of existentially uncertain spatial data and define two intuitive probabilistic query types on them; *thresholding* and *ranking* queries.
- Assuming that the spatial attributes of the objects are indexed by 2-dimensional R–trees, we propose search

algorithms for probabilistic variants of spatial range queries, nearest neighbor (NN) search, spatial skyline (SS) queries, and reverse nearest neighbor (RNN) queries.

- Regarding different variants of R–trees, we derive appropriate lower/upper probabilistic bounds for effectively reducing the search I/O cost. Our search algorithms for NN, SS, RNN are carefully designed to handle disqualified entries in such a way that their removal is guaranteed not to influence the probabilistic bounds of any potential result object.

The rest of the paper is organized as follows. Section 2 provides background on querying spatial objects with uncertain locations and extents. Section 3 defines existentially uncertain data and query types on them. In Section 4 we study the evaluation of probabilistic spatial queries, when they are primarily indexed on their spatial attributes, or when considering existential probability as an additional dimension. Section 5 addresses probabilistic variants for interesting advanced spatial queries. Section 6 is a comprehensive experimental study for the performance of the proposed methods. Section 7 discusses the case where the existential probabilities of objects are correlated. Finally, Section 8 concludes the paper with a discussion about future work.

2 Background and Related Work

In this section, we review popular spatial query types and show how they can be processed when the spatial objects are indexed by R–trees. In addition, we provide related work on modeling and querying spatial objects of uncertain location and/or extent.

2.1 Spatial Query Processing

The most popular spatial access method is the R–tree [8], which indexes minimum bounding rectangles (MBRs) of objects. R–trees can efficiently process main spatial query types, including spatial range queries, nearest neighbor queries, and spatial joins. Figure 1 shows a collection $R = \{p_1, \ldots, p_8\}$ of spatial objects (e.g., points) and an R–tree structure that indexes them. Given a spatial region W, a *spatial range query* retrieves from R the objects that intersect W. For instance, consider a range query that asks for all objects within distance 3 from q, corresponding to the shaded area in Figure 1. Starting from the root of the tree, the query is processed by recursively following entries, having MBRs that intersect the query region.

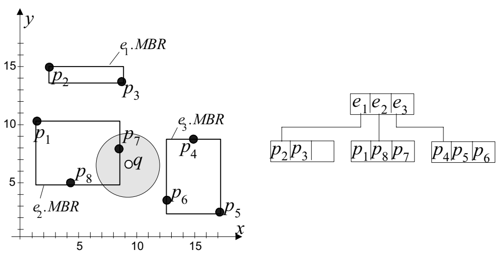


Fig. 1. Spatial queries on R–trees

A nearest neighbor (NN) query takes as input a query object q and returns the closest object in R to q. For instance, the NN of q in Figure 1 is p_7. If R is indexed by an R–tree, then the *best-first* (BF) algorithm of [9] is the most efficient solution for processing NN queries. A *best-first* priority queue PQ, which organizes R–tree entries based on the (minimum) distance of their MBRs to q, is initialized with the root entries. The top entry of the queue e is then retrieved; if e is a leaf node entry, the corresponding object is returned as the NN (assuming point objects). Otherwise, the node pointed by e is accessed and all entries are inserted to PQ. In order to find the NN of q in Figure 1, BF first inserts to PQ entries e_1, e_2, e_3, and their distances to q. Then the nearest entry e_2 is retrieved from PQ and objects p_1, p_7, p_8 are inserted to PQ. The next nearest entry in PQ is p_7, which is the NN of q. In Section 4, we will extend BF for processing probabilistic versions of NN search on existentially uncertain data.

2.2 Locationally Uncertain Spatial Data

Recently, there is an increasing interest on the modeling, indexing, and querying of objects with uncertain location and/or extent. For instance, consider a collection of moving objects, whose positions are tracked by GPS devices. Exact locations are unknown due to GPS errors and transmission delays; e.g., if the object is in motion its location might be outdated when reaching the listening server. As a result, the set of possible locations of an object is captured by a probability density function (PDF), which combines GPS measurement error, the last reported object location, and object velocity [2]. Figure 2a exemplifies a locationally uncertain object o_1, modeled by a 2D Gaussian PDF, with the regions of higher probability marked in darker color. According to [10], [7], an arbitrary PDF can be approximated by a spatial histogram (e.g., 3×3 bins in Figure 2a), where each bin stores the probability to include the object, and their sum equals to 1.

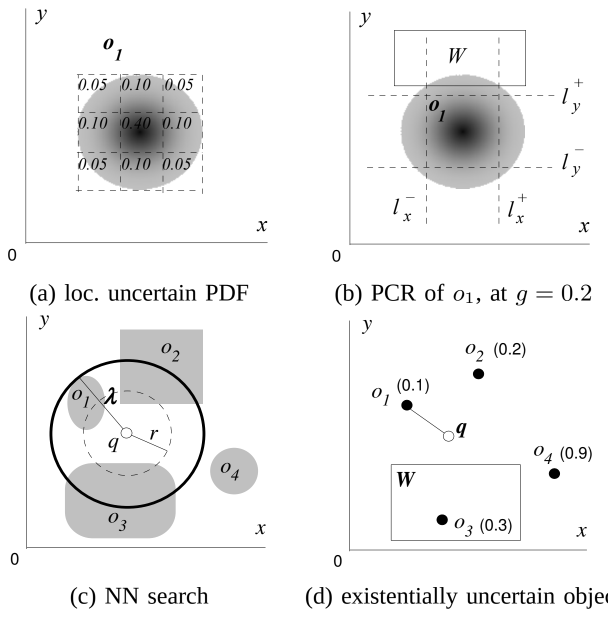


Fig. 2. Locationally and Existentially Uncertainty Objects

TABLE 1
Fundamental Differences between Two Notions of Uncertainty

Property	**Locationally Uncertain Object**	**Existentially Uncertain Object**
Location	uncertain	certain
Existence	certain	uncertain
Storage	a spatial histogram	a point o with existence probability E_o
Probability computation	expensive numerical integration	constant time constant time

Given a locationally uncertain object o and a query range W (see Figure 2b), the probability that o intersects a query range W is formally defined by: $P_{rng}(o,W) = \int_{o \cap W} o.pdf(v)\ dv$ where $o.pdf(v)$ denotes the probability that o coincides with point v. Probabilistic threshold range queries [10], [7] retrieve result pairs $\langle o, P_{rng}(o,W) \rangle$ such that $P_{rng}(o,W) \geq t$, where t is a user-specified threshold. The filter-refinement framework is adopted to accelerate their evaluation. An inexpensive *filter step* is applied to determine fast whether an object o can belong to the result. Only when o may potentially become a result, the *refinement* step is executed to compute the $P_{rng}(o,W)$ value. In the state of the art method of [7] *probabilistic constrained rectangle* (PCR) is used for the filter step of the queries. Given a system parameter g, modeling a minimum value for t, the PCR of a 2D object o is pre-computed by sliding each axis-parallel line inwards until the swept area over the PDF of o equals to g. Figure 2b illustrates the PCR of an object o_1, for $g = 0.2$; o_1 appears in the region on the left of line l_x^- with probability 0.2. Similarly, o_1 appears in regions on the right/bottom/top of lines $l_x^+/l_y^-/l_y^+$ respectively, with probability 0.2. To answer the threshold range query W (with $t = 0.5$), we first compare W with the lines $l_x^-/l_x^+/l_y^-/l_y^+$. Since W does not intersect the PCR of o_1 (i.e., it is above line l_y^+), we can immediately infer that $P_{rng}(o_1,W) \leq 0.2 < t$. Thus, o_1 is discarded during the filter step of query W, saving the expensive computation of the exact probability $P_{rng}(o_1,W)$.

Table 1 summarizes the fundamental differences between locationally uncertain objects and existentially uncertain objects. As depicted in Figure 2d, an existentially uncertain object o_3 has a certain location (i.e., a point) but its existence is associated with a probability E_{o_3}=0.3. The probability of o_3 satisfying a range query W is $P_{rng}(o_3,W) = E_{o_3}$ if o_1 intersects W; or 0 otherwise. Thus, $P_{rng}(o_3,W)$ can be computed in constant time.

One may argue that an existentially uncertain point o with existence probability E_o could be modeled as a locationally uncertain object with the PDF consisting of exactly 2 locations: one point o with probability E_o, and a point at infinity with probability $(1 - E_o)$. This model encumbers the application of existing locationally uncertain techniques [10], [7], because they assume multiple locations with probabilities and the continuity of PDF in the space. Consider, for instance, the probabilistic NN search algorithm for locationally uncertain data, proposed in [6]. Given a query point q and a set R of locationally uncertain objects, we can derive $\lambda = \min_{o \in R} maxd(q,o)$, i.e., the minimum furthest distance of any o from q. For instance, in Figure 2c, the object o_1 leads to the minimum λ. Since the PDF of o_1 sums to 1 within the circle (centered at q with radius λ), it is clear that, any object o'' (e.g., o_4) with $mind(q,o'') > \lambda$ has no chance of being the NN of q. For any remaining object o (e.g., o_1, o_2, o_3), its probability of being the NN of q is denoted by $P_{nn}(o,q)$. Assuming independent PDFs between different objects, [6] define $P_{nn}(o,q)$ as follows:

$$P_{nn}(o,q) = \int_{mind(q,o)}^{\lambda} o.pdf(\odot_q(r)) \cdot \prod_{o' \in R, o' \neq o, mind(q,o') \leq \lambda} (1 - o'.pdf(\otimes_q(r)))\ dr$$

where $\odot_q(r)$ and $\otimes_q(r)$ represent the hollow ring and the concrete circle respectively, centered at q with radius r. The evaluation of the above probability is expensive for arbitrary PDFs so [6] focuses on basic PDFs and develops efficient computation techniques for $P_{nn}(o,q)$. Note that the above probabilistic NN search technique is inapplicable to existentially uncertain data. Figure 2d depicts a set of existentially uncertain objects, with a similar spatial configuration as in Figure 2c. In this case, o_1 is still the object causing the minimum λ value. However, since its existence probability is not 1, it cannot be used to bound the search space. For instance, the object o_4 now has non-zero probability of being the NN of q; this happens with the probability $(1-0.1)(1-0.2)(1-0.3)0.9 = 0.454$, when o_4 exists but o_1, o_2, o_3 do not exist.

