Cost Models for Nearest Neighbor Query Processing over Existentially Uncertain Spatial Data

Elias Frentzos[1], Nikos Pelekis[2], Nikos Giatrakos[3*], Yannis Theodoridis[1]

[1] Department of Informatics, University of Piraeus, Piraeus, Greece
[2] Department of Statistics & Insurance Science, University of Piraeus, Piraeus, Greece
{efrentzo, npelekis, ngiatrak, ytheod}@unipi.gr
[3] Dept of Electronics & Computer Engineering, Technical University of Crete, Crete, Greece
ngiatrakos@softnet.tuc.gr

Abstract. A major challenge posed by real-world applications involving spatial information deals with the uncertainty inherent in the data. One type of uncertainty in spatial objects may come from their existence, which is expressed by a probability accompanying the spatial value of an object reflecting the confidence of the object's existence. A challenging query type over existentially uncertain data is the search of the Nearest Neighbor (NN), as the likelihood of an object to be the NN of the query object does not only depend on its distances from other objects, but also from their existence. In this paper, we present exact and approximate statistical methodologies for supporting cost models for Probabilistic Thresholding NN (PTNN) queries that deal with arbitrarily distributed data points and existential uncertainty, with the aim of appropriate novel histograms, sampling and statistical approximations. Our cost model can be also modified in order to support Probabilistic Ranking NN (PRNN) queries with the aid of sampling. The accuracy of our approaches is exhibited through extensive experimentation on synthetic and real datasets.

Keywords: Spatial Databases, Existential Uncertain Data, Nearest Neighbor Query Processing

1 Introduction

In the literature, two types of uncertainty have gained the interest of the research community, namely the *locational* and the *existential* uncertainty. *Locationally* uncertain are the objects that do exist but their location is uncertain. This kind of uncertainty is described by a probability density function. On the other hand, *existentially* uncertain objects are those that their uncertainty emanates from their existence, and this is expressed by a probability E_x accompanying the spatial value of an object x reflecting the confidence of x's existence. As a motivating example, consider the case where an image processing tool extracts some interesting formations of pixels that may or may not correspond to a predefined type of objects due to low image resolution. Another example is evident in semantically-enriched

[*] Work done during author's PhD studies at the Dept. of Informatics, University of Piraeus

representations of trajectories of moving objects [8], where a point of interest may be part of a semantic trajectory of a user if the latter has been predicted to perform an activity at that place. Existential uncertainty is also natural in the case of fuzzy classification [3], [13].

The related work on existentially uncertain data [3], [13] focuses on two probabilistic versions of several spatial queries. A *thresholding* query returns the objects that satisfy some spatial condition with probability more than a given threshold t, while a *ranking* query returns the objects that satisfy a spatial condition in order of their confidence. Dai et al. [3] proposed search algorithms for the above two types of spatial range and NN queries, where the existentially uncertain data are indexed by 2-dimensional R-trees [7] or appropriate augmented variants of them. In [13] authors also present appropriate algorithms for *Spatial Skyline* [9], and *Reverse Nearest Neighbor* [10] queries, based on the idea of incremental NN search.

In this paper, we focus on the *probabilistic thresholding* (PTNN) and *probabilistic ranking nearest neighbor* (PRNN) queries on existentially uncertain data. The motivation is that, this type of query presents a quite involved search complexity, as the probability of an object to be the NN depends not only on the location, but also on the existential probability of other objects. Moreover, compared to the other operators presented in [13], they are more popular with broader applicability. In [4] we utilized a statistical model in order to estimate the number f of NNs that are to be retrieved from the database so as to be at least CI % confident (i.e. CI is a user-defined confidence, e.g. 99%) that the PTNN search will end without the need to retrieve $n > f$ NNs. The motivation behind this approach is to provide efficient search algorithms, with predetermined cost, and with custom defined certainty (as high as required) of resolution. On the other hand, this is a case which can be only applied to uniform data and existential uncertainty distribution.

We are aware that PTNN2D and PRNN2D are overwhelmed in terms of efficient query processing by the other schemes proposed in [3], which employ augmented versions of R-trees and 3D R-trees. On the other hand, experience has showed that it is very difficult for commercial Spatial Database Management Systems (SDBMS) to support novel proposals, especially when they require altering the data structures used on their engines. Then again, PTNN2D and PRNN2D while not optimal, they can be directly employed with conventional 2D R-trees already implemented in commercial SDBMS. Moreover, the analysis provided in this paper can be easily modified in order to provide similar results that support all schemes of [3].

Outlining the major issues addressed in this paper, our main contributions are:

- Following the assumption of uniformity regarding the existential uncertainty distribution, we present an exact statistical-based analysis for the determination of the *discrete distribution probability density function (dpdf)*, that a PTNN query terminates after having retrieved exactly n objects; exploiting this analysis, we present a cost model for the forecasting of the number of disk page accesses required to process a PTNN query, given that the dataset is indexed by R-trees [7], and the dataset is uniformly distributed in the data space. We further exploit well-known properties of distribution expected values in order to provide an

approximate model for PTNN and PRNN queries,

- We show how to utilize histograms in order to relax the assumption of uniformly distributed data points and existential uncertainty and provide an efficient cost model that predicts the number of disk page accesses required to process PTNN, over arbitrarily distributed data and existential uncertainty. We also utilize random sampling so as to achieve better forecasts, as well as, overpass the problem that is faced regarding an analytical PRNN cost model calculation. Specifically, we alternately apply the results of our statistical analysis and the sampling method, over augmented versions of well-known histograms [1], together with the approach of [11].
- Finally, we report the results of the comprehensive set of experiments, demonstrating the correctness and accuracy of our analysis.

To the best of our knowledge, our work is the first on these topics. The rest of the paper is structured as follows: Section 2 overviews the background work. Section 3 describes the statistical analysis of PTNN queries based on the assumption of uniformly distributed data and existential uncertainty. In Section 4 we present the details of an efficient cost model for PTNN and PRNN queries that supports arbitrary distributions regarding the problem parameters, Section 5 evaluates the accuracy of our model through an extensive experimental study over several datasets, while, Section 6 provides the paper conclusions and some interesting research directions.

