Probabilistic Spatial Queries on Existentially Uncertain Data[*]

Xiangyuan Dai[1], Man Lung Yiu[1], Nikos Mamoulis[1], Yufei Tao[2], and Michail Vaitis[3]

[1] Department of Computer Science, University of Hong Kong
{xydai, mlyiu2, nikos}@cs.hku.hk
[2] Department of Computer Science, City University of Hong Kong
taoyf@cs.cityu.edu.hk
[3] Department of Geography, University of the Aegean
vaitis@aegean.gr

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 and nearest neighbors and conduct an extensive experimental study, which evaluates the effectiveness of proposed solutions.

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 x; a probability E_x could be assigned to x, reflecting the confidence of x's existence. We call such objects *existentially uncertain*, since uncertainty does not refer to their locations, but 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 various factors (e.g., crime-rate of the caller's district, caller's voice, operator's experience, etc.), for each call we can generate a spatial event associated with a potential emergency and a probability that the emergency is actual. 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 class label takes an existential probability and the sum of all probabilities is 1.

We can naturally define probabilistic versions of spatial queries that apply on collections of existentially uncertain objects. We identify two types of such probabilistic

[*] Supported by grant HKU 7149/03E from Hong Kong RGC.

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 $x \in S$ 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 [20,15,12,21,5] 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. On the other hand, there is no prior work on existentially uncertain spatial data, to our knowledge. In this paper, we fill this gap by proposing indexing and querying techniques for this important class of 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 indexes (i.e., R–trees), we propose search algorithms for probabilistic variants of spatial range queries and nearest neighbor search.
- We show how extensions of R–trees that capture information about existential probabilities in non-leaf node entries can be used to answer probabilistic queries at lower I/O cost.

The rest of the paper is organized as follows. Section 2 provides background on querying spatial objects with rigid or 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 is a comprehensive experimental study for the performance of the proposed methods. Section 6 discusses extensions of our methods for probabilistic versions of complex query types and other (non-spatial) types of existentially uncertain data. Finally, Section 7 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

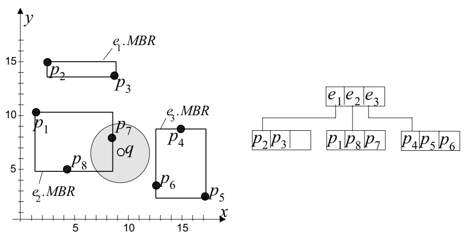


Fig. 1. Spatial queries on R–trees

query types, including spatial range queries, nearest neighbor queries, and spatial joins. Figure 1 shows a collection $R = \{p_1, \dots, p_8\}$ 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. For instance, $e_1.MBR$ does not intersect the query region, thus no object in the subtree pointed by e_1 can contain query results. On the other hand, e_2 is followed by the search algorithm and the points in the corresponding node are examined recursively to find the query result p_7.

A nearest neighbor (NN) query takes as input a query object q and returns the closest object in R to q. For instance, the nearest neighbor of q in Figure 1 is p_7. A popular generalization is the k-NN query, which returns the k closest objects to q, given a positive integer k. NN (and k-NN) queries can be efficiently processed if R is indexed by an R–tree, using the *best-first* (BF) algorithm of [10]. 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 next nearest neighbor (assuming objects with no extent). Otherwise, the node pointed by e is accessed and all entries there are inserted to PQ. The process is repeated, until k objects are found. The BF algorithm is shown [10] to be no worse in terms of I/O than any NN algorithm that applies on the same R–tree. 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 nearest neighbor of q. In Section 4, we will show how BF can be extended to process probabilistic versions of nearest neighbor 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, locations are approximated by probability density functions (PDFs), which integrate GPS error ranges and known moving object velocities. For instance, the uncertainty of a location can be modeled by a 2-dimensional Gaussian function, centered at the coordinates tracked from the GPS.