Other work on locationally uncertain data includes indexing the trajectory of an object as a cylindrical volume around the tracked polyline (e.g., by a GPS), capturing uncertainty up to a certain distance from the polyline [11]. A similar approach is followed in [3], where recorded trajectories are converted to sequences of locations connected by elliptical volumes. [5] also models the uncertain locations of spatial objects by (circular) uncertainty regions and discuss how to process simple and aggregate spatial range queries using the fuzzy representations. [4] studies the evaluation of spatial joins between two sets of objects, for the case where the object extents are ‘floating’ according to uncertainty distance bounds. An extension of the R–tree that captures uncertainty in directory node entries is proposed, and R–tree join techniques are adapted to process the join efficiently. Cheng et al. [12], [10] study a problem related to probabilistic spatial range queries. The uncertain data are not spatial, but ordinal 1D values (e.g., temperature values recorded from sensors). [10] indexes such uncertain data for efficient evaluation of probabilistic range queries. [12] classifies queries on such data to *entity-based* queries asking for the set of objects satisfying a query predicate and *value-based* queries asking for a PDF describing the distribution of a query result when it is a single aggregate value (e.g., the sum of values, the maximum value, etc.). Finally, [13] studies the evaluation of queries over uncertain or summarized data, where the user specifies thresholds (precision, recall, laxity) regarding the quality (i.e., accuracy) of the desired result.

3 Existentially Uncertain Spatial Data

An object x is *existentially* uncertain if its existence is described by a probability E_x, $0 < E_x \leq 1$. We refer to E_x as *existential probability* or *confidence* of x. Note that since we

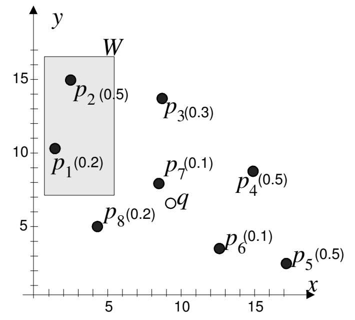


Fig. 3. NN search example

can have $E_x = 1$, we (trivially) regard a 100% known object x as existentially uncertain. This allows us to model object collections which are mixtures of uncertain and certain data. On the other hand, $E_x = 0$ corresponds to an object x that definitely does not exist, so there is no need to store it in a database. We take the *existential independence assumption* that the confidence values of two different objects are independent of each other. This assumption is reasonable for the applications mentioned in the Introduction (e.g., satellite image extraction, emergency call). We will relax this assumption in Section 7 and handle existentially uncertain objects whose confidence values are correlated.

Figure 3 shows a collection $R = \{p_1, p_2, \ldots, p_8\}$ of existentially uncertain points. Next to each point label p_i, is its existential probability E_{p_i} enclosed in parentheses (e.g., $E_{p_1} = 0.2$). We are interested in answering spatial queries that take uncertainty into account. Let R be a collection of existentially uncertain objects. We then define probabilistic versions of basic spatial query types:

Definition 1: A probabilistic spatial range query *takes as input a spatial region W and returns all (x, P_x) pairs, such that $x \in R$ and x intersects W with probability $P_x = E_x$, where $P_x > 0$.*

Definition 2: A probabilistic nearest neighbor query *takes as input an object q and returns all (x, P_x) pairs, such that $x \in R$ and x is the nearest neighbor of q, with probability $P_x = E_x \cdot \prod_{x' \in R, x' \neq x, d(q,x') < d(q,x)} (1 - E_{x'})$, where $P_x > 0$ and $d(q, x)$ denotes the distance between q and x.*

The output of a probabilistic query is a conventional query result coupled with a positive probability that the item satisfies the query. The case of probabilistic range queries is simple; $P_x = E_x$ for each object that qualifies the spatial predicate. Consider, for instance, the shaded window W, shown in Figure 3. Two objects p_1 and p_2 intersect W, with confidences $E_{p_1} = 0.2$ and $E_{p_2} = 0.5$, respectively. Similar to locationally uncertain data, the probability of an object x to qualify a spatial range query is irrelevant of the locations and confidences of other objects.

On the other hand, the probability of an object to be the nearest neighbor depends on the locations and probabilities of other objects. Consider again Figure 3 and assume that we want to find the potential nearest neighbor of q. The nearest point to q (i.e., p_7) is the actual NN iff p_7 exists. Thus, (p_7, E_{p_7}) is a query result. In order for the second nearest point p_6 to be the NN of q (i) p_7 must *not* exist and (ii) p_6 must exist. Thus, $(p_6, (1 - E_{p_7}) \cdot E_{p_6})$ is another result. By continuing this way, we can explore the whole set of points in R and assign a probability to each of them to be the NN of q. This nearest neighbor query example not only shows the search complexity in uncertain data, but also unveils that the result of probabilistic queries may be arbitrarily large. For instance, the result of any NN query is as large as $|R|$, if $E_x < 1$ for all $x \in R$. We can define practical versions of probabilistic queries with controlled output by either *thresholding* the results of low probability to occur or *ranking* them and selecting the most probable ones:

Definition 3: Let (x, P_x) be an output item of a probabilistic spatial query Q. The thresholding *version of Q takes as additional input a threshold t, $0 < t \leq 1$ and returns the results for which $P_x \geq t$. The* ranking *version of Q takes as additional input a positive integer m and returns the m results with the highest P_x.*

For example, a thresholding range (window) query W with $t = 0.6$ on the objects of Figure 3 returns $\varnothing$, whereas a ranking range query W with $m = 1$ returns $(p_2, 0.5)$.

4 Evaluation of Basic Probabilistic Queries

Like spatial queries on exact data, probabilistic spatial queries can be efficiently processed with the use of appropriate access methods. In this section, we explore alternative indexing schemes and propose algorithms for probabilistic queries on them. We focus on the most important spatial query types; namely, range queries and nearest neighbor queries.

4.1 Algorithms for 2D R–trees

The most straightforward way to index a set R of existentially uncertain spatial data is to create a 2-dimensional R–tree on their spatial attribute. The confidences of the spatial objects are stored together with their geometric representation or approximation (for complex objects) at the leaves of the tree. We now study the evaluation of probabilistic queries on top of this indexing scheme.

4.1.1 Range Queries

Probabilistic range queries can be easily processed in two steps; a standard depth-first search algorithm is applied on the R–tree to retrieve the objects that qualify the spatial predicate of the query. For each retrieved object x, $P_x = E_x$. If the query Q is a thresholding query, the threshold t is used to filter out objects with $P_x < t$.[1] If Q is a ranking query, a priority queue maintains the m results with the highest P_x, during search, and outputs them at the end of query processing.

1. Especially for thresholding range queries of very large thresholds t, a viable alternative could be to use a B^+–tree that indexes objects based on their probability to efficiently access the objects x with $E_x \geq t$ and then filter them using the spatial query predicate.

4.1.2 Nearest neighbor search

NN search is more complex compared to range queries, because the probability of an object to qualify the query depends on the locations and confidences of other objects. Algorithm 1 elegantly and efficiently computes the probability P_x of x to be nearest neighbor of q, for all x having $P_x > 0$.

Algorithm 1 Probabilistic NN on a 2D R–tree

```
**Algorithm** PNN2D(Query point $q$, 2D R–tree on $R$)
1: $P^{first}$ := 1;                              ▷ Prob. of no object before $x$
2: **while** $P^{first} > 0$ and more objects in $R$ **do**
3:     $x$ := next NN of $q$ in $R$;
4:     $P_x := P^{first} \cdot E_x$;
5:     **output** ($x$,$P_x$);
6:     $P^{first} := P^{first} \cdot (1 - E_x)$;
```

Algorithm PNN2D applies best-first NN-search [9] on the R–tree to incrementally retrieve the nearest neighbors of q, without considering confidences. It also incrementally maintains a variable P^{first} which captures the probability that no object retrieved before the current object x is the actual NN. P^{first} is equal to $\prod_y (1 - E_y)$, for all objects y seen before x. Thus the probability of x to be the nearest neighbor of q is $P^{first} \cdot E_x$. In the example of Figure 3, PNN2D gradually computes $P_{p_7} = 0.1$, $P_{p_6} = (1 - 0.1) \cdot 0.1 = 0.09$, $P_{p_8} = (1 - 0.1)(1 - 0.1) \cdot 0.2 = 0.162$, $P_{p_4} = (1 - 0.1)(1 - 0.1)(1 - 0.2) \cdot 0.5 = 0.324$, etc. Note that *all* objects of R in this example are retrieved and inserted to the response set. In other words, PNN2D does not terminate, until an object x with $E_x = 1$ is found; if no such object exists, all objects have a positive probability to be the nearest neighbor.

4.1.2.1 Thresholding and ranking: As discussed in Section 3, the user may want to restrict the response set by thresholding or ranking. Algorithm 2 is the thresholding version of PNN2D, which returns only the objects x with $P_x \geq t$. The only differences with the non-thresholding version are the termination condition at Line 2 and the filtering of results having $P_x < t$ (Line 5). As soon as $P^{first} < t$, we know that the next objects, even with 100% confidence cannot be the NN of q, so we can safely terminate. For example, assume that we wish to retrieve the points in Figure 3 which are the NN of q with probability at least $t = 0.23$. First p_7 with $P_{p_7} = E_{p_7} = 0.1$ is retrieved, which is filtered out at Line 5 and P^{first} is set to $0.9 \geq t$. Then we retrieve p_6 with $P_{p_6} = P^{first} \cdot E_{p_6} = 0.09$ (also disqualified) and set $P^{first} = 0.81 \geq t$. Next, p_8 is retrieved with $P_{p_8} = 0.162$ (also disqualified) and $P^{first} = 0.648 \geq t$. The next object p_4 satisfies $P_{p_4} = 0.324 \geq t$, thus $(p_4, 0.324)$ is output. Then $P^{first} = 0.324 \geq t$ and we retrieve p_3 with $P_{p_3} = 0.0972$ (disqualified). Finally, $P^{first} = 0.2268 < t$ and the algorithm terminates having produced only $(p_4, 0.324)$.

PRNN2D (Algorithm 3), the ranking version of PNN2D, maintains a heap H of m objects with the largest P_x found so far. Let P^m be the m-th largest P_x in H; as soon as $P^{first} < P^m$, we know that the next objects, even with 100% confidence cannot be the in the set of m most probable NN of q, so we can safely terminate. For example, assume that we wish to retrieve the point with the highest probability of being the NN of q in Figure 3. PRNN2D progressively

Algorithm 2 Probabilistic NN on a 2D R–tree with thresholding

```
**Algorithm** PTNN2D(Query point $q$, 2D R–tree on $R$, Threshold $t$)
1: $P^{first}$ := 1;                              ▷ Prob. of no object before $x$
2: **while** $P^{first} \geq t$ and more objects in $R$ **do**
3:     $x$ := next NN of $q$ in $R$;
4:     $P_x := P^{first} \cdot E_x$;
5:     **if** $P_x \geq t$ **then**
6:         **output** ($x$,$P_x$);
7:     $P^{first} := P^{first} \cdot (1 - E_x)$;
```

maintains the object with the highest P_x. After each of the first 4 object accesses, P^m becomes 0.1, 0.1, 0.162, and 0.324. The algorithm terminates after the 4-th loop, when $P^{first} = 0.324$ and $P^m = P_{p_4} = 0.324$; this indicates that the next object can have P_x at most P_{p_4}, thus p_4 has the highest chances among all objects to be the NN of q.