2 Background

In this section we introduce the knowledge required for presenting our cost model.

2.1 Probabilistic NN search over spatial data with existential uncertainty

Formally, a PTNN query takes as input a query object q and a probability threshold t, while the data are represented as tuples of the form (x, E_x). As proposed by Dai et al. [3], the 2-dimensional PTNN (PTNN2D) algorithm, illustrated in Figure **1**, iteratively retrieves spatially nearest objects in a Best-First (BF) mode [6], and terminates only after the value of P^{fist} becomes smaller than the given threshold t. The PTNN2D algorithm iteratively calculates the value of P^{first}, which is the probability that no object retrieved before the current object x is the actual NN, according to [3]:

$$P_x^{first} = \prod_{i=1}^{n-1} (1 - E_i), \quad (1)$$

where n-1 is the number of objects that are closer to the query object than the current object x, i.e., the number of objects retrieved from the BF algorithm before x, and E_i their existential uncertainty. Then, the probability that x is the actual NN, is [3]:

$$P_x = E_x \cdot P_x^{first} \quad (2)$$

The intuition behind the PTNN2D algorithm is that once $P^{fist} < t$, we are sure that the subsequent nearest objects, even if they exist with 100% probability, they cannot be the NN of q, so the algorithm can safely terminate. Note also that PTNN2D algorithm utilizes R-tree indexes so as to incrementally retrieve the k-th NN; as such, the R-tree can be replaced by other access method supporting incremental NN search.

```
     Algorithm PTNN2D(q, 2D R-tree on S, t)
1.      P^first=1;
2.      While P^first ≥ t and more objects in S do
3.        x:=next NN of q in S (use BF [7]);
4.        P_x:= P^first · E_x;
5.        If P_x ≥ t then output (x, P_x);
6.        P^first= P^first · (1-E_x);
```

Figure 1. Probabilistic NN on a 2D R-tree with thresholding

```
     Algorithm PRNN2D(q, 2D R-tree on S, m)
1.      P^first:=1;
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 ≥ P^m and more objects in S do
5.        x:=next NN of q in S (use BF [7]);
6.        P_x:= P^first · E_x;
7.
8.        If P_x ≥ P^m then
9.           Update H to include x;
10.          P^m:= m-th probability in H;
          P^first:= P^first · (1-E_x);
```

Figure 2. Probabilistic NN on a 2D R-tree with ranking

Similarly, a ranking spatial query returns the objects, which qualify the spatial predicates of the query, in order of their confidence. Ranking queries can be also thresholded by a parameter m returning thus the m most confident objects. Therefore, a *probability ranking NN* (PRNN) query takes as input a query object q and the number of objects required with the highest probability, over data of the same form, i.e., (x, E_x). Dai et al. [3], also propose the 2-dimensional PRNN (PRNN2D) algorithm, illustrated in Figure 2, which iteratively retrieves spatially nearest objects x in a Best-First (BF) mode iteratively calculating P_x and P^{first} using Eq.1 and Eq.2 respectively. The difference here is that the output is a heap H containing the m most probable NN objects. Therefore the threshold used to terminate is based on P^m which is the P_x of the m-th object in the heap H and the algorithm terminates only after the value of P^{fist} becomes smaller than P^m.

2.2 Cost models for NN search over conventional spatial data

Tao et al. [11] present an efficient cost model for the optimization of NN queries in low and medium-dimensional spaces. They provide a closed formula for the estimation of (a) the average nearest distance D_k from the query point q to its k-th NN and (b) the number of tree nodes whose MBRs intersect the vicinity circle $\Theta(q, D_k)$ with center q and radius D_k, which is equal with the average number of node accesses $NA(k)$ required by an R-tree to retrieve the k-th NN. Specifically, the analysis of [6] shows that the average nearest distance D_k is estimated by:

$$D_k \approx \frac{2}{C_V}\left[1-\sqrt{1-\left(k/N\right)^{1/d}}\right] \qquad (3)$$

where d is the dimensionality, N is the cardinality and C_V is calculated by:

$$C_V = \frac{\sqrt{\pi}}{\left[\Gamma\left(d/2+1\right)\right]^{1/d}} \tag{4}$$

These formulas work only with uniformly distributed data in the search space. On the other hand, real-world data employ arbitrary distributions; as such Tao et al. [11], provide an extension of the above presented model by using MinSkew histograms.

Specifically, the *MinSkew* technique proposed by Acharya et al. [1], is a binary space partitioning (BSP) technique employing the *spatial skew* definition provided in [1]. Each MinSkew Histogram *HS* and can be seen as a set of spatial disjoint buckets B_i that cover the whole data space: $HS = \left\{B_i : \bigcup\left(B_i\right) = S \wedge \bigcap\left(B_i\right) = \varnothing\right\}$ and $B_i = \left\{\left[x_{i,L}, x_{i,U}\right], \left[y_{i,L}, y_{i,U}\right]\right\}$. The main advantage of this technique is that the area grouped together within the same bucket has small spatial skew, i.e., objects are close to uniform distributed inside it; as a result, it is usually assumed that the data distribution inside each bucket B_i is uniform.