In [20], objects are assumed to move and send their positions to a centralized server. Each object o knows its last recorded location l_o; given a threshold θ, if the object finds itself θ units away from l_o, it sends an update with its new position. In this way, the server knows that objects are no more than θ away from their recorded locations. Based on this framework, a spatial region U_o (or line segment if the object's movement is constrained on a line) coupled with a PDF models the set of possible locations for each object o. The probability for o to intersect a query range W can be computed by applying the PDF to the spatial intersection of U_o and W. In this way, we can compute the result of a *probabilistic* spatial range query, which includes all pairs $\langle o, P_o \rangle$, where P_o is the probability that object o intersects W, and $P_o > 0$.

Note that the probability of an object to intersect a given query range is independent of that of other objects, a fact that makes range query processing straightforward. On the other hand, the probability of objects to be the nearest neighbor of a reference object q is not independent. Probabilistic nearest neighbor search for locationally uncertain data has been studied in [5]. The algorithm proposed there first computes fast the set of objects with $P_o > 0$, using their (indexed) uncertainty regions U_o only. Then for each object o in this set it integrates its probability to be closer to q than any other object, using the PDFs, over all possible locations of o. This process can be very expensive for arbitrary PDFs, however, [5] shows how to optimize it for basic uncertainty regions and PDFs. [19] indexes 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. A similar approach is followed in [15], where recorded trajectories are converted to sequences of locations connected by elliptical volumes.

[21] 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. In addition, they provide a methodology that sets the maximum precision error given a desired guaranteed uncertainty of the query results. [12] 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. Both the filter and refinement steps of RJ are then adapted to process the join efficiently.

Cheng et al. [4,6] study a problem related to probabilistic spatial range queries. The uncertain data are not spatial, but ordinal (e.g., temperature values recorded from sensors). Due to measurement/sampling errors, an actual value is modeled by a range of possible values and a PDF that captures their probability. [6] indexes such uncertain data for efficient evaluation of probabilistic range queries (e.g., 'find all temperatures between 30°F and 40°F together with their probability to be in this range'). [4] 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.). This work also proposes generic query evaluation techniques and entropy-based measures for quantifying the quality of a probabilistic query result (e.g., how *certain* it is). Finally, [11] 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. The query is initially applied on the uncertain data and based on how accurate the retrieved result is, some of the actual objects may be probed, in order to refine the accuracy of the result and bring its quality to the desired levels.

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 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. Figure 2 shows a collection $R = \{p_1, p_2, \ldots, p_8\}$ of 8 existentially uncertain points. Next to each point label p_i, is its existential probability E_{p_i} enclosed in parentheses (e.g., $E_{p_1} = 0.2$). Given such object collections, we are interested in answering spatial queries that take uncertainty into account. We can easily define probabilistic versions of basic spatial query types:

Definition 1. Let R be a collection of existentially uncertain objects. 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 $P_x > 0$. 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 $P_x > 0$.

In the above definitions 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 2. 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 independent of the locations and confidences of other objects.

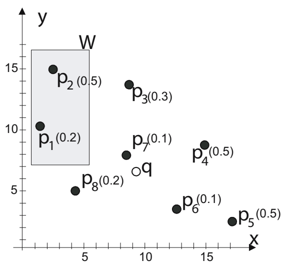


Fig. 2. NN search example

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 2 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 2. Let (τ, P_τ) 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 $P_\tau \geq t$. The *ranking* version of Q takes as additional input a positive integer m and returns the m results with the highest P_τ.

For example, a thresholding range (window) query W with $t = 0.6$ on the objects of Figure 2 returns $\varnothing$, whereas a ranking range query W with $m = 1$ returns $(p_2, 0.5)$.

4 Evaluation of 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.

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.