Algorithm 3 Probabilistic NN on a 2D R–tree with ranking

```
**Algorithm** PRNN2D(Query point $q$, 2D R–tree on $R$, Integer $m$)
1: $P^{first}$ := 1;                              ▷ Prob. of no object before $x$
2: $H$ := ∅;                              ▷ heap of $m$ objects with highest $P_x$
3: $P^m$ := 0;                                    ▷ $P_x$ of $m$-th object in $H$
4: **while** $P^{first} \geq P^m$ and more objects in $R$ **do**
5:     $x$ := next NN of $q$ in $R$;
6:     $P_x := P^{first} \cdot E_x$;
7:     **if** $P_x \geq P^m$ **then**
8:         update $H$ to include $x$;
9:         $P^m$ := $m$-th probability in $H$;
10:    $P^{first} := P^{first} \cdot (1 - E_x)$;
```

4.2 Query Evaluation using Augmented R–trees

We can enhance the efficiency of the probabilistic search algorithms, by augmenting some statistical information to the R–tree directory node MBRs. A simple and intuitive method is to store with each directory node entry e a value e^{maxE}; the maximum E_x for all objects x indexed under e. This value can be used to prune R–tree nodes, while processing thresholding or ranking queries. Similar augmentation techniques are proposed in [4], [10] for locationally uncertain data.

TABLE 2
Checking disqualified entries in augmented 2D R–trees

query type	*range search*	*NN search*
thresholding	$e^{maxE} < t$	$P^{first} \cdot e^{maxE} < t$
ranking	$e^{maxE} \leq P^m$	$P^{first} \cdot e^{maxE} \leq P^m$

Table 2 summarizes the conditions for pruning R–tree entries (and the corresponding sub-trees) which do not point to any results, during range or NN thresholding and ranking queries. For range queries, we can directly prune an entry e when: (i) e.MBR does not intersect the query range, or (ii) its e^{maxE} satisfies the condition in the table. On the other hand, for NN search, a disqualified entry cannot be directly pruned, because the confidences of objects in the pointed subtree may be needed for computing the probabilities of objects with greater distances to q, but high enough probabilities to be included in the result.

Let us assume for the moment that for each non-leaf entry e we know the exact number of objects e^{num} in its subtree. Algorithm 4 is the thresholding NN procedure for the augmented 2D R–tree. BF is extended as follows: If a non-leaf entry e is de-heaped for which $P^{first} \cdot e^{maxE} < t$, the node where e points is not immediately loaded (as in PTNN2D) but e is inserted into a set L of *deleted* entries. For objects x retrieved later from the Best-First heap, we use entries in L to compute P_x^{min} and P_x^{max}; lower and upper bounds for P_x. If $P_x^{min} \geq t$, we know that x is definitely a result. If $P_x^{max} < t$, we know that x is definitely not a result. On the other hand, if $P_x^{min} < t \leq P_x^{max}$ (Lines 6–12), we must refine the probability range for x. For this purpose, we pick the entry e with the minimum $mind(q, e)$ in L.[2] Observe that any entries with $mind(q, e) > d(q, x)$ cannot contribute to the probability of x. As $P_x^{min} < P_x^{max}$ (at Line 6), the entry e selected at Line 7 must satisfy $mind(q, e) \leq d(q, x)$. If e is an object, then q must be nearer to e than x and we update P^{first} with the confidence of e. Otherwise, its confidence does not affect P^{first}, we access its child node n_e and insert all entries of n_e into L. In either case, the probability range of x shrinks. The process is repeated while the range covers t.

Algorithm 4 Probabilistic NN on an augmented 2D R–tree with thresholding

```
**Algorithm** PTNN2Daug(Query point $q$, Augmented 2D R–tree on $R$, Threshold $t$)
1:  $P^{first}$ := 1;                                   ▷ Prob. of no object before $x$
2:  $L$ := $\varnothing$;                                ▷ list of disqualified entries
3:  **while** $P^{first} \geq t$ and more objects in $R$ **do**
4:      $x$ := next NN of $q$ in $R$;
        ▷ during BF-search, each non-leaf entry with $P^{first} \cdot e^{maxE} < t$ is removed from Best-First heap and inserted into $L$
5:      compute $P_x^{min}$ and $P_x^{max}$ by using $P^{first}$, $L$ and $E_x$;
6:      **while** $P_x^{min} < t \leq P_x^{max}$ **do**
7:          pick the entry $e$ with the smallest $mind(q, e)$ in $L$; remove $e$ from $L$;
8:          **if** $e$ is an object **then**        ▷ $e$ is an object closer to $q$ than $x$ is
9:              $P^{first}$ := $P^{first} \cdot (1 - E_e)$;
10:         **else**
11:             read node $n_e$ pointed by $e$ and insert all entries of $n_e$ into $L$;
12:         compute $P_x^{min}$ and $P_x^{max}$ by using $P^{first}$, $L$ and $E_x$;
13:     **if** $P_x^{min} \geq t$ **then**
14:         **output** ($x$,$P_x^{min}$,$P_x^{max}$);
15:     $P^{first}$ := $P^{first} \cdot (1 - E_x)$;
```

It remains to clarify how P_x^{min} and P_x^{max} for an object x are computed. Note that L only contains entries whose minimum distance to q are smaller than $d(q, x)$. For an entry e in the list L, the confidence of each object in its subtree is in the range $(0, e^{maxE}]$. In addition, there exists at least one object in e whose confidence is exactly e^{maxE}. Thus, P_x^{min} corresponds to the case where for all objects under all entries in L are closer to q than x is and they all have the maximum possible confidences. P_x^{max} corresponds to the case, where for all $e \in L$, with maximum distance from q greater than $d(q, x)$, there is only one object with e^{maxE} confidence (for all other objects under e the confidence converges to 0):

$$P_x^{min} = P^{first} \cdot E_x \cdot \prod_{e \in L \,\wedge\, mind(q,e) \leq d(q,x)} (1 - e^{maxE})^{e^{num}} \quad (1)$$

$$P_x^{max} = P^{first} \cdot E_x \cdot \prod_{e \in L \,\wedge\, maxd(q,e) \leq d(q,x)} (1 - e^{maxE}) \quad (2)$$

So far, we have assumed that for each non-leaf entry e the number of objects e^{num} in its subtree is known (e.g., this information is augmented, or the tree is packed). We can still apply the algorithm for the case where this information is not known, by using an upper bound for e^{num}: $f^{level(e)}$, where $level(e)$ is the level of the entry e (leaves are at level 0) and f is the maximum R–tree node fanout. This upper bound replaces e^{num} in Equation 1.

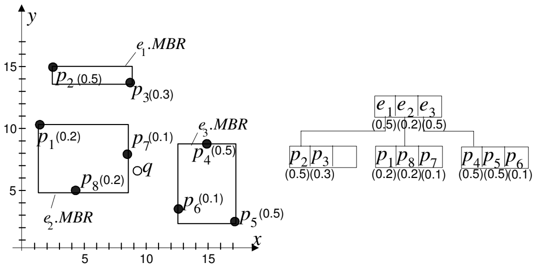


Fig. 4. Example of augmented 2D R–tree

Let us now show the functionality of the PTNN2Daug algorithm by an example. Consider the augmented R–tree of Figure 4 that indexes the pointset of Figure 3 and assume that we want to find the points that are the NN of q with probability at least $t = 0.23$. First, the entries in the root are enheaped in the Best-First heap. Next, the entry e_2 is dequeued. Since it disqualifies the query ($P^{first} \cdot e_2^{maxE} = 0.2 < t$), it is inserted into the list L. Then, the entry e_3 is dequeued. Its objects p_4, p_5, p_6 are enheaped in the Best-First Queue. The nearest object p_6 is dequeued. From Equations 1 and 2, we derive a probability range for P_{p_6} by using P^{first} and L. p_6 is disqualified as $P_{p_6}^{max} = E_{p_6} = 0.1 < t$. Then, $P^{first} = 0.9 \geq t$ and we retrieve p_4. Since $P_{p_4}^{min} = 0.9 \cdot 0.5 \cdot (1 - 0.2)^3 = 0.2304 \geq t$, p_4 is a result. Next, $P^{first} = 0.45 \geq t$ and the next entry retrieved from the priority queue of the BF algorithm is e_1. We do not access the node pointed by e_1, since we know that for each object x indexed under e_1, $P_x \leq e_1^{maxE} \cdot P^{first} = 0.225 < t$. Thus, e_1 is inserted into L. Next, p_5 is dequeued and discarded as $P_{p_5}^{max} = 0.45 \cdot 0.5 \cdot (1 - 0.2) \cdot (1 - 0.5) < t$. Now, the Best-First heap becomes empty and the algorithm terminates. Note that PTNN2D accesses all nodes of the tree in this example, whereas PTNN2Daug saves two leaf node accesses.

Ranking NN retrieval on the augmented R–tree is performed by Algorithm 5. PRNN2Daug has several differences from the thresholding NN algorithm. A heap H is employed to organize objects o by their P_o^{min}. P^m denotes the m-th highest P_o^{min} in the heap. Observe that more complicated techniques are used for updating H, as the accesses to L may affect the order of objects in H. Each object o in H maintains P_o^{first}, which is the value of P^{first} when o is enheaped (Line 18). At Lines 12–

2. Throughout the paper, we use $d(q, x)$ to denote the distance between two points q and x; and use $mind(q, e)$ ($maxd(q, e)$) to denote the minimum (maximum) possible distance between q and any data point indexed by the sub-tree pointed by e.