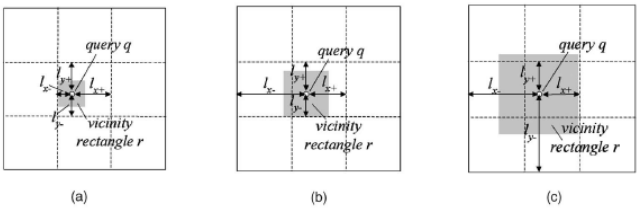


Figure 3. Estimating the "radius" of the vicinity rectangle L_r [11]

[11] provide an algorithm that works over an input histogram *HS* and a query point *q*. The algorithm employs the notion of the *vicinity rectangle* that approximates the *vicinity circle* so as to minimize the number of complex (vicinity) circle- (histogram) rectangle intersection discoveries, and reduce them to less expensive rectangle – rectangle inspections. The algorithm initially determines the distances that *q* needs to travel along each dimension so as to reach the boundaries of each histogram bucket (cf. Figure 3), and stores them in a heap. Then, utilizing the histogram, the algorithm iterates by computing the expected number of points *En* found inside the vicinity rectangle formed by the next distance in the heap; if *En* is smaller than *k*, i.e., the number of requested nearest neighbors, the appropriate vicinity radius is calculated (reduced) according to the formula:

$$L_r \approx \left(\frac{{L_{old}}^d \left(k - En\right) - L^d \left(k - En_{old}\right)}{En_{old} - En} \right)^{1/d} \tag{5}$$

where L is the "diameter" (i.e., side length) of the current vicinity rectangle, while L_{old} and En_{old} are the respective diameter and expected number of points found inside the vicinity rectangle in the previous iteration, respectively. In the case *En* is smaller than *k*, the algorithm proceeds with the next distance in the heap until *En* becomes greater than *k*. Finaly, D_k is obtained by $D_k = L_r / C_v$.

After obtaining D_k the cost model developed for uniform data is applied. Specifically, the query cost in terms of node accesses $NA(k)$ is provided by the following equation:

$$NA(k) = \sum_{i=0}^{\log_f \frac{N}{f}} \left[\frac{N}{f^{i+1}} \cdot \left(\frac{L_i - (L_i/2 + s_i/2)^2}{1 - s_i} \right)^d \right] \quad (6)$$

where N is the cardinality of the dataset, f is the average node fanout, s_i the extent of a level-i node and L_i calculated as a function of D_k and the respective s_i. We have also to note that N is determined based on the *local density* provided by the histogram in the area "near" the query point. The interested reader is cited to [11] for more details.

In our approach, we make use of the techniques proposed in [11], so as to estimate the radius of the *vicinity circle* D_k required to be browsed in order to process PTNN and PRNN queries. Specifically, both PTNN2D and PRNN2D browse the database according to the distance of the query to the dataset objects until a probabilistic criterion is met. Both algorithms perform a number of iterations, continuously requesting in each iteration the next nearest object in an incremental way. The number of iterations is actually equal to the number of nearest objects to the query that have to be retrieved from the database. Consequently, when utilizing an R-tree, as PTNN2D and PRNN2D suggests, and given that the analysis of [11] estimates the number of node accesses $NA(k)$ as a function of D_k and known R-tree parameters, our problem can be reduced to the problem of providing a good estimation of D_k.

Table 1. Table of notations

Notation	Description
x, E_x	A data point and its existential probability
S	A dataset of tuples (x, E_x)
q, t, m	The query object, threshold probability of a PTNN query and number of requested objects of a PRNN query
P^{first}	The probability that no object retrieved before the current object x is the actual NN
P_n^{first}	The probability that no object retrieved before the n-th iteration is the actual NN
P_x	The probability that an object x is the actual NN
$P_{exact}(n)$	The probability that the PTNN algorithm terminates after having retrieved exactly n objects
H	A heap used in the PRNN algorithm
P^m	the P_x of the m-th object in the heap H
$EV(u)$	The expected (average) value of a given variable u
D_k	The nearest distance from the query point q to its k-th NN

3 STATISTICAL ANALYSIS OF PTNN QUERIES

In this section, aiming at a statistical analysis of probabilistic thresholding NN queries, we initially calculate the *expected number* of iterations $EV(n)$ needed for the PTNN2D algorithm to terminate, and then we make use of existing work on cost

models so as to determine the average number of node accesses $NA(EV(n))$ needed in order to process such queries over conventional R-trees. In the sequel, due to the difficulty of extending the exact solution to support such queries over arbitrary distributed data, we present an approximate solution regarding for PTNN queries. We close the section by discussing the extension of this model in the case of PRNN queries. In this first approach we make two assumptions regarding the data distribution:

- *data uniformity assumption*: points x_i are uniformly distributed in the data space,
- *uncertainty uniformity assumption*: the existential uncertainty E_x of all objects in S is uniformly distributed inside the unit interval [0,1],

Both assumptions are relaxed in the subsequent section where an efficient cost model is presented. Table 1 summarizes the notation used in the rest of the paper.

3.1 Exact statistical analysis of PTNN queries

To start with, we provide a lemma from which a cost model for PTNN queries is straightforwardly devised in the case of uniformly distributed data and existential uncertainty. More specifically, the first step towards a cost model for the PTNN2D algorithm 3, is to determine the *dpdf* that the algorithm terminates after exactly n iterations, i.e., the distribution of the number of objects retrieved before P^{first} becomes less than the given threshold t. Formally, we provide the following lemma:

Lemma 1: *The dpdf that the PTNN2D algorithm terminates after exactly n iterations, under the uncertainty uniformity assumption, is given by the following formula*:

$$P_{exact}(n) = \frac{(-1)^{n-1} t \ln(t)^{n-1}}{(n-1)!} \tag{7}$$

where t is the algorithm threshold.

Proof: Our goal is to determine the *dpdf* $P_{exact}(n)$, such that, the algorithm terminates after having retrieved exactly n objects. For this we distinguish between two cases, namely $n = 1$ and $n > 1$. In the first case, the algorithm terminates with a single iteration iff the value of $P_2^{first} = \prod_{i=1}^{1}(1-E_i) = (1-E_1)$ calculated at the end of the first iteration (i.e., line 7 in Figure 1) is less than the given threshold t. Thus, from the uncertainty uniformity assumption, it holds that $P_{exact}(1) = P(1-E_1 \le t) = P(E_1 \ge 1-t) = t$. Given that $-1^0 = (\ln(t))^0 = 0! = 1$ we have proved Lemma 1 in the case where $n = 1$.