Nearest Neighbor Search. As discussed, 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. Figure 3 shows an elegant and efficient algorithm that computes the probability P_x of x to be nearest neighbor of q, for all x having $P_x > 0$.

```
Algorithm PNN2D($q$, 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 $R$ (use BF [10]);
4.     $P_x$ := $P^{first} \cdot E_x$;
5.     output ($x$,$P_x$);
6.     $P^{first}$ := $P^{first} \cdot (1 - E_x)$;
```

Fig. 3. Probabilistic NN on a 2D R–tree

Algorithm PNN2D applies best-first NN-search [10] 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 $\prod(1 - E_y)$, 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 2, 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.

Thresholding and ranking. As discussed in Section 3, the user may want to restrict the response set by thresholding or ranking. Figure 4 shows PTNN2D; 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 2 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 (Figure 5), 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 2. PRNN2D progressively 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 PTNN2D($q$, 2D R–tree on $R$, $t$)
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 $R$ (use BF [10]);
4.     $P_x$ := $P^{first} \cdot E_x$;
5.     if $P_x \geq t$ then output ($x$,$P_x$);
6.     $P^{first}$ := $P^{first} \cdot (1 - E_x)$;
```

Fig. 4. Probabilistic NN on a 2D R–tree with thresholding

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

Fig. 5. Probabilistic NN on a 2D R–tree with ranking

4.2 Using Augmented R–Trees to Improve Efficiency

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 [12,6] for locationally uncertain data.

Table 1 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. Figure 6 shows the thresholding NN algorithm 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 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 6–16), we must refine the probability range for x. For this purpose, we pick an entry e in L and load the corresponding node n_e. If n_e is a leaf node, we access the objects e' in n_e. If q is nearer to e' than x, P^{first} is updated with the confidence of e'. Otherwise, its confidence does not affect P^{first} and we enqueue e' to the Best-First Queue. If n_e is a non-leaf node, for each entry $e' \in n_e$, we enqueue e' to the Best-First Queue if $d(q, e') > d(q, x)$, or insert e' into L otherwise. In either case, the probability range of x shrinks. The process is repeated while the range covers t.

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):

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

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

In order to refine the probability range at Line 7 we must pick an entry e in L. We can use several heuristics for determining which e to select: (i) the one with the largest e^{maxE}, (ii) the one with the largest $e^{maxE} \cdot e^{num}$, (iii) the one with the smallest $d(q, e)$, or (iv) by random. By experimentation, we found that heuristic (iii) achieves the best results in most cases.

Table 1. Checking disqualified entries using augmented 2D R–trees

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