Algorithm 5 Probabilistic NN on an augmented 2D R–tree with ranking

```
    Algorithm PRNN2Daug(Query point $q$, Augmented 2D R–tree on $R$, Integer $m$)
 1: $P^{first}$ := 1;                                   ▷ Prob. of no object before $x$
 2: $L$ := $\varnothing$;                               ▷ list of disqualified entries
 3: $H$ := $\varnothing$;                               ▷ heap of objects, organized by $P_o^{min}$
 4: $P^m$ := 0;                                         ▷ $P^{min}$ of $m$-th object in $H$
 5: while $P^{first} > P^m$ and more objects in $R$ do
 6:     $x$ := next NN of $q$ in $R$;
        ▷ during BF-search, each non-leaf entry with $P^{first} \cdot e^{maxE} < t$ is removed from Best-First heap and inserted into $L$
 7:     compute $P_x^{min}$ and $P_x^{max}$ by using $P^{first}$, $L$ and $E_x$;
 8:     while $P_x^{min} < P^m \leq P_x^{max}$ do
 9:         pick the entry $e$ with the smallest $mind(q,e)$ in $L$; remove $e$ from $L$;
10:         if $e$ is an object then        ▷ $e$ is an object closer to $q$ than $x$ is
11:             $P^{first}$ := $P^{first} \cdot (1 - E_e)$;
12:             for all entry $o \in H$ such that $d(q,e) \leq d(q,o)$ do
13:                 $P_o^{first}$ := $P_o^{first} \cdot (1 - E_e)$;
14:         else
15:             read node $n_e$ pointed by $e$ and insert all entries of $n_e$ into $L$;
16:         compute $P_x^{min}$ and $P_x^{max}$ by using $P^{first}$, $L$ and $E_x$;
17:     if $P_x^{min} > P^m$ then
18:         enheap($H$,($x$,$P_x^{first}$:=$P^{first}$,$P_x^{min}$,$P_x^{max}$));
19:     if $H$ is changed or $L$ is changed then
20:         recompute, for each $o \in H$, $P_o^{min}$ and $P_o^{max}$ by using $P_o^{first}$, $L$ and $E_o$;
21:         $P^m$ := $m$-th $P^{min}$ in $H$;
22:         remove entries $o$ from $H$ with $P_o^{max} < P^m$;
23:     $P^{first}$ := $P^{first} \cdot (1 - E_x)$;
24: while $|H| > m$ and $|L| > 0$ do
25:     apply Lines 9–16;
26:     apply Lines 20–22;
27:     remove $e$ from $L$ with $mind(q,e) > \max\{d(q,o) : o \in H\}$;
```

13, P_o^{first} (for some entries in H) is updated for each object e found no further than o from q. The new P_o^{first} value is used to update P_o^{min} and potentially the order of objects in H at Lines 20–21. Note that H may store more than m entries, since there may be objects o in it satisfying $P_o^{max} \geq P^m \geq P_o^{min}$. However, entries o are removed from H once $P_o^{max} < P^m$. The algorithm does not need to access any more objects from the Best-First heap as soon as $P^{first} < P^m$. In case H has more than m objects at that point, we need to refine the probability ranges of the objects in H (by processing entries in L) until we have the best m objects. In this case, entries e are removed from L once $mind(q,e) > \max\{d(q,o) : o \in H\}$ because such entries cannot be used to refine the probability ranges of the objects in H.

4.3 Query Evaluation using 3D R–trees

An alternative method for indexing existentially uncertain data is to model the confidences E_x of objects x as an additional dimension and use a 3D R–tree to index the objects. Now, each non-leaf entry e in the tree, apart from the spatial dimensions, has a range $[e^{minE}, e^{maxE}]$ within which the existential probabilities of all objects in its subtree fall.

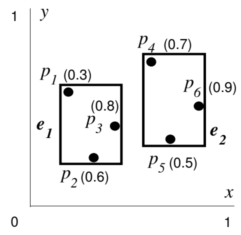
Figure 5 illustrates the differences between the augmented 2D R–tree and the 3D R–tree. Figure 5a depicts the structure of the augmented 2D R–tree for the points $p_1, p_2, \cdots, p_6$. The R*–tree insertion algorithm [14] aims at grouping the points into leaf nodes such that the their MBR areas are minimized. As such, the (non-leaf) entry e_1 points to a leaf node containing the points p_1, p_2, p_3; whereas the entry e_2 points to a leaf node containing the points p_4, p_5, p_6. The spatial X, Y ranges, and the augmented probability, for these two entries in the augmented 2D R–tree are listed in Figure 5c. Note that each entry consists of 6 values (including its child node pointer). Figure 5b shows the structure of the 3D R–tree, for the same set of points. The R*–tree insertion optimizes the bounding rectangles of nodes defined by three dimensions: spatial dimensions X and Y, as well as the probability dimension E. Hence, the entry e_1 points to a leaf node containing the points p_1, p_2, p_5; whereas the entry e_2 points to a leaf node containing the points p_3, p_4, p_6. The values stored in these entries in the 3D R–tree are also listed in Figure 5c. Now, each entry consists of 7 values (including its child node pointer), implying that the fanout of the 3D R–tree is slightly smaller than the augmented 2D R–tree.

The methods for processing the probabilistic range and NN queries over the augmented 2D R–tree (in Section 4.2) are applicable for the 3D R–tree, since each tree entry still stores an e^{maxE} value. In particular, for the NN query, we utilize e^{minE} to derive tighter probability ranges:

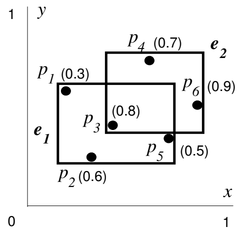
$$P_x^{min} = P^{first} \cdot E_x \cdot \prod_{e \in L \,\wedge\, mind(q,e) \leq d(q,x)} (1 - e^{minE})(1 - e^{maxE})^{(e^{num}-1)} \quad (3)$$

$$P_x^{max} = P^{first} \cdot E_x \cdot \prod_{e \in L \,\wedge\, maxd(q,e) \leq d(q,x)} (1 - e^{minE})^{(e^{num}-1)}(1 - e^{maxE}) \quad (4)$$

If the exact number e^{num} of objects in the subtree pointed by e is not known, we can use the fanout f and the minimum node utilization (0.4 for R*–trees) and replace e^{num} by $f^{level(e)}$ in Equation 3 and by $(0.4 \cdot f)^{level(e)}$ in Equation 4.



(a) Augmented 2D R–tree



(b) 3D R–tree

Name	Entry e_1	Entry e_2	Grouping	Fields
Aug. 2D R–tree	X=[0.15,0.40] Y=[0.20,0.60] $maxE$=0.80	X=[0.55,0.85] Y=[0.30,0.80] $maxE$=0.90	Spatial only	6 (1+4+1)
3D R–tree	X=[0.15,0.70] Y=[0.20,0.60] E=[0.30,0.60]	X=[0.40,0.85] Y=[0.40,0.80] E=[0.70,0.90]	Both spatial and probabilities	7 (1+4+2)

(c) Comparison between the two trees

Fig. 5. Structures of different R–tree variants

Interestingly, the query performance of the 3D R–tree is not necessarily better than the augmented 2D R–tree. A careful examination of Equations 3,4 reveals that these probability bounds are determined by both the spatial and probabilistic intervals of the entries. Even the e^{minE} values in the 3D R–tree are helpful for tightening the bounds, this effect is counteracted by the large spatial bounding rectangles in the tree. Thus, more (disqualified) entries $e \in L$ satisfy the $mind(q,e) \leq d(q,x)$ condition in Equation 3 and fewer entries $e \in L$ satisfy the $maxd(q,e) \leq d(q,x)$ condition in Equation 4. Hence, the final probability bounds for the 3D R–tree may indeed become looser. Besides, the 3D R–tree has a slightly smaller fanout, which may lead to more page accesses.

5 Advanced Spatial Queries

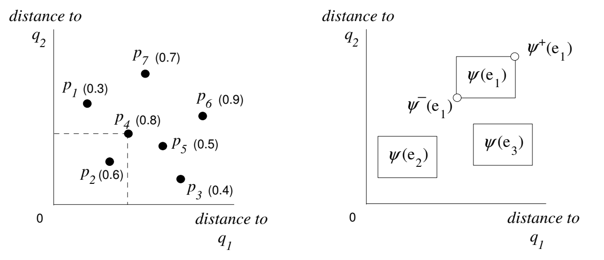
In this section, we discuss probabilistic variants of spatial skyline queries [15] and reverse nearest neighbor queries [16], due to their applications in spatial decision support systems. For each query type, we first present its background, then define its probabilistic variant, and finally develop corresponding query algorithms for the thresholding and ranking versions.

5.1 Spatial Skyline Queries

Given a set Q of query points (e.g., user locations) and two points p and p' (e.g., two facilities), p *spatially dominates* [15] p' when all query points in Q are closer to p than to p':

$$\forall\, q \in Q,\ d(q,p) \leq d(q,p') \tag{5}$$

Given a point dataset R, its *spatial skyline* [15] (with respect to Q) contains the objects $p \in R$ that are not spatially dominated by any other object in R. As an example, consider the distances of the stations $p_i \in R$ from a group of 2 users $Q = \{q_1, q_2\}$ in Figure 6a. The spatial skyline contains p_1, p_2, and p_3. The main application of spatial skyline queries is to discover facilities that are not farther than other facilities, for *all* users.



(a) a point set (b) dominance relationship

Fig. 6. Feature space defined by the distances from query points

To ease our discussion, we first introduce some notation. The spatial skyline query is formulated in a *feature space* in which each dimension captures the distance to a query point. Given a set $Q = \{q_1, q_2, \cdots, q_z\}$ of query points, a spatial location (i.e., data point) p (or a MBR e) can be mapped to a point $\psi(p)$ (or a MBR $\psi(e)$) in a z-dimensional *feature* space, where the i-th dimension captures the distances of the points to q_i (for $i \in [1,z]$). Table 3 illustrates the mapping of a data point or an MBR (corresponding to a non-leaf R–tree entry, assuming that the data points are indexed by an R–tree) to this feature space.

As a shorthand notation, we use $\psi(p) \succ \psi(p')$ to mean that p spatially dominates p'. Let $\psi^-(e)$ and $\psi^+(e)$ be the lower and upper bound corners of the MBR $\psi(e)$ respectively (see Figure 6b). Since $\psi^+(e_2) \succ \psi^-(e_1)$, each point in e_2 must spatially dominate all points in e_1. On the other hand, only some point in e_3 may spatially dominate some points in e_1 as $\psi^-(e_3) \succ \psi^+(e_1)$ and $\psi^+(e_3) \not\succ \psi^-(e_1)$.

TABLE 3
Mapping from the original space to the feature space

Original space	**Feature space $\psi(\cdot)$, i-th dimension**
Point p	$d(q_i, p)$
MBR e	$[mind(q_i,e), maxd(q_i,e)]$

With the above mapping technique, [17] propose an R–tree based algorithm for computing the dynamic skyline in the feature space. The idea is to apply the best-first search algorithm [9] on the R–tree to visit the entries e, from the origin 0^z in the feature space, in ascending order of the value:

$$\omega_Q(e) = \sum_{q \in Q} mind(q,e) \tag{6}$$

[17] proved that a point must be discovered earlier than the points it dominates (if any). Hence, a point p is reported as a result if it cannot be dominated by any examined points. We then adapt the above algorithm for the probabilistic spatial skyline query.