In the second case, i.e., $n > 1$, the algorithm terminates iff P_{n+1}^{first}, which is calculated at the end of the n^{th} iteration (i.e., line 7 in Figure 1), becomes less than t after *exactly* n iterations. In other words, we must first determine the conditional probability that P^{first} becomes less than t after n iterations, given also that it must not terminate before reaching n iterations:

$$P_{cond}(n) = P\left(\prod_{i=1}^{n}(1-E_i) \le t \mid \prod_{i=1}^{m}(1-E_i) > t, \forall m \le n-1\right) \quad (8)$$

Then, the total probability that the algorithm terminates after having retrieved exactly n objects can be obtained by multiplying P_{cond} with the probability that the algorithm has not terminated until reaching n iterations:

$$P_{exact}(n) = P_{cond}(n) \cdot P\left(\prod_{i=1}^{m}(1-E_i) > t, \forall m \le n-1\right) \quad (9)$$

Moreover, since $0 \le E_1 \le 1 \Leftrightarrow 0 \le 1 - E_1 \le 1$, it also holds that $(1-E_1) \ge .. \ge \prod_{i=1}^{n-2}(1-E_i) \ge \prod_{i=1}^{n-1}(1-E_i)$. Therefore, given that $\prod_{i=1}^{n-1}(1-E_i) > t$, it stands that $\prod_{i=1}^{m}(1-E_i) > t, \forall m \le n-2$; then, (8) and (9) can be rewritten as follows:

$$P_{cond}(n) = P\left(\prod_{i=1}^{n}(1-E_i) \le t \mid \prod_{i=1}^{n-1}(1-E_i) > t\right) \quad (10)$$

$$P_{exact}(n) = P_{cond}(n) \cdot P\left(\prod_{i=1}^{n-1}(1-E_i) > t\right) \quad (11)$$

Since the values of E_x follow the uniform distribution, the same also stands for 1-E_x; as such the product of the n uniformly distributed values of 1-E_x should follow the *uniform product distribution*, i.e., the distribution of the product of n uniformly distributed uncorrelated variables $x_1, x_2, .. x_n$, with *pdf* given by [12]:

$$P_{x_1 x_2 .. x_n}(u) = \frac{(-1)^{n-1}}{(n-1)!}(\ln u)^{n-1} \quad (12)$$

where u is the product $\prod x_i$.

In our case, we first set as u the product $\prod_{i=1}^{n-1}(1-E_i)$, and then determine the amount of objects $X \in S$, such as $\prod_{i=1}^{n}(1-E_i) = (1-E_n) \cdot u \le t$ which leads to:

$$(1-E_n) \le t/u \quad (13)$$

Given that $(1-E_n)$ is also uniformly distributed, it should hold that the amount of objects fulfilling the above expression V_n is

$$V_n = t/u \quad (14)$$

Known the above, we can calculate the probability $P_{cond}(n)$ by summing up (i.e., integrating) the amount of objects V_n for each value of u, weighted by the value of the distribution of u, and divided by the respective sum (i.e., integral) of the distribution

of u. Moreover, since it is known that $u = \prod_{i=1}^{n-1}(1-E_i) \geq t$, the above integrals should involve only the values of u between t and 1. Summarizing:

$$P_{cond}(n) = \frac{\int_t^1 \frac{t}{u} P_{n-1}(u)\,du}{\int_t^1 P_{n-1}(u)\,du} \tag{15}$$

Moreover, the total probability that the algorithm has not been terminated until reaching n iterations (i.e, $\prod_{i=1}^{n-1}(1-E_i) > t$), can be easily calculated, using the pdf of the product of n-1 uniformly distributed variables:

$$P\left(\prod_{i=1}^{n-1}(1-E_i) > t\right) = \int_t^1 P_{n-1}(u)du \tag{16}$$

Finally, by substituting (15) and (16) into (11) and performing the necessary calculations, we have proved Lemma 1 in the case where $n > 1$■

Lemma 1 provides us with the dpdf that the algorithm terminates after exactly n iterations. The dpdf expressed by (7) is a closed formula, since it involves only the logarithm of the threshold t and the factorial of n. Obviously, the density of the probability obtained from (7) for several values of n, is dominated by the factorial of n-1; as such, it is expected that as the number of iterations grows, the respective probability density will tend to zero very fast. In the sequel we present a corollary derived from Lemma 1, which helps us to determine the cost model for PTNN queries over existentially uncertain data that follow the uncertainty uniformity assumption.

Corollary 1: *The expected number of iterations in the execution of the PTNN2D algorithm, under the uncertainty uniformity assumption, is*:

$$EV(n) = 1 - \ln(t) \tag{17}$$

Proof: The expected number of iterations needed from the PTNN2D algorithm to terminate is actually the mean value of (7) for each $n \in$ N. As such, $EV(n)$ can be calculated by averaging the dpdf $P_{cond}(n)$ over all possible values of n.

$$EV(n) = \sum_{i=1}^{\infty} \frac{(-1)^{i-1} t \ln(t)^{i-1}}{(i-1)!} \cdot i \tag{18}$$

Equation (18) cannot be straightforwardly evaluated since it involves infinity; however, we may calculate its limit:

$$\sum_{i=1}^{\infty} \frac{(-1)^{i-1} t \ln(t)^{i-1}}{(i-1)!} \cdot i = \lim_{n\to\infty} \sum_{i=1}^{n} \frac{(-1)^{i-1} t \ln(t)^{i-1}}{(i-1)!} \cdot i \tag{19}$$

which after the necessary calculations turns into:

$$\sum_{i=1}^{\infty} \frac{(-1)^{i-1} t \ln(t)^{i-1}}{(i-1)!} \cdot i = 1 - \ln(t) \tag{20}$$

Finally, by substituting (20) into (18) we have proved corollary 1■

Obviously, the expected number of iterations $EV(n)$ needed from the PTNN2D in order to terminate, is equal with the number of NNs needed to be retrieved from an existentially uncertain spatial database queried with a query point and a given threshold t. Thus, we may employ the analysis presented in [11], so as to estimate the average radius D_k on which the $EV(n)$-th NN will be found, under the data uniformity assumption. Apparently, this model can be applied in our case where the dimensionality d is 2 and the value of $\Gamma(d/2+1)$ is $\Gamma(2/2+1)=1$; then, by substituting the expected number of n produced by (17) into the number of k NNs requested, (3) can be rewritten as follows:

$$D_k \approx \frac{2}{\sqrt{\pi}} \left[1 - \sqrt{1 - \sqrt{\frac{1 - \ln(t)}{N}}} \right] \tag{21}$$

From this point on, the analysis of [6] that estimates the number of node accesses $NA(EV(n))$ in the case of uniform data distribution (which is identical with our *data uniformity assumption*) remains unaffected; the single modification to be made is to calculate D_k using (21) instead of (3), and then apply Eq.(6) accordingly. Concluding, the cost model for PTNN queries over existentially uncertain data that follow both the uncertainty uniformity and the data uniformity assumptions is based on (21), which estimates the distance from the query point that has to be browsed from the database so as to answer such a query; then, the required node accesses $NA(EV(n))$ can be straightforwardly estimated by replacing the D_k into the analysis of [11].