```
Algorithm PTNN2Daug($q$, augmented 2D R–tree on $R$, $t$)
1.  $P^{first}$ := 1; /*Prob. of no object before $x$*/
2.  $L$ := ∅; /*List of disqualified non-leaf entries*/
3.  while $P^{first} \geq t$ and more objects in $R$ do
4.      $x$ := next NN of $q$ in $R$ (use BF [10]); /* 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 an entry $e$ in $L$;
8.          remove the entry $e$ from $L$, read node $n_e$ pointed by $e$;
9.          for each entry $e' \in n_e$
10.             if $mindist(q, e') > d(q, x)$ then
11.                 enheap $e'$ in the Best-First heap;
12.             else if $n_e$ is a non-leaf node then
13.                 insert $e'$ into $L$;
14.             else /*$e'$ is an object*/
15.                 $P^{first} := P^{first} \cdot (1 - E_{e'})$;
16.         compute $P_x^{min}$ and $P_x^{max}$ by using $P^{first}$, $L$ and $E_x$;
17.     if $P_x^{min} \geq t$ then output ($x$,$P_x^{min}$,$P_x^{max}$);
18.     $P^{first} := P^{first} \cdot (1 - E_x)$;
```

Fig. 6. Probabilistic NN on a augmented 2D R–tree with thresholding

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.

Let us now show the functionality of the PTNN2Daug algorithm by an example. Consider the augmented R–tree of Figure 7 that indexes the pointset of Figure 2 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 the PTNN2D algorithm accesses all nodes of the tree in this example, whereas PTNN2Daug saves two leaf node accesses.

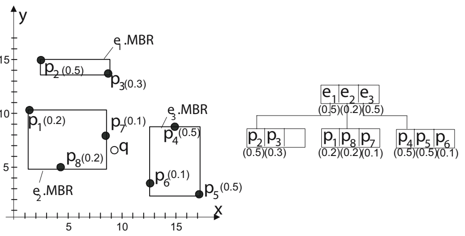


Fig. 7. Example of augmented 2D R–tree

The *ranking* NN algorithm that operates on the augmented R–tree is shown in Figure 8. It 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 21). At Lines 18–19, 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 23–24. 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 $mindist(q,e) > \max\{d(q,o) : o \in H\}$ because such entries cannot be used to refine the probability ranges of the objects in H.

We provide some insight for the space/time complexity of thresholding NN queries for the augmented tree approach. The worst case is that, for all disqualified entries (if any), their child nodes are accessed for refining the probability range of the objects seen. Therefore, the value of k estimated in Section 4.1 can be used as the upper bound in this case. The list L stores disqualified non-leaf entries in the tree and the cost of refining the probability range of an object is directly proportional to the size of L. Thus, the space/time complexity depends on the size of L. As the minimum distance of all entries in L from the query point is at most the distance of the last object seen (in BF-search), the maximum size of L can be estimated by using the value of k. In practice, the average size of L is quite small (10–100) and the space/time required is much less than that in the worst case.

4.3 Evaluation of Probabilistic Queries 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. Since every entry e still stores an e^{maxE}, the methods discussed in Section