5.1.0.2 Probabilistic spatial skyline query and its properties: For existentially uncertain data, a point x is a query result with probability:

$$P_x = E_x \cdot \prod_{x' \in R \,\wedge\, \psi(x') \succ \psi(x)} (1 - E_{x'}) \tag{7}$$

which corresponds to the case that x exists and the points dominating x do not exist. A *probabilistic spatial skyline query* takes as input a set Q of query points and returns all (x, P_x) pairs, such that $x \in R$ and x belongs to the spatial skyline of Q with probability $P_x > 0$. For instance, in Figure 6a, we have $P_{p_4} = 0.8 \cdot (1 - 0.6) = 0.32$ because p_2 dominates p_4. Since no points dominate p_2, we derive $P_{p_2} = 0.6 \cdot 1 = 0.6$. The probability of other points can be computed in a similar way.

In Section 4.1.2, we used a single variable P^{first} to incrementally compute the upper bound probability for the remaining objects to be examined. This technique is inapplicable to the spatial skyline query, since the points visited in decreasing order from the origin do not necessarily influence the points that will be visited next. For instance, the existence of point p_2 in Figure 6a does not influence the probability that p_1 (which is further than p_2 from the origin and will be visited next) is in the skyline. However, it influences p_4, since p_4 is dominated by p_2. In general, given a set $S \subseteq R$ of already examined points, in order of their distance to the origin, an upper bound $P_x^{first}(S)$ of the probability that point x is in the skyline with respect to S can be computed by:

$$P_x^{first}(S) = \prod_{x' \in S \,\wedge\, \psi(x') \succ \psi(x)} (1 - E_{x'}) \tag{8}$$

For a MBR e, the upper bound probability $P_e^{first}(S)$ of any point in e to be in the skyline can be computed as follows:

$$P_e^{first}(S) = \prod_{x' \in S \,\wedge\, \psi(x') \succ \psi^-(e)} (1 - E_{x'}) \quad (9)$$

, since $\psi^-(e)$ dominates any point in e. Next, we discuss how thresholding and ranking versions of the query are evaluated on a 2D R–tree.

5.1.0.3 Thresholding and ranking: Assume that we want to find the points with probability at least t to be in the skyline. Algorithm 6 describes the procedure to retrieve these points from a 2D R–tree. At Line 3, objects are incrementally retrieved from the tree in increasing order of their $\omega_Q(x)$ value, which is defined in Equation 6. Set S is used for storing objects examined so far, in order to derive the probability $P_x^{first}(S)$ of remaining objects (using Equation 8). The probability derivation of P_x as $P_x^{first}(S) \cdot E_x$ is correct because [17] proved that all the points dominating x must have been examined before x (and stored into S). When $P_x \geq t$, x is reported as a result. In case $P_x^{first}(S) < t$, any (remaining) point x' dominated by x must be at least dominated by the same subset of points in S such that $P_{x'}^{first}(S) < t$. Thus, x is inserted into S only when $P_x^{first}(S) \geq t$. Following the above logic, we can optimize the algorithm at Line 3 by removing non-leaf entries with $P_e^{first}(S) < t$ from the Best-First heap.

Algorithm 6 Probabilistic spatial skyline on a 2D R–tree with thresholding

```
    Algorithm PTSKY2D(Query set $Q$, 2D R–tree on $R$, Threshold $t$)
 1: $S$ := ∅;                                          ▷ set of examined objects
 2: while more objects in $R$ do
 3:     $x$ := next point in $R$ with minimum $\omega_Q(x)$;
    ▷ during BF-search, non-leaf entries $e$ with $P_e^{first}(S) < t$ are removed
    from Best-First heap
 4:     $P_x$ := $P_x^{first}(S) \cdot E_x$;
 5:     if $P_x \geq t$ then
 6:         output ($x$,$P_x$);
 7:     if $P_x^{first}(S) \geq t$ then
 8:         insert $x$ into $S$;
```

Threshold-based retrieval (of Algorithm 6) can be extended to retrieve the m points with the highest probability to be in the skyline (i.e., the ranking probabilistic variant of the query). The general idea is to maintain a heap H of m objects with the highest P_x found so far. In addition, we replace the fixed threshold t by a floating bound P_m, which indicates the m-th highest P_x in H. If P_x is found to be greater than P^m, then the result heap H and the bound P^m are updated. As P^m increases, (unnecessary) objects with $P_o^{first}(S) < P^m$ are removed from S in order to save space.

5.1.0.4 Extensions for augmented R–trees: As discussed in Section 4.2, augmented R–trees can be used to improve the query efficiency. Algorithm 7 generalizes Algorithm 4 to utilize information from an augmented 2D R–tree. During the BF-search at Line 4, each non-leaf entry e with $P_e^{first}(S) \cdot e^{maxE} < t$ is removed from Best-First heap because they cannot contain any results. If the removed entry has $P_e^{first}(S)$ at least t, then it may influence the remaining points and e is inserted into the list L for further processing. At Line 5, the lower bound P_x^{min} and upper bound P_x^{max} probabilities of a point x are computed from S, L and E_x using Equations 10 and 11 respectively.

$$P_x^{min} = P_x^{first}(S) \cdot E_x \cdot \prod_{e \in L \,\wedge\, \psi^-(e) \succ \psi(x)} (1 - e^{maxE})^{e^{num}} \quad (10)$$

$$P_x^{max} = P_x^{first}(S) \cdot E_x \cdot \prod_{e \in L \,\wedge\, \psi^+(e) \succ \psi(x)} (1 - e^{maxE}) \quad (11)$$

When $P_x^{min} < t \leq P_x^{max}$, we need to refine the probability range for x (Lines 6-13). After that, x is reported as a result if $P_x^{min} \geq t$. In case $P_x^{first}(S) \geq t$, x is inserted into S because it influences the probability of other points that may end up in the result.

Algorithm 7 Probabilistic spatial skyline on an augmented 2D R–tree with thresholding

```
    Algorithm PTSKY2Daug(Query point $q$, Augmented 2D R–tree on $R$,
    Threshold $t$)
 1: $S$ := ∅;                                          ▷ set of examined objects
 2: $L$ := ∅;                                        ▷ list of disqualified entries
 3: while more objects in $R$ do
 4:     $x$ := next point in $R$ with minimum $\omega_Q(x)$;
    ▷ during BF-search, each non-leaf entry $e$ with $P_e^{first}(S) \cdot e^{maxE} < t$
    is removed from Best-First heap, and inserted into $L$ if $P_e^{first}(S) \geq t$
 5:     compute $P_x^{min}$ and $P_x^{max}$ by using $P_x^{first}(S)$, $L$ and $E_x$;
 6:     while $P_x^{min} < t \leq P_x^{max}$ do
 7:         pick the entry $e$ with the smallest $\omega_Q(e)$ in $L$; remove $e$ from $L$;
 8:         if $P_e^{first}(S) \geq t$ then       ▷ $e$ may influence the probability of
    potential results
 9:             if $e$ is an object then
10:                 insert $e$ into $S$;
11:             else
12:                 read node $n_e$ pointed by $e$ and insert all entries of $n_e$ into
    $L$;
13:         compute $P_x^{min}$ and $P_x^{max}$ by using $P_x^{first}(S)$, $L$ and $E_x$;
14:     if $P_x^{min} \geq t$ then
15:         output ($x$,$P_x^{min}$,$P_x^{max}$);
16:     if $P_x^{first}(S) \geq t$ then
17:         insert $x$ into $S$;
```

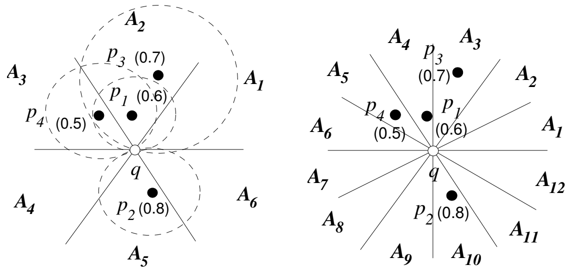
Similarly, we can generalize the algorithm for evaluating ranking spatial skyline queries on an augmented 2D R–tree. Regarding 3D R–trees, Equations 12 and 13 are applied to compute the P_x^{min} and P_x^{max} values of a point x respectively. In case the exact number e^{num} of objects in the subtree pointed by e is not known, we can use the fanout f and the minimum node utilization (0.4) and replace e^{num} by $f^{level(e)}$ in Equations 10,12, and by $(0.4 \cdot f)^{level(e)}$ in Equation 13.

$$P_x^{min} = P_x^{first}(S) \cdot E_x \cdot \prod_{e \in L \,\wedge\, \psi^-(e) \succ \psi(x)} (1 - e^{minE})(1 - e^{maxE})^{(e^{num}-1)} \quad (12)$$

$$P_x^{max} = P_x^{first}(S) \cdot E_x \cdot \prod_{e \in L \,\wedge\, \psi^+(e) \succ \psi(x)} (1 - e^{minE})^{(e^{num}-1)}(1 - e^{maxE}) \quad (13)$$

5.2 Reverse Nearest Neighbor Queries

Given a point dataset R and a query point q, a reverse nearest neighbor (RNN) query [16] retrieves the objects $p \in R$ having q as their nearest neighbor. This query has applications in decision support and resource allocation. [16], [18] develop R–tree based algorithms for reverse nearest neighbor queries. In this section, we extend the geometric partitioning method of [16] to solve probabilistic versions of this problem. According to [16], an RNN query can be answered in two steps. In the filter step, the 2D data space (shown in Figure 7a) is divided into six equal sectors $A_1, A_2, \cdots, A_6$ around the query point q. The NN of q in each sector (if any) is included into the *candidate set*. In the example, the candidates are the points p_1 (in A_2), p_2 (in A_5), and p_4 (in A_3). [16] proved that the candidate set is a superset of the result set. During the refinement step, each candidate is verified by retrieving its NN. A candidate (e.g., p_2) is reported as a result if its NN is q. Otherwise, the candidate (e.g., p_3) is a *false hit* and it is discarded.



(a) 6-sector partitioning (b) 12-sector partitioning

Fig. 7. Reverse nearest neighbor query example

5.2.0.5 Probabilistic reverse nearest neighbor query and its properties: For existentially uncertain data, a point x belongs to the RNN set of q with probability:

$$P_x = E_x \cdot \prod_{x' \in R \,\wedge\, x' \neq x \,\wedge\, d(x,x') < d(q,x)} (1 - E_{x'}) \tag{14}$$

which corresponds to the case that x exists and the points x' that are closer to x than to q do not exist. For instance, in Figure 7a, we have $P_{p_3} = 0.7 \cdot (1 - 0.6) \cdot (1 - 0.5) = 0.14$ because p_3 is closer to p_1 and p_4 than to q. Since p_2 is closer to q than to other points, we derive $P_{p_2} = 0.8 \cdot 1 = 0.8$. The probability of other points can be computed in a similar way.