3.2 Approximate statistical analysis of PTNN queries

Unfortunately, the extension of the above-described theoretical model in the case of arbitrary distributed data is not straightforward at all. Histograms widely used in order to provide statistical estimations in DBMS, pose insuperable problems to this extension due to their discrete nature. Specifically, given the simplest case where a 1-dimensional histogram $HS = \{[0, E_1], [E_1, E_2], .., [E_{m-1}, 1]\}$ is used to describe the existential uncertainty distribution in a given point in space, the distribution of the exact number of iterations following the methodology of Lemma 1 would be given as a function defined in m^n parts. This is due to the fact that the resulted distribution would be calculated as the product of n sets containing m spaces each. Obviously, such as an approach is not practical. On the other hand, we may provide an approximate solution which utilizes the notion of the expected value of the probability of a random object retrieved in the n-th iteration to be included in the query results. More formally we provide the following Lemma 2.

Lemma 2: *The number of iterations n required for the expected value P^{first} of the PTNN2D algorithm to become equal to the threshold t, is given by*:

$$EV\left(P_n^{first}\right) = t \Rightarrow n = 1 + Log_{1-EV(E_x)}(t) \tag{22}$$

where $EV(E_x)$ is the expected value of existential uncertainty E_x of a random x in S.

Proof: Our main objective is to express $EV(P^{first})$ as a function of known values. Towards this goal, we know that the expected (mean) value of a random variable produced as the product of two other random variables, is equal with the product of the expected value of the two variables. Formally, given two random variables *u* and *v* the following stands:

$$EV(u \cdot v) = EV(u) \cdot EV(v) \tag{23}$$

From the definition of P^{first} (1) and from (23) we have that the expected value $EV(P^{first})$ after *n* iterations is:

$$EV\left(P_n^{first}\right) = \left(1 - EV(E_x)\right)^{n-1} \tag{24}$$

Now, in order for (24) to become equal to *t* we have:

$$\left(1 - EV(E_x)\right)^{n-1} = t \Rightarrow n - 1 = Log_{1-EV(E_x)}(t)$$

which proves Lemma 2. ■

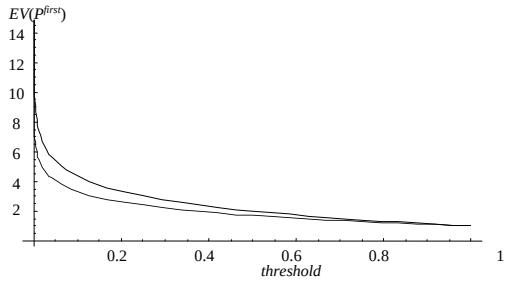


Figure 4. Estimating the number of iterations of PTNN2D over uniform data by exact (solid line) and approximate solutions (doted line)

It is clear that Lemma 2 can be utilized in order to provide an approximate estimation for the number of iterations needed by the PTNN2D algorithm in order to terminate. It certainly does not provide the exact value of $EV(n)$ as Corollary 1 does, however, it provides strong evidence that the algorithm may terminate when *n* becomes greater than the value provided. What is more, Lemma 2 does not utilize the uncertainty uniformity assumption; as such it can be applied over data with arbitrary distributed existential uncertainty, relaxing therefore our uncertainty uniformity assumption. Also interestingly when employing lemma 2 under the uncertainty uniformity assumption, where $EV(E_x)$=0.5, (22) results in $n = 1 + Log_{0.5}(t)$. A comparison between this result and (17) is given in Figure 4. It is clear that the approximate solution always overestimates *n*, with its difference from the exact solution increasing when the value of *t* becomes less than 0.2 (and the number of iterations increases above 3).

3.3 Discussion on PRNN queries

One may suggest that Lemma 1 and its Corollary 1 can be easily extended to cover the case of PRNN queries, processed by the PRNN2D algorithm, since their main difference is on their termination condition, i.e., the continuous evolving value of P^m employed instead of the constant value of t. Towards this goal, we could utilize the fact that the expected (mean) value of a random variable produced as the product of two independent random variables, is equal with the product of the expected value of the two variables. However, the calculation of a theoretical value for P^m is a very hard task which involves order statistics [2]. Specifically, even in the – simplest - case of m=2, the expected value of the m-th P_x inside H, is determined by distinguishing between two cases regarding the order of values in H, i.e., $\{P_1, P_2\}$, $\{P_2, P_1\}$:

- In the case where $E_1 \geq 0.5$, since $P_1^{first} = 1$, it follows that $P_1 \geq 0.5$, $P_2^{first} = (1-E_1) < 0.5$ and $P_2 < 0.5$. Therefore the order of P_i inside H will be $\{P_1, P_2\}$. Now, given from the uncertainty uniformity assumption that $EV(P_1)=EV(E_1)=0.5$ and $EV\left(P_2\right) = EV\left(P_2^{first} E_2\right) = EV\left(P_2^{first}\right) EV\left(E_2\right) = 0.25 \cdot 0.5 = 0.125$, $H=\{0.5,0.125\}$ and $EV(P^m)=0.125$.
- In the case where $E_1<0.5$, it follows that $P_1<0.5$ and $P_2^{first} = (1-E_1) > 0.5$; therefore for $P_2>P_1$ it should hold that $E_2 > E_1 / (1-E_1)$, and $EV(P^m)=EV(P_1)=0.25$. However, in the case of $E_2 < E_1 / (1-E_1)$ it follows that $P_1>P_2$, and $EV(P^m)=EV(P_2)$, a value that cannot be straightforwardly calculated.