```
Algorithm PRNN2Daug($q$, augmented 2D R–tree on $R$, $m$)
1.  $P^{first}$ := 1; /*Prob. of no object before $x$*/
2.  $L$ := $\varnothing$; /*List of disqualified non-leaf 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 $R$ (use BF [10]); /* 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 an entry $e$ in $L$;
10.         remove the entry $e$ from $L$, read node $n_e$ pointed by $e$;
11.         for each entry $e' \in n_e$
12.             if $mindist(q, e') > d(q, x)$ then
13.                 enheap $e'$ on Best-First heap;
14.             else if $n_e$ is a non-leaf node then
15.                 insert $e'$ into $L$;
16.             else /*$e'$ is an object*/
17.                 $P^{first}$ := $P^{first} \cdot (1 - E_{e'})$;
18.                 for each entry $o \in H$ such that $d(q, e') \leq d(q, o)$
19.                     $P_o^{first}$ := $P_o^{first} \cdot (1 - E_{e'})$;
20.         compute $P_x^{min}$ and $P_x^{max}$ by using $P^{first}$, $L$ and $E_x$;
21.     if $P_x^{min} > P^m$ then enheap($H$,($x$,$P_x^{first}$:=$P^{first}$,$P_x^{min}$,$P_x^{max}$));
22.     if $H$ is changed then
23.         recompute, for each $o \in H$, $P_o^{min}$ and $P_o^{max}$ by using $P_o^{first}$, $L$ and $E_o$;
24.         $P^m$ := $m$-th $P^{min}$ in $H$;
25.         remove entries $o$ from $H$ with $P_o^{max} < P^m$;
26.     $P^{first}$ := $P^{first} \cdot (1 - E_x)$;
27. while $|H| > m$ and $|L| > 0$ do
28.     repeat Lines 9–19;
29.     repeat Lines 22–25;
30.     remove $e$ from $L$ with $mindist(q, e) > \max\{d(q, o) : o \in H\}$;
```

Fig. 8. Probabilistic NN on a augmented 2D R–tree with ranking

4.2 for the augmented 2D R–tree can be directly applied for the 3D R–tree. Moreover, we can utilize e^{minE} to derive tighter probability ranges:

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

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

If the exact number e^{num} of object 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.

5 Experimental Evaluation

In this section, we evaluate the efficiency of the proposed techniques. All algorithms were implemented in C++. Experiments were run on a PC with a Pentium 4 CPU of 2.3GHz. In all experiments, the page size was set to 1Kb, unless otherwise stated. No memory buffers are used for caching disk pages between different queries; the number of node accesses directly reflects the I/O cost.

We compare the performances of five indexes and their corresponding algorithms for thresholding and ranking range queries and nearest neighbor search. 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 2D 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 2D 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). When comparing the indexes note that (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). For each experiment, the measured I/O cost is the average cost of 20 queries with the same parameter values (but with different locations randomly chosen from the dataset).

5.1 Description of Data

For our experiments, we used various real datasets of different sizes and object distributions, described in Table 2. The datasets TG and SF are obtained from [2] while 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. By changing θ (default value: 1) we can control the skew.

5.2 Experimental Results

Table 2 shows the performances of the five indexes for thresholding and ranking NN queries on different datasets. We fix $t = 0.002$ for thresholding NN queries and $m = 10$ for ranking NN queries.[2] Observe that the augmented and 3D R–trees perform better than the 2D R–tree, even though they are larger in size. The algorithms of Figures 6 and 8 manage to prune a large number of nodes that do not contain query results, which are

[2] 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.

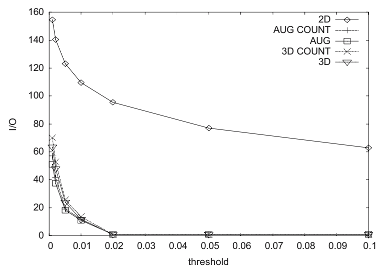
otherwise visited in the simple 2D R–tree index. The cost of 2D R–tree variants (i.e., methods {2D, 2D AUG, 2D AUG COUNT} does not change much with the database size. By the analysis in Section 4.1, the number of points to be examined is independent of the data size for 2D R–trees. The analysis in [18] shows that the cost increases slowly as the data size increases. On the other hand, 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.