Similarly to the skyline query and unlike the NN query of Section 4.1.2, we cannot define an order of visiting the points around q, such that the upper bound probability of remaining points to be in the RNN result, can be maintained by incrementally updating a single P^{first} value. To elaborate this, suppose that we first examined the point p_1 in Figure 7a. Note that p_1 only influences the probabilities of p_3, p_4 but not that of p_2. The example also demonstrates that the upper bound probability of a point can be computed by using examined points. Given a set $S \subseteq R$ of (examined) points, the upper bound probability P_x^{first} of a point x with respect to q and S is defined as:

$$P_x^{first} = \prod_{x' \in S \,\wedge\, x' \neq x \,\wedge\, d(x,x') < d(q,x)} (1 - E_{x'}) \tag{15}$$

Geometric properties of reverse nearest neighbors can be exploited to derive the upper bound probability of remaining points in a specific sector (see Figure 7a). [16] proved that, if two points p and p' are in the same sector and q is closer to p than to p', then p' must be closer to p than to q. Based on this property, a natural solution for the query is to retrieve the points in ascending order of their distances from q. For each sector A_j, its $P_{A_j}^{first}$ value is used as the upper bound probability of any remaining point in A_j. $P_{A_j}^{first}$ is set to 1 initially and it is multiplied by the factor $(1 - E_x)$ when a new point x is discovered in A_j.

We observe that introducing additional sectors may help deriving tighter probability bounds for unexamined points. Consider the 12-sector partitioning shown in Figure 7b. When a point (say, p_1) is discovered in the sector A_4, it is used to update $P_{A_j}^{first}$ for the sectors (i.e., A_3, A_4, A_5) that are within (maximum) 60° angular range from A_4. Conversely, the probability bound of a sector is contributed by the points within $(30 \cdot 3)^\circ = 90^\circ$ angular range. Recall that, for the original 6-sector partitioning in [16], a sector is only affected by the points within 60° angular range. In general, given a positive integer V, in the $(6V)$-sector partitioning scheme, $(2V - 1)$ sectors need to be examined per visited point. The more sectors we have, the tighter probability bounds are derived for (unexamined points in) the sectors, and the earlier unqualified sectors can be pruned. On the other hand, the computational overhead of updating probability bounds for the sectors is proportional to V. In Section 6, we will determine an appropriate number of sectors that achieves significant I/O cost reduction and adds little computational overhead for updating probability bounds for the sectors. Next, we discuss how this partitioning scheme can be used to evaluate probabilistic reverse nearest neighbor queries.

5.2.0.6 Thresholding and ranking: Algorithm 8 shows how thresholding RNN queries are evaluated on a 2D–tree. The system parameter κ specifies the number of sectors to be used. First, the space is divided into κ sectors A_i and their probability bounds $P_{A_i}^{first}$ are set to 1. The algorithm maintains candidate objects (i.e., potential results) in a set C and delays computing the actual probability of a candidate until all objects influencing it have been examined. Examined objects are stored in the set S and they are used to compute upper bound probabilities for candidate objects. Both C and S are initialized to empty sets.

At Line 6, we apply Best-First search [9] to incrementally retrieve the next NN (i.e., the object x) of q from the tree R. Suppose that $A(x)$ denotes the sector containing x. If the upper bound probability $E_x \cdot P_{A(x)}^{first}$ is greater than the threshold t, then a tighter upper bound probability $E_x \cdot P_x^{first}$ is computed, by examining the objects in S (see Equation 15). When the above probability is at least t, the object x is inserted into C. After that, x is inserted into S and $P_{A_i}^{first}$ is updated for each sector A_i within 60° angular range from $A(x)$. In turn, x is used to update the P_o^{first} value for objects in C, and those satisfying $E_o \cdot P_o^{first} < t$ are removed from C. If the last deheaped distance d_{last} (from the Best-First heap) is greater than $2 \cdot d(q, o)$ for a candidate object $o \in C$, then all entries in Best-First heap cannot affect the probability of o. At Line 17, we compute the actual probability P_o as $P_o^{first} \cdot E_o$ and report o as a result when $P_o \geq t$. The loop (Lines 5-20) continues while some sector may contain potential results to be discovered or C is not empty.

The above algorithm can be extended to retrieve the top-m ranked reverse nearest neighbors from the 2D R–tree. It

Algorithm 8 Probabilistic reverse nearest neighbor on a 2D R–tree with thresholding

```
    Algorithm PTRNN2D(Query point $q$, 2D R–tree on $R$, Threshold $t$)
    $\kappa$: number of sectors (system parameter)
 1: divide the space into $\kappa$ equal sectors around $q$: $A_i, i \in [1,\kappa]$;
 2: for all $i \in [1,\kappa]$ do
 3:     $P^{first}_{A_i}$:=1;  ▷ Upper prob. bound of remaining objects in sector $A_i$
 4: $C$:=∅; $S$:=∅;
 5: while $\exists i \in [1,\kappa], P^{first}_{A_i} \geq t$ or $|C| > 0$ do
 6:     $x$ := next NN of $q$ in $R$; $d_{last} := d(q,x)$;
 7:     let $A(x)$ be the sector of $x$;
 8:     if $E_x \cdot P^{first}_{A(x)} \geq t$ and $E_x \cdot P^{first}_x \geq t$ then  ▷ apply cheap filter first, and then expensive filter
 9:         $C := C \cup \{x\}$;
10:     $S := S \cup \{x\}$;
11:     for all $i \in [1,\kappa]$ such that $A_i$ is within (maximum) 60° angular range from $A(x)$ do
12:         $P^{first}_{A_i} := P^{first}_{A_i} \cdot (1 - E_x)$;
13:     for all $o \in C$ such that $d(x,o) < d(q,o)$ do  ▷ update $P^{first}_o$
14:         $P^{first}_o$:=$P^{first}_o \cdot (1 - E_x)$;
15:     remove objects $o$ from $C$ with $E_o \cdot P^{first}_o < t$;  ▷ filter false hits
16:     for all $o \in C$ such that $d_{last} \geq 2 \cdot d(q,o)$ do  ▷ entries in Best-First heap cannot affect the probability of $o$
17:         $P_o := P^{first}_o \cdot E_o$;
18:         if $P_o \geq t$ then
19:             output ($o$,$P_o$);
20:         remove $o$ from $C$;
```

Algorithm 9 Probabilistic reverse nearest neighbor on an augmented 2D R–tree with thresholding

```
    Algorithm PTRNN2Daug(Query point $q$, Augmented 2D R–tree on $R$, Threshold $t$)
    $\kappa$: number of sectors (system parameter)
 1: divide the space into $\kappa$ equal sectors around $q$: $A_i, i \in [1,\kappa]$;
 2: for all $i \in [1,\kappa]$ do
 3:     $P^{first}_{A_i}$:=1;  ▷ Upper prob. bound of remaining objects in sector $A_i$
 4: $C$:=∅; $S$:=∅; $L$:=∅;
 5: while more objects in $R$ do
 6:     $x$ := next NN of $q$ in $R$; $d_{last} := d(q,x)$;
        ▷ during BF-search, each non-leaf entry $e$ intersecting only sector(s) with $P^{first}_{A_i} \cdot e^{maxE} < t$ is removed from Best-First heap and inserted into $L$
 7:     apply Lines 7–15 of Algorithm 8;
 8:     for all $o \in C$ such that $d_{last} \geq 2 \cdot d(q,o)$ do ▷ entries in Best-First heap cannot affect the probability of $o$
 9:         while $P^{first}_o \cdot E_o \geq t$ and $\exists e \in L, mind(e,o) < d(q,o)$ do ▷ refinement step
10:             remove the entry $e$ with the smallest $mind(e,o)$ in $L$;
11:             if $e$ is an object then
12:                 apply Lines 11–15 of Algo. 8, but by replacing $x$ with $e$;
13:             else
14:                 read the node $n_e$ pointed by $e$ and insert all entries of $n_e$ into $L$;
15:         $P_o := P^{first}_o \cdot E_o$;
16:         if $P_o \geq t$ then
17:             output ($o$,$P_o$);
18:         remove $o$ from $C$;
19: for all $o \in C$ do  ▷ verify remaining candidates in $C$
20:     apply Lines 9–18 of this Algorithm;
```

maintains a heap H of m objects with the highest P_x found so far. In addition, we replace the fixed threshold t by a floating bound P_m, which indicates the m-th highest P_x in H. At Lines 18-19, if P_o is greater than P^m, then the result heap H and the bound P^m are updated.

Besides, the above thresholding[3] algorithm can be adapted to Algorithm 9, for augmented 2D R–trees and 3D R–trees. At Line 6, each non-leaf entry e intersecting only sector(s) with $P^{first}_{A_i} \cdot e^{maxE} < t$ is removed from Best-First heap because they cannot contain any results. However, such entries may affect the probability of other points so they are inserted into L. Lines 9-14 compute the actual probability for such an object, by refining its P^{first}_o value with the entries in L. For this, we check whether the upper bound probability $P^{first}_o \cdot E_o$ is above t and q is closer to some entries in L than to q. If so, the entry closest to o is removed from L and its child node n_e is accessed. In case n_e points to a tree node, all its entries are inserted into L. Otherwise, entries in n_e are used to update the set S, P^{first}_o values of candidate objects, and $P^{first}_{A_i}$ values of sectors. At Lines 19-20, the remaining candidates in C are verified by accessing entries in L that may influence their probabilities.

6 EXPERIMENTAL EVALUATION

In this section, we evaluate the efficiency of the proposed techniques. We compare the performances of five indexes and their corresponding algorithms for thresholding and ranking versions of range queries, nearest neighbor search, skyline queries and reverse nearest neighbor retrieval. The five indexes are (i) a simple 2D R–tree (denoted by 2D), (ii) a 2D R–tree, where each non-leaf entry e is augmented with e^{maxE} (denoted by AUG), (iii) a 2D R–tree, where each non-leaf entry e is augmented with e^{maxE} and e^{num} (i.e., the number of objects in the subtree indexed by it), denoted by AUG COUNT, (iv) a 3D R–tree (denoted by 3D), and (v) a 3D R–tree, where each non-leaf entry e is augmented with e^{num} (denoted by 3D COUNT). For indexes (iv) and (v), all (spatial/probability) dimensions are normalized to the same domain interval. Note that index (i) captures minimum information in non-leaf entries and occupies the least space, whereas index (v) is at the other end (entries capture maximum information and the index occupies the most space).