It is clear that the calculation of $EV(P^m)$ for arbitrary values of m is a very demanding task. However, the usefulness of such a calculation can be argued, since by approximate sampling methods as those described in the next section, we may obtain good estimates of the expected number of iterations.

4 A COST MODEL FOR PTNN AND PRNN QUERIES

In the exact analysis of section 3 we assumed that both data points, and their existential uncertainty, are uniformly distributed in their space. In this section, we relax both assumptions and apply our approach to arbitrarily distributed data with the employment of augmented *histograms*. The presence of the histogram is to provide (a) local estimations regarding the density of existentially uncertain spatial objects in the neighborhood of the query point, and, (b) statistics that can be used in order to estimate the number of iterations needed from the PTNN2D algorithm to terminate.

4.1 Augmented histograms

The proposal of Acharya et al. [1], may be easily extended in order to support our scenario of existentially uncertain spatial objects, by augmenting it in a third

dimension describing the existential uncertainty. Formally, the proposed histogram is $HS = \{B_i : \cup(B_i) = S \times [0,1] \wedge \cap(B_i) = \varnothing\}$ and $B_i = \{[x_{i,L}, x_{i,U}], [y_{i,L}, y_{i,U}], [E_{i,L}, E_{i,U}]\}$, and the data distribution inside each 3D bucket B_i is considered as uniform. The histogram is created using the methodology of [1] by simply treating the existential uncertainty dimension as an additional spatial dimension.

4.2 A sampling-based approximation method

The above proposed histogram, besides its conventional use, i.e., to estimate the local density of data, it can be used in order to produce a 1D histogram of the data points' existential uncertainty distribution in the area "near" the query point. Subsequently, random values of existential uncertainty can be produced following the local distribution provided by the 1D histogram, and then, used to simulate the behavior of the PTNN2D algorithm. The basic dilemma that is posed towards a good estimation following such a technique is to provide an efficient termination condition for the sampling process. This condition can be provided by computing the standard deviation of the sampled mean value:

$$\sigma_{mean} = \frac{\sigma}{\sqrt{N}} \tag{25}$$

where σ is the sample standard deviation and N the sample size. Then, by using the hypothesis that n follows the normal distribution, and a confidence interval CI=95%, the expected number of iterations $EV(n)$ is:

$$\bar{n} - 1.96\frac{\sigma_n}{\sqrt{N}} \le EV(n) \le \bar{n} + 1.96\frac{\sigma_n}{\sqrt{N}} \tag{26}$$

where N is the number of observations (number of PTNN2D simulation runs), σ_n the (computed so far) standard deviation of n and 1.96 is the approximate value of the 97.5 percentile point of the normal distribution, used in the construction of approximate 95% confidence interval.

Figure 5 illustrates the algorithm *SamplePTNN2D* which summarizes the proposed methodology regarding the estimation of the number of iterations of the PTNN2D algorithm using sampling. The algorithm utilizes a 1D histogram *HS* describing the existential uncertainty distribution in the local query area, the algorithm's threshold t, and the precision p (e.g., 5%) of the expected value of n. The precision is used instead of an absolute value of standard deviation in order to compute it as a percentage of the calculated mean value. The algorithm begins by instantiating *km*, i.e. the calculated mean of the number of iterations needed by the PTNN2D algorithm to terminate, and *kt*, which is the standard deviation of the calculated mean. After that (lines 4-6), the algorithm instantiates P^{first}, N (i.e. the number of PTNN2D simulations) and n (i.e., the number of iterations of the PTNN2D algorithm in the current run). In lines 7-11 the PTNN2D algorithm is simulated, and the number of iterations n in its current run is determined. The histogram *HS* is used in line 8 in order to produce random values

based on the local area's existential uncertainty distribution. After simulation, the algorithm calculates the new mean value of the number of iterations required in every run, as well as the mean's standard deviation (line 13). The algorithm eventually terminates and returns the calculated mean value of iterations when there is 95% probability (which is included in the area $1.96 \times kt$) that the mean differs by at most p regarding its accurate value.

```
     Algorithm SamplePTNN2D(HS 1D Histogram, threshold t, precision p)
 1.      km:=0; //calculated mean iterations
 2.      kt:=+∞; // calculated stdev of mean iterations
 3.      While p<km<1.96<kt do
 4.        N:=N+1; //num of runs
 5.        P^first:=1;
 6.        n:=0; //run's iterations
 7.        While P^first>t do // simulate PTNN2D
 8.          n:=n+1;
 9.          E_x:=ProduceRandomValue(HS);
10.          P^first:= P^first*(1-E_x);
11.        End While;
12.        km:=Mean(n);
13.        kt:=Stdev(n)/Sqrt(N);
14.      End While;
15.      Return km;
```

Figure 5. Sampling algorithm for estimating the number of iterations of PTNN2D

```
     Algorithm SamplePRNN2D(HS 1D Histogram,# objects m, precision p)
 1.      km:=0; //calculated mean iterations
 2.      kt:=+∞; // calculated stdev of mean iterations
 3.      While p<km<1.96<kt do
 4.        P^first:=1;
 5.        N:=N+1; //num of runs
 6.        H:=∅;
 7.        P^m:=+∞;
 8.        n:=0; //run's iterations
 9.        While P^first>P^m do //simulate PRNN2D
10.          n:=n+1;
11.          E_x:=ProduceRandomValue(HS);
12.          Px:=P^first*E_x;
13.          If P_x> P^m then
14.             Update H to include x;
15.             P^m:= m-th probability in H;
16.          P^first:= P^first*(1-E_x);
17.        End While;
18.        km:=Mean(k);
19.        kt:=Stdev(k)/Sqrt(k);
20.      End While;
21.      Return km;
```

Figure 6. Sampling algorithm for estimating the number of iterations of PRNN2D

Interestingly, the method of sampling can be directly applied with limited only modifications in the case of PRNN queries. The respective *SamplePRNN2D* algorithm is illustrated in Figure 6. The algorithm utilizes the same ideas as *SamplePTNN2D* with the only difference that PRNN2D is simulated (instead of PTNN2D) between lines 9-17, with P^m calculated and eventually tested so as to be used as a termination condition. This observation enables us to introduce a cost model for PRNN queries as well, as described in the following section.