Table 2. I/O cost of thresholding/ranking NN on different datasets, $t = 0.002$, $m = 10$

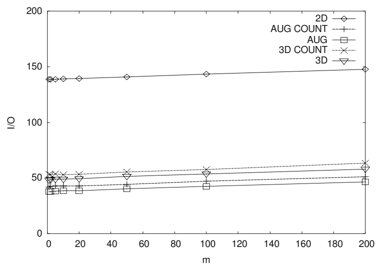
Dataset	**Size**	2D	2D AUG	2D AUG COUNT	3D	3D COUNT
(TG) San Joaquin roads	18623	122.7/116.7	45.2/41.0	37.3/34.2	36.5/32.9	35.4/31.6
(GR) Greece roads	23268	115.3/108.2	40.5/34.8	34.2/29.8	37.0/31.9	32.8/28.5
(LB) Long Beach roads	53145	107.5/100.1	37.3/32.7	32.4/28.2	44.7/41.1	42.0/38.0
(LA) LA streets	131461	135.4/132.3	43.1/42.3	38.1/36.9	48.4/47.4	45.6/45.2
(SF) San Francisco roads	174956	131.5/129.3	42.1/42.4	37.0/37.1	46.0/45.6	41.4/41.7
(TS) Tiger streams	194971	130.7/129.2	40.5/40.4	36.0/35.8	50.6/48.6	45.4/44.7

Figure 9 shows the I/O performance of the indexes for thresholding and ranking queries on the SF dataset. Methods {2D AUG, 2D AUG COUNT, 3D, 3D COUNT} 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 the algorithms of Figures 6 and 8. 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 partitionings as the one using the spatial dimensions only; however, search is performed primarily using the spatial dimensions.

In the next experiment, we compare the performances of the indexes by varying the skewness θ of existential probability distribution of the objects (using the SF dataset). Figure 10 shows the experimental results for this case. We fix $t = 0.002$ and $m = 10$ for thresholding and ranking queries, respectively. The cost of the 2D R–tree increases much faster than the other trees when θ increases. For large θ there are a few, high probabilities around the anchors and the rest are very small. Thus, most points have low existential probabilities and the distances of the results from the query increase, causing an increase in the cost of 2D; only spatial information is used in the algorithms of Figures 4 and 5. On the other hand, the advanced NN algorithms on the augmented and 3D structures manage to prune disqualified directory nodes early.

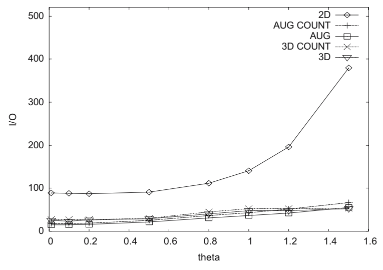


(a) thresholding queries

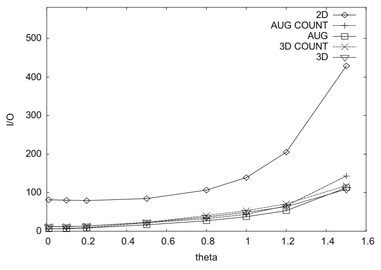


(b) ranking queries

Fig. 9. Queries on the SF dataset, $\theta = 1$



(a) thresholding queries, $t = 0.002$

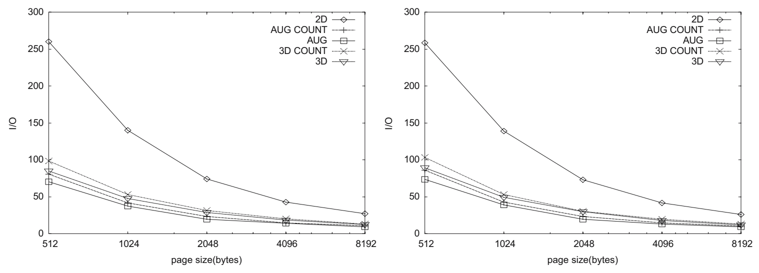


(b) ranking queries, $m = 10$

Fig. 10. Queries on the SF dataset, varying θ

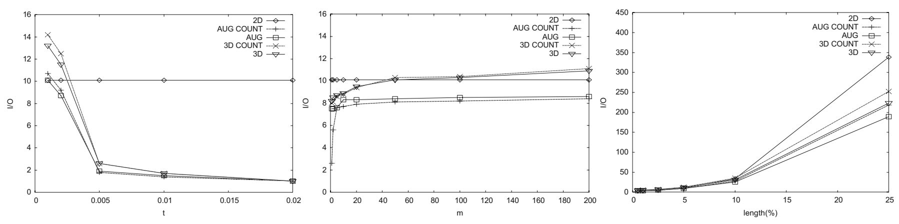
We also study the effect of page size on the performances of the indexes. As Figure 11 shows, the I/O costs of all indexes are inversely proportional to the page size. This is expected, due to the decrease of the number of nodes and heights of the trees.

Finally, we examine the performances of range queries on the indexes, using the SF dataset. For range queries, we use an additional parameter len, which is the extent of the query window in each dimension. The default value of len is set to 5% of values range (domain) at each dimension. Figure 12a and 12b 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 12c shows the cost of thresholding queries as a function of len, at $t = 0.002$. As expected, the costs of all methods increase linearly with len^2. In summary, in most cases of probabilistic NN and range queries, a 2D R–tree with augmented e^{maxE} non-leaf entries achieves the best performance.



(a) thresholding queries, $t = 0.002$ (b) ranking queries, $m = 10$

Fig. 11. Queries on the SF dataset, varying page size



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

Fig. 12. Range queries on the SF dataset

6 Discussion

We have defined and studied in detail probabilistic range and nearest neighbor queries on existentially uncertain spatial data. In this section, we briefly discuss probabilistic versions of other spatial query types and queries on other (non-spatial) existentially uncertain data.