All algorithms were implemented in C++. Experiments were run on a PC with a Pentium D CPU of 2.8GHz. The page size of indexes was set to 1Kb; the relative performance results of the above methods were observed for other page sizes (up to 8Kb). No memory buffers are used for caching disk pages between different queries; the number of node accesses directly reflects the I/O cost. In each experiment, the measured I/O cost is the average I/O cost of 100 queries with the same parameter values (but with different locations randomly chosen from the dataset). For range queries, nearest neighbor search, and reverse nearest neighbor retrieval, the I/O time is over 90% of the total execution cost so the CPU time is not reported.

6.1 Description of Data

For our experiments, we used various real datasets of different sizes and object distributions, described in Table 4. The datasets TG and SF are obtained from [19] while

3. Adaptations of ranking algorithms for RNN queries are omitted due to space constraints.

TABLE 4
I/O cost of thresholding/ranking NN on different datasets, $t = 0.005$, $m = 10$

Dataset	**Size**	2D	AUG	AUG COUNT	3D	3D COUNT
San Joaquin roads (TG)	18263	48.68/ 49.33	16.49/ 17.54	19.75/ 20.68	28.60/ 29.56	31.11/ 32.81
Greece roads (GR)	23268	77.69/ 83.30	17.23/ 23.87	21.79/ 29.07	24.58/ 34.06	25.12/ 36.84
Long Beach roads (LB)	53145	93.14/ 103.20	19.07/ 28.78	21.49/ 32.97	24.17/ 38.32	25.80/ 42.65
LA streets (LA)	131461	67.38/ 71.21	19.19/ 22.90	21.89/ 26.43	32.83/ 36.43	36.17/ 41.59
San Francisco roads (SF)	174956	70.50/ 73.62	20.39/ 24.80	22.05/ 27.51	32.71/ 37.48	36.17/ 42.72
Tiger streams (TS)	194971	99.30/ 110.32	18.36/ 34.48	19.69/ 39.41	27.37/ 47.62	32.94/ 58.74

the other datasets are obtained from the R–tree Portal (`www.rtreeportal.org`).

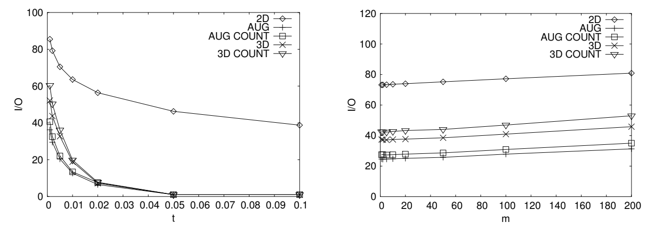
Due to the lack of a real spatial dataset with objects having existential probabilities, we generated probabilities for the objects, using the following methodology. First we generated $K = 20$ *anchor* points randomly on the map, following the data distribution. These points model locations around which there is large certainty for the existence of data (e.g., they could be antennas of receivers close to which information is accurate). For each point x of the dataset, we (i) find the closest anchor a and (ii) assign an existential probability proportional to $\frac{1}{(c \cdot dist(x,a))^\theta}$. Thus, the distribution of probabilities around the anchors is a Zipfian one. The probabilities are normalized (using c) with respect to the maximum probability (1) corresponding to the anchor point. The default skew value is $\theta = 1$; experiments on different skew values can be found in our preliminary work [20].

6.2 Experimental Results

Table 4 shows the performances of the five indexes for thresholding and ranking NN queries on different datasets. We fix $t = 0.005$ for thresholding NN queries and $m = 10$ for ranking NN queries.[4] Observe that the augmented and 3D R–trees perform better than the 2D R–tree, even though they are larger in size. Algorithms 4 and 5 manage to prune a large number of nodes that do not contain query results, which are otherwise visited in the simple 2D R–tree index. The cost of 2D R–tree variants (i.e., methods AUG, AUG COUNT) does not change much with the database size. The I/O costs of 3D R–tree variants increase slowly as the database size increases. This is due to the fact that 3D R–trees group entries using both spatial and probability dimensions, but the query algorithms mainly search for objects based on spatial dimensions.

In subsequent experiments, we compare the performance of the indexes on the SF dataset and default parameter values are $t = 0.005$ and $m = 10$ for thresholding and ranking queries respectively. Figure 8 shows the I/O performance of the indexes for thresholding and ranking queries. Augmented and

4. A small value for t is necessary in order to observe difference between the indexes. Larger values for t will be tested in a subsequent experiment.



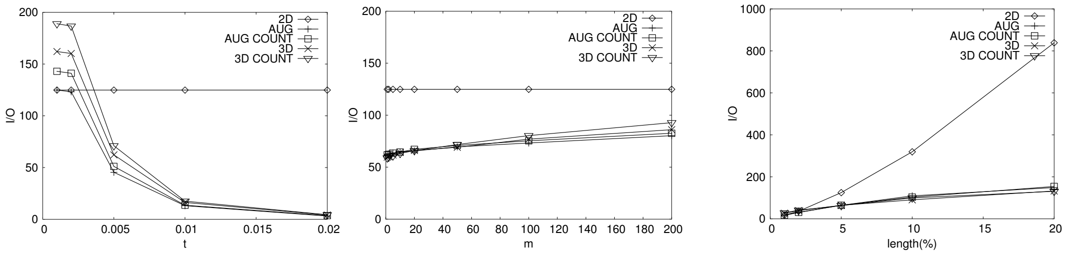
(a) thresholding queries (b) ranking queries

Fig. 8. NN queries on the SF dataset, $\theta = 1$

3D R–trees perform much better than the simple 2D R–tree for all tested values of t and m. For $t \geq 0.02$, less than 5 accesses are required to find the query result when using the four advanced indexes and Algorithms 4 and 5. When comparing these indexes, we observe that augmenting e^{num} is not a good idea; using the fanout f gives accurate enough estimations of P^{min} and P^{max}. Thus the extra space (translated to extra accesses) required for augmenting e^{num} does not pay off. In addition, the augmented R–tree performs better than the 3D R–tree. First, the 3D R–tree occupies more space (the capacity of each non-leaf node is smaller) and results in more accesses, since the extra space is not compensated by tighter P^{min} and P^{max} (see Equations 3 and 4). Second, since the 3D R–tree groups entries to nodes using the existential probabilities as well as spatial dimensions, it does not achieve as good partitioning as the one using the spatial dimensions only; however, search is performed primarily using the spatial dimensions.

Next, we examine the performances of range queries on the indexes. The parameter len denotes the extent of the query window (in each dimension), whose default value is set to 5% of the domain length. Figure 9a and 9b show the cost of thresholding and ranking queries as a function of t and m respectively. Except for the simple 2D R–tree, all indexes follow similar trends as in probabilistic nearest neighbor queries. The cost of range queries on the 2D R–tree is independent of t and m as all points within the spatial range are retrieved. Observe that for very small t, the augmented and 3D indexes may perform worse than the 2D R–tree because (i) they prune no or very few directory entries that have lower e^{maxE} than t and (ii) they are larger in size than the simple 2D R–tree. Similarly, P^m decreases with m, affecting the costs of the advanced methods. The 3D R–tree performs worse than the augmented 2D R–tree also for range queries. Figure 9c shows the cost of thresholding queries as a function of len. The costs of all methods increase with len.

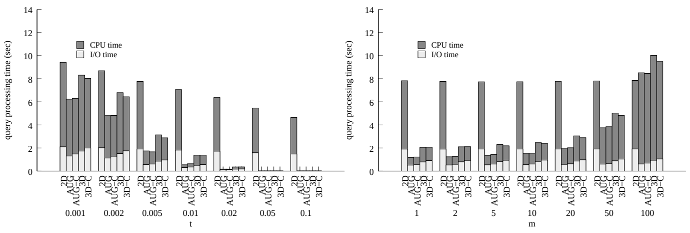
We proceed to compare the performances of spatial skyline queries on the indexes. For each query, a set of $|Q|$ query points are randomly generated in a query window with side length len, such that the window follows the data distribution. The default values of $|Q|$ and len are 6 and 5% of the domain range respectively. Figure 10 shows the I/O-CPU time breakdown of thresholding and ranking queries as a function of t an m respectively. Each page fault is charged 10 milliseconds of I/O time. Observe that the method AUG outperforms its competitors for a wide range of parameters. In terms of I/O,



(a) thresholding queries vs t (b) ranking queries vs m (c) thresholding queries vs len, $t = 0.005$

Fig. 9. Range queries on the SF dataset

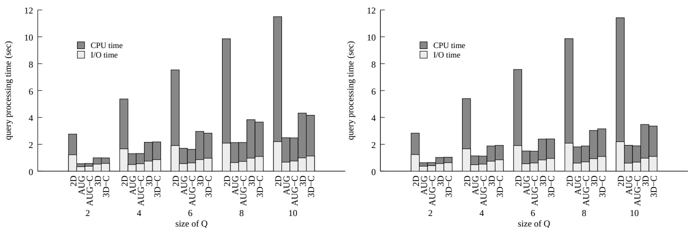
the trends are similar to the ones in Figure 8. However, the CPU time of augmented and 3D trees becomes high at low t value and high m value.



(a) thresholding queries vs t (b) ranking queries vs m

Fig. 10. Spatial skyline queries on the SF dataset, $|Q| = 6$, $len = 5\%$

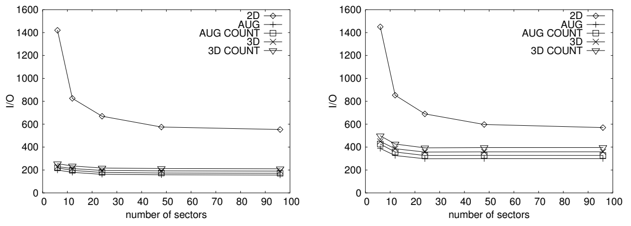
Figure 11 plots the cost of the indexes by varying the number $|Q|$ of query points. In general, when $|Q|$ increases, a point is spatially dominated by fewer points, and thus the probability of the point to be in the skyline increases. Thus, more points need to be examined by thresholding queries and its I/O cost increases rapidly. On the other hand, P_m increases with $|Q|$, strengthening the pruning power of advanced indexes. Thus, the cost of ranking queries increases at a slower rate.