4.3 An effective cost model for PTNN and PRNN queries

In this section we present an effective cost model for PTNN queries that works over arbitrary distributed spatial data with existential uncertainty. The proposed cost model is calculated using the algorithm presented in Figure 7, which employs several ideas presented in [12]. In particular, algorithm *EstimateThresholdDk* takes as input a simple spatial histogram, an augmented histogram, a query point q and a threshold t, and estimates the radius D_k of the vicinity circle that has to be browsed by the PTNN2D algorithm. The radius D_k is then applied over Eq.(6) so as to estimate the number of node accesses NA that are needed in order to answer the query. The algorithm initially (lines 2-4) determines the critical vicinity rectangle "radiuses", i.e., the rectangle's half-side, on which the object's density changes. These radiuses are determined by simply calculating the distance that q needs to travel along each axis so as to reach each bucket's boundaries. After their calculation, these values are inserted into a min-heap so as to be used in incremental order.

```
Algorithm EstimateThresholdDk(Histogram HS, Augmented Histogram AHS, point q, threshold t)
 1.  HP = new min-Heap
 2.  for each bucket B in HS;
 3.      Determine the radius that is needed for a rectangle
         with center q to reach B and add it to HP
 4.  end for
 5.  EnOld=:0; lOld:=0;
 6.  While true do // algorithm eventually terminates at line 13
 7.     l=:HP.pop;
 8.     En=:HS.Density(q,l)*(4*l*l);//calculate # objects inside rec
 9.     m=:AHS.MeanValue(q,l)
10.     k=:Log(t)/Log(1-m)+1
11.     If k<En then
12.        Compute Lr by equation (5)
13.        Return Lr/Sqrt(PI)
14.     Else
15.        lOld=l;EnOld=En;
16.     End if
17.  End while
```

Figure 7. Algorithm *EstimatedThresholdDk* for computing D_k

Then, the algorithm iteratively retrieves candidate critical distances l on which the vicinity rectangle's density is changed (via the min heap), and calculates (line 8) the expected number of objects En found inside it, by simply multiplying the local density produced by HS by the area of the respective vicinity rectangle. It also determines in line 9 via the augmented histogram, the mean value m of the existential uncertainty of objects found inside the vicinity rectangle, using as input the query point q as well as the radius l of the vicinity rectangle. The value m is eventually used in line 10 to calculate the (approximated) number of nearest neighbors k that must be retrieved in order for the PTNN2D algorithm to terminate. Then, in line 11, the values of k and En are compared, in order to determine whether the number of required nearest neighbors k is less than the objects contained inside the (so far calculated) vicinity rectangle. If it is not so, the algorithm stores l in $lOld$ and En in $EnOld$ to be used by Eq.5 in a subsequent iteration, and performs another iteration, so as to use a greater critical radius l (which are stored in the minheap). Eventually, the algorithm terminates by computing Lr via Eq.(5), and returning Dk (lines 12-13) when the iteratively increasing radius of the vicinity rectangle, produces an approximate number objects contained inside the respective vicinity rectangle, greater than k.

The previously presented algorithm 2 provides a good approximation of the number of objects that have to be retrieved from the database in order for the PTNN2D to terminate. However, this can be also achieved via sampling, as described in the previous section. Specifically, lines 9-10 of the *EstimatedThresholdDk* can be replaced with (a) the calculation of a 1-dimensional histogram, *AHS*, and (b) algorithm *SamplePTNN* (cf. Figure 5) that estimated *k* based on a 1-dimensional histogram of existential uncertainty. Similarly, by replacing lines 9-10 with the calculation of the 1D histogram and the algorithm *SampleRTNN* used to estimate the number of iterations of PRNN2D, algorithm *EstimatedThresholdDk* may be also used as a cost model for PRNN search.

Summarizing, the proposed cost model the bases on the *EstimatedThresholdDk* algorithm, can be used for estimating the radius of the vicinity circle, used for both PTNN and PRNN queries.

5 Experimental Study

Our experimental study is based on real point datasets. In particular, as in [13], we used the San Francisco roads' dataset (SF) dataset. Due to the lack of a real spatial dataset with objects having existential probabilities, we generated probabilities for the objects, using the following methodology. As [13] suggest, we first generated K = 10 anchor points on the map in positions of high data density. These points model locations around which there is large certainty for the existence of data. For each point *x* of the dataset, we find the closest anchor and we assign an existential probability inversely proportional to its distance from it. Thus, the distribution of probabilities around the anchors is a Zipfian one. The probabilities are normalized w.r.t. the maximum probability.

We conducted our experiments on a Windows XP workstation with AMD Athlon II X4 640 3GHz processor CPU. All evaluated methods were implemented using the .NET framework. Two statistical measures were used so as to demonstrate the behavior of our model. The *average radius of the vicinity circle* $\overline{D}$, the *average estimated radius of the vicinity circle* $\overline{D_e}$, and the *average error in the estimation of the vicinity circle* $\overline{DS}$. Formally, these measures are defined as:

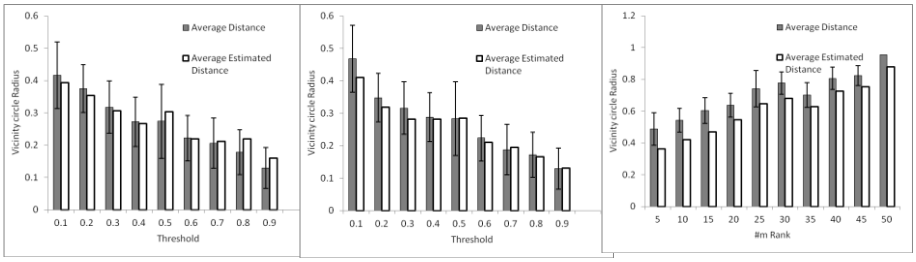
$$\overline{D} = \frac{1}{n}\sum_{i=1..n} D_i \text{ , } \overline{D_e} = \frac{1}{n}\sum_{i=1..n} D_i^e \text{ , and, } \overline{DS} = \frac{1}{n}\sum_{i=1..n} \left| D_i - D_i^e \right|$$

where *n* is the number of executed queries, D_i the actual distance of the vicinity circle from the *i-th* query, and D_i^e the estimated radius of the vicinity circle via the respective cost model. We distinguish between, $\overline{D}$ and $\overline{DS}$, in order to disclose the details of the behavior of our model, as will be shown in the following experiments.