Extended query types. Given two spatial datasets R and S, a *probabilistic spatial join* returns all $(\langle r, s\rangle, P_{r\wedge s})$ pairs, such that $r \in R$, $s \in S$, and r intersects s with probability $P_{r\wedge s} > 0$. We can easily define thresholding and ranking versions of this query. Extending the well-known R–tree join algorithm [3] for probabilistic joins is straightforward, because $P_{r\wedge s}$ depends solely on E_r and E_s (i.e., $P_{r\wedge s} = E_{r\wedge s} = E_r \cdot E_s$) and is independent of the probabilities of other pairs. Given two spatial datasets R and S and a positive integer k, a closest pairs (CP) query [9,7] returns from the Cartesian product $R \times S$ the k $\langle r, s\rangle$ object pairs with the smallest distance. The probabilistic version of a CP query is challenging, due to the interdependence of the existential probabilities of qualifying pairs. The problem can be solved by extending the techniques for probabilistic NN queries and it is left for future work. Other interesting spatial query types for which we can define probabilistic versions are aggregate nearest neighbor queries [13], skyline queries [14], and reverse nearest neighbor search [17].

Spatio-temporal and ordinal data. Our methods can be easily extended for the case, where the objects also carry temporal attributes, i.e., they are spatio-temporal. In this case, the queries also include the time dimension e.g.,'find the most probable nearest neighbor at some moment in the whole past history'. R–trees that index object trajectories (e.g., [16]) can be used by our algorithms for searching. The temporal range may also be restricted to some timestamps or time interval (e.g.,'find the most probable nearest neighbor at some moment in the whole past history'). Finally, although our discussion so far has been on spatial (or spatio-temporal) data, the queries and solutions can be directly refer to ordinal data of any dimensionality (e.g., uncertain transmissions of combinations of measures, like temperature values).

7 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 more complex spatial 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].

References

1. P. M. Atkinson and N. J. Tate, editors. *Advances in Remote Sensing and GIS Analysis*. John Wiley & Sons, 1999.
2. T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.
3. T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *Proc. of ACM SIGMOD*, 1993.
4. R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. of ACM SIGMOD*, 2003.
5. R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE TKDE*, 16(9):1112–1127, 2004.
6. R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proc. of VLDB*, 2004.
7. A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest pair queries in spatial databases. In *Proc. of ACM SIGMOD*, 2000.
8. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. of ACM SIGMOD*, pages 47–57, 1984.
9. G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *Proc. of ACM SIGMOD*, 1998.

10. G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM TODS*, 24(2):265–318, 1999.
11. I. Lazaridis and S. Mehrotra. Approximate selection queries over imprecise data. In *Proc. of ICDE*, pages 140–152, 2004.
12. J. Ni, C. V. Ravishankar, and B. Bhanu. Probabilistic spatial database operations. In *Proc. of SSTD*, 2003.
13. D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *Proc. of ICDE*, 2004.
14. D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proc. of ACM SIGMOD*, 2003.
15. D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. In *Proc. of SSD*, 1999.
16. D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *Proc. of VLDB*, 2000.
17. Y. Tao, D. Papadias, and X. Lian. Reverse knn search in arbitrary dimensionality. In *Proc. of VLDB*, 2004.
18. Y. Tao, J. Zhang, D. Papadias, and N. Mamoulis. An efficient cost model for optimization of nearest neighbor search in low and medium dimensional spaces. *IEEE TKDE*, 16(10):1169–1184, 2004.
19. G. Trajcevski, O. Wolfson, F. Zhang, and S. Chamberlain. The geometry of uncertainty in moving objects databases. In *Proc. of EDBT Conf.*, 2002.
20. O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3):257–387, 1999.
21. X. Yu and S. Mehrotra. Capturing uncertainty in spatial queries over imprecise data. In *Proc. of DEXA*, 2003.