(a) thresholding queries, $t = 0.005$ (b) ranking queries, $m = 10$

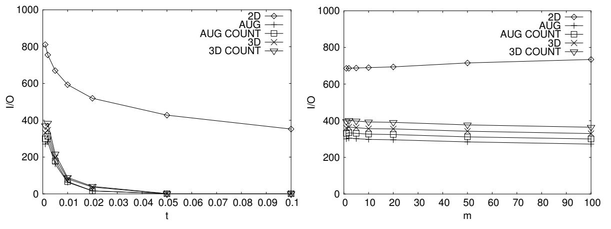
Fig. 11. Spatial skyline queries on the SF dataset, varying $|Q|$

Finally, we study the performance of the indexes for reverse nearest neighbor queries. Figure 12 shows the effect of the number of sectors in performance. When more sectors are used, tighter probability bounds are derived for the sectors and hence the algorithm terminates faster. In particular, the 96-sector partitioning achieves substantial cost reduction (over the basic 6-sector partitioning) for thresholding and ranking queries respectively. Observe that the cost starts converging to its final value with as few as 24 partitions. Figure 13 plots the



(a) thresholding queries, $t = 0.005$ (b) ranking queries, $m = 10$

Fig. 12. Reverse nearest neighbor queries on the SF dataset, varying the number of sectors



(a) thresholding queries vs t (b) ranking queries vs m

Fig. 13. Reverse nearest neighbor queries on the SF dataset, using the 24-sector partitioning

cost of the methods as a function of t and m respectively, when using the 24-sector partitioning. For thresholding queries, the performance gap between the 2D R–tree and other indexes widens as t increases because of the increased pruning power of the advanced indexes. On the other hand, the cost differences among the indexes are not sensitive to the value of m. As with previous queries, AUG prevails.

7 Discussion: Relaxing the Independence Assumption

Our analysis so far assumes that the existential probabilities of objects are independent. This assumption is valid in a large number of applications (e.g., those mentioned in Section 1); hence, our solutions have significant value in practice. However, there are also other applications where the existential probabilities of different objects are correlated. For example, consider a collection of sensors distributed in a forest for detecting wildfire. When a sensor detects smoke, sensors in its neighborhood are likely to sense it as well. A thorough solution in this scenario falls out of the scope of this paper. Nevertheless, in the sequel, we point out the direction towards extending the proposed algorithms and indexing schemes to support correlated existential probabilities.

We now elaborate how to evaluate the thresholding probabilistic NN query using a simple 2D R–tree, in the correlated probability model. Instead of Definition 2, we define the probability of an object x to be the NN of q as: $JP_x = \texttt{JPr}(\ \Gamma(x) \wedge \bigwedge_{x' \in R, x' \neq x, d(q,x') < d(q,x)} (\texttt{NOT}\ \Gamma(x'))\ )$ where JPr denotes the *joint probability* that x exists (i.e., the event $\Gamma(x)$) and all objects x' closer to q do not exist. This probability can be computed from a *Bayesian network* modeling dependent probabilities among the objects. The value $\texttt{JPr}(\ \bigwedge_{x' \in R, x' \neq x, d(q,x') < d(q,x)} (\texttt{NOT}\ \Gamma(x'))\ )$ serves as an upper bound of JP_x, regardless of how the probabilities are correlated. Based on this property, we modify Algo. 2 as follows: (i) we maintain the set S of visited objects, (ii) at Line 7, we insert the object x into S and compute $P^{first} = \texttt{JPr}(\ \bigwedge_{x' \in S} (\texttt{NOT}\ \Gamma(x'))\ )$, (iii) at Line 4, we compute $JP_x = \texttt{JPr}(\ \Gamma(x) \wedge \bigwedge_{x' \in S} (\texttt{NOT}\ \Gamma(x'))\ )$. The above idea works also for spatial skyline (Equation 7, Algo. 6) and reverse nearest neighbor (Equation 14, Algo. 8), after replacing each multiplication by $\wedge$, each E_x by $\Gamma(x)$, each $(1 - E_{x'})$ by $\texttt{NOT}\ \Gamma(x')$, and the final probability by JPr().

Extensions of other R–tree solutions (e.g., 2D augmented trees, 3D trees) generate non-trivial research issues, due to the fact that: (i) the number of possible joint probabilities is enormous (i.e., exponential to the data cardinality), and (ii) it remains unclear how to augment a non-leaf entry to effectively capture the joint probabilities of the objects in its subtree. In the future, we will develop efficient solutions for augmented trees that are applicable for the correlated probability model.

8 Conclusions

In this paper, we presented the interesting problem of evaluating spatial queries for existentially uncertain data. Variants of common spatial queries, like range and nearest neighbor search, have probabilistic versions for this data model. We proposed algorithms for these probabilistic versions and several extensions of spatial access methods (i.e., R–trees) where these algorithms are applied. In addition, we discuss how complex spatial queries such as spatial skyline queries and reverse nearest neighbor queries can be processed in our framework. Finally, we conducted extensive experiments to evaluate the search algorithms and the corresponding spatial indexes. In most of the tested cases, the data structure that performs best is a R–tree, where non-leaf entries are augmented with maximum existential probabilities of the sub-tree they point at. In the future, we plan to study in detail more advanced query types and extend our methods to apply on data that are both existentially and locationally uncertain, as well as results of fuzzy classifiers [1].

Acknowledgments

This work was supported by grant HKU 7149/07E from Hong Kong RGC. The work of Yufei Tao was supported by grants CUHK 1202/06 and CUHK 4161/07 from Hong Kong RGC. A preliminary version of this work appeared in Ref. [20].

References

[1] P. M. Atkinson and N. J. Tate, Eds., *Advances in Remote Sensing and GIS Analysis*. John Wiley & Sons, 1999.

[2] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha, "Updating and Querying Databases that Track Mobile Units," *Distributed and Parallel Databases*, vol. 7, no. 3, pp. 257–387, 1999.

[3] D. Pfoser and C. S. Jensen, "Capturing the Uncertainty of Moving-Object Representations," in *Proc. of SSD*, 1999.

[4] J. Ni, C. V. Ravishankar, and B. Bhanu, "Probabilistic Spatial Database Operations," in *Proc. of SSTD*, 2003.

[5] X. Yu and S. Mehrotra, "Capturing Uncertainty in Spatial Queries over Imprecise Data," in *Proc. of DEXA*, 2003.

[6] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Querying Imprecise Data in Moving Object Environments," *IEEE TKDE*, vol. 16, no. 9, pp. 1112–1127, 2004.

[7] Y. Tao, X. Xiao, and R. Cheng, "Range Search on Multidimensional Uncertain Data," *ACM TODS*, vol. 32, no. 3, p. 15, 2007.

[8] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," in *Proc. of ACM SIGMOD*, 1984.

[9] G. R. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," *ACM TODS*, vol. 24, no. 2, pp. 265–318, 1999.

[10] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter, "Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data," in *Proc. of VLDB*, 2004.

[11] G. Trajcevski, O. Wolfson, F. Zhang, and S. Chamberlain, "The Geometry of Uncertainty in Moving Objects Databases," in *Proc. of EDBT Conf.*, 2002.

[12] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Evaluating Probabilistic Queries over Imprecise Data," in *Proc. of ACM SIGMOD*, 2003.

[13] I. Lazaridis and S. Mehrotra, "Approximate Selection Queries over Imprecise Data," in *Proc. of ICDE*, 2004.

[14] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," in *SIGMOD*, 1990.

[15] M. Sharifzadeh and C. Shahabi, "The Spatial Skyline Queries," in *Proc. of VLDB*, 2006.

[16] I. Stanoi, D. Agrawal, and A. Abbadi, "Reverse Nearest Neighbor Queries for Dynamic Databases," in *SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000.

[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive Skyline Computation in Database Systems," *ACM TODS*, vol. 30, no. 1, pp. 41–82, 2005.

[18] Y. Tao, D. Papadias, and X. Lian, "Reverse kNN Search in Arbitrary Dimensionality," in *Proc. of VLDB*, 2004.

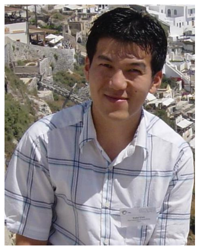
[19] T. Brinkhoff, "A Framework for Generating Network-Based Moving Objects," *GeoInformatica*, vol. 6, no. 2, pp. 153–180, 2002.

[20] X. Dai, M. L. Yiu, N. Mamoulis, Y. Tao, and M. Vaitis, "Probabilistic Spatial Queries on Existentially Uncertain Data," in *Proc. of SSTD*, 2005.

Man Lung Yiu Man Lung Yiu received the Bachelor Degree in Computer Engineering and the PhD Degree in Computer Science from the University of Hong Kong in 2002 and 2006 respectively. He is currently an assistant professor at Department of Computer Science, Aalborg University. His research focuses on the management of complex data, in particular the query processing topics on spatio-temporal data and multidimensional data.

Nikos Mamoulis Nikos Mamoulis received a diploma in Computer Engineering and Informatics in 1995 from the University of Patras, Greece, and a PhD in Computer Science in 2000 from the Hong Kong University of Science and Technology. He is currently an associate professor at the Department of Computer Science, University of Hong Kong, which he joined in 2001. In the past, he has worked as a research and development engineer at the Computer Technology Institute, Patras, Greece and as a post-doctoral researcher at the Centrum voor Wiskunde en Informatica (CWI), the Netherlands. His research focuses on management and mining of complex data types. He has served on the program committees of over 40 international conferences and workshops on data management and data mining. He was the general chair of SSDBM 2008 and a coorganizer of SSTDM 2006. He is an editorial board member for Geoinformatica Journal and a field editor of the Encyclopedia of Geographic Information Systems.

Xiangyuan Dai Xiangyuan Dai received the Bachelor of Engineering Degree from the Department of Computer Science and Technology of the University of Science and Technology of China in 2004. He obtained the MPhil Degree in Computer Science from the University of Hong Kong in 2006. His research interests include query processing problems on spatial data.



Yufei Tao Dr. Tao is engaged in research of database systems. He is particularly interested in index structures and query algorithms on multidimensional data, and has published primarily on temporal databases, spatial databases, and privacy preservation. He received the Hong Kong young scientist award in 2002. He has served the program committees of most prestigious database conferences such as SIGMOD, VLDB, ICDE, and is currently an associate editor of ACM Transactions on Database Systems (TODS). He joined the Chinese University of Hong Kong in September 2006. Before that, he held positions at the Carnegie Mellon University and the City University of Hong Kong. He is a member of the ACM.

Michail Vaitis Michail Vaitis holds an Engineering Diploma (1992) and a Ph.D. degree (2001) in Computer Engineering and Informatics from the University of Patras, Greece. Since 2003 he has been a faculty member of the Department of Geography at the University of the Aegean, Greece. Now he is an assistant professor. In the past, he was working for 5 years at the Research Academic Computer Technology Institute (RACTI), Greece, on hypertext and database systems. His research interests include geographical databases, spatial data infrastructures, geographic hypermedia and geo-spatial semantic web. He is a member of the ACM and the Technical Chamber of Greece.