In order to test the accuracy of the proposed model, we performed 500 PTNN queries

in locations selected driven by the datadet density, under various threshold values, and counted the actual number of iterations that the algorithm performed; we also compared the values gathered from the experiment with the one calculated using our model. The corresponding results are illustrated in Figure 8(a) and (b), regarding the PTNN2D algorithm with estimates gathered via Lemma 2 and sampling, respectively. It is clear that the values $\bar{D}$ and D_i^e displayed in both bars (actual and estimated vicinity circle radiuses) are almost identical, meaning that the estimation gathered by our model is very accurate, with an error that never exceeds 12%, regarding the average number of iterations for all 500 queries. Moreover, the mean deviation $\overline{DS}$ (i.e., the average unsigned error of the estimation in each individual query), illustrated by the error bars, is between 20% and 50% in all experimental settings, increasing with the threshold. This is actually an expected result since the increase of threshold results in decreasing the number of iterations of the PTNN2D algorithm, which leads to the deviation growth. A comparison between the two alternative ways of estimation, i.e., Lemma 2 and sampling, results that the latter performs slightly better.

Similar results are exposed regarding the PRNN2D algorithm, illustrated in Figure 8(c), where our estimations are once again accurate, with an error that never exceeds 15%, except of the case where small cardinalities of objects are requested, i.e., smaller than 10, where the error reaches 25%.



(a) (b) (c)

Figure 8. Average actual and estimated search radius of the PTNN2D algorithm scaling the threshold using (a) mean probability, (b) Sampling, and (c) the PRNN2D algorithm scaling the number of objects requested

6 Conclusions and Future Work

In this paper, we have worked with the problem of performing *probabilistic thresholding nearest neighbor* and *probabilistic ranking nearest neighbor* queries over existentially uncertain spatial point datasets [3],[13]. Following a statistical approach, we estimate the average number of the nearest neighbors required for processing PTNN queries as a function of the threshold *t*, and then, utilizing existing approaches [6] we propose a cost model for such queries. We have also provided approximate solutions for the same problem, which turn out to be applicable over arbitrary distributed data. Our experimental study proves the effectiveness and efficiency of the proposed techniques. There are numerous interesting research directions arising from this work, including the application of our model in data

spaces of higher dimensionality, its extension in order to support reverse nearest neighbor, and spatial skyline queries according to [13], as well as objects with time-varying existential uncertainty.

7 Acknowledgements

Elias Frentzos was supported by the Greek States Scholarships foundation. Nikos Pelekis and Yannis Theodoridis were supported by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n°270833, ICT project DATASIM (www.datasim-fp7.eu).

References

1. Acharya, S., Poosala, V., and Ramaswamy, S.: Selectivity Estimation in Spatial Databases. In Proceedings of the ACM SIGMOD Int'l Conference on Management of Data (SIGMOD'99) (1999) pp.13-24
2. Balakrishnan, N. and Rao, C. R. (Eds.): Order Statistics: Applications. Amsterdam, Netherlands: Elsevier (1998)
3. Dai, X., Yiu, M.L., Mamoulis, N., Tao, Y., and Vaitis, M.: Probabilistic Spatial Queries on Existentially Uncertain Data. In Proceedings of the Int'l Symp. Spatial and Temporal Databases (SSTD'05) (2005) pp.254-272
4. Frentzos, E., Pelekis, N., and Theodoridis, Y.: Cost Models and Efficient Algorithms on Existentially Uncertain Spatial Data. In Proceedings of the 12th Panhellenic Conference in Informatics (PCI'08) Samos, Greece (2008)
5. Frentzos, E. Gratsias, K. and Theodoridis Y.: On the Effect of Location Uncertainty in Spatial Querying. 21(3) IEEE Trans. Knowl. Data Eng. (2009) 366-383
6. Hjaltason, G., and Samet, H.: Distance Browsing in Spatial Databases, ACM Transactions in Database Systems. vol. 24(2) (1999) pp. 265-318.
7. Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A.N., and Theodoridis, Y.: Rtrees: Theory and Applications. Springer (2005)
8. Parent, C., Spaccapietra, S., Renso, C., Andrienko, G., Andrienko, N., Bogorny, V., Damiani, M. L., Gkoulalas-Divanis, A., Macedo, J., Pelekis, N., Theodoridis, Y., and Yan, Z.: Semantic Trajectories Modeling and Analysis. ACM Computing Surveys, (2013)
9. Sharifzadeh M., and Shahabi, C.: The Spatial Skyline Queries. In Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB), Seoul, Korea (2006)
10. Stanoi, I., Agrawal, D., and Abbadi, A.: Reverse Nearest Neighbor Queries for Dynamic Databases. In Proceedings of the SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (2000)
11. Tao, Y., Zhang, J., Papadias, D., and Mamoulis, N.: An Efficient Cost Model for Optimization of Nearest Neighbor Search in Low and Medium Dimensional Spaces. Vol.16, no.10, IEEE Trans. Knowledge and Data Eng. (2004) pp.1169-1184.
12. Weisstein, Eric W.: Uniform Product Distribution. From MathWorld, A Wolfram Web Resource.
13. Yiu, M., Mamoulis, N., Dai, X., Tao, Y., Vaitis, M.: Efficient Evaluation of Probabilistic Advanced Spatial Queries on Existentially Uncertain Data", Vol. 21(1) IEEE Trans. Knowledge and Data Eng (2009)