Efficient Search for the Top-k Probable Nearest Neighbors in Uncertain Databases

George Beskales Mohamed A. Soliman Ihab F. Ilyas
David R. Cheriton School of Computer Science, University of Waterloo
{gbeskale,m2ali,ilyas}@cs.uwaterloo.ca

ABSTRACT

Uncertainty pervades many domains in our lives. Current real-life applications, e.g., location tracking using GPS devices or cell phones, multimedia feature extraction, and sensor data management, deal with different kinds of uncertainty. Finding the nearest neighbor objects to a given query point is an important query type in these applications.

In this paper, we study the problem of finding objects with the highest marginal probability of being the nearest neighbors to a query object. We adopt a general uncertainty model allowing for data and query uncertainty. Under this model, we define new query semantics, and provide several efficient evaluation algorithms. We analyze the cost factors involved in query evaluation, and present novel techniques to address the trade-offs among these factors. We give multiple extensions to our techniques including handling dependencies among data objects, and answering threshold queries. We conduct an extensive experimental study to evaluate our techniques on both real and synthetic data.

1. INTRODUCTION

Nearest neighbor (NN) queries are widely used in many applications including geographical information systems [11], and similarity search [30]. The problem can be defined as follows: 'given a number of objects, find the nearest object(s) to a given query point, based on a distance metric.' A large body of research studies NN queries for precise (certain) data, e.g., [14, 25]. Many of these works focus on using index structures for efficient computation.

Applications in domains that involve uncertainty such as location tracking [9] and sensor data management [21] have motivated the need to support new query types [27], uncertainty models [26], and query processing techniques [24]. Queries that involve ranking objects under uncertainty based on a *scoring* criterion, e.g., top-k and NN queries, are important to a wide range of these applications, as illustrated by the next examples.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 978-1-60558-305-1/08/08

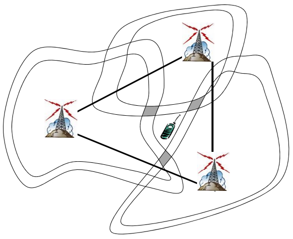


Figure 1: Using Cell Towers to Locate Mobile Users

Example 1.1. *Cell towers are used to locate mobile users using techniques similar to [31]. In Figure 1, the signals communicated between a mobile user and the three nearest towers define contour lines of user's possible locations. Each contour line gives a uniform distribution of user's location. Hence, a user is more likely to be located at the intersection of contour lines (the shaded areas in Figure 1).*

In Example 1.1, assume the query point is an accident location, where the data objects are the mobile users. Each data object has an uncertain location attribute. In these settings, finding the most probable nearest witnesses/police cars to accident location is an important query. The next examples give other variations of the same problem.

Example 1.2. *In location-based services, a user may request the locations of the nearest gas stations. To protect user's privacy, an area that encloses user's actual location may be used as the query object. Gas stations (the data objects) have deterministic locations and thus they can be modeled as deterministic points.*

Example 1.3. *In face recognition systems, e.g., [19], a person is identified by computing the similarity between a query object (description of person's face), and data objects (descriptions of faces stored in a database). Each description is a vector of features such as the relative locations of face elements, and their sizes. Due to imprecision of feature extraction, such vectors usually involve uncertainty. Each feature is thus modeled as a probability distribution on possible values. Finding persons that are most similar to the person in question involves a nearest neighbor search with uncertainty in both data and query objects.*

A related issue to the previous examples is the uncertainty of the existence of data objects. For example, mobile users can continuously appear and disappear in an area of interest.

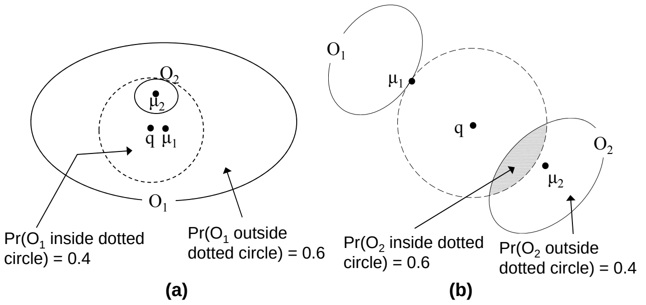

O2
μ2
q μ1
O1
Pr(O1 inside dotted circle) = 0.4
Pr(O1 outside dotted circle) = 0.6
(a)
O1
μ1
q
O2
μ2
Pr(O2 inside dotted circle) = 0.6
Pr(O2 outside dotted circle) = 0.4
(b)

Figure 2: Expectation-based NN Queries (a) Uniform Distributions (b) Non-uniform Distributions

Hence, each user belongs to the database with less than absolute confidence.

Integrating the above sources of uncertainty with the conventional distance-based criterion of NN queries raises new challenges regarding query semantics and evaluation.

1.1 Motivation and Challenges

A large body of research addresses NN queries where both data and query objects are deterministic points. Classical algorithms include the HS algorithm [14] and the RKV algorithm [25]. Recent proposals address NN queries under uncertainty with different query semantics. We describe some of these proposals in the following.

In [20], the problem is reduced to a conventional NN query, where the NNs are computed based on their expected distance to the query point. This method can give different answers from the alternative approach that reports objects with the highest marginal NN probabilities, and hence exploits data uncertainty rather than reducing the problem to a deterministic version. To illustrate, Figure 2 (a) depicts two moving objects O_1 and O_2 whose possible locations are uniformly distributed in the shown solid ovals with means μ_1 and μ_2, respectively. For the query point q, O_1 is the NN based on its expected distance to q. Assume that the probability of O_1 being inside the dotted circle, centered at q, is 0.4. Hence, O_2 is the NN based on marginal probabilities since the probability of O_1 being the NN is at most 0.4 (the probability of O_1 being inside the dotted circle). Non-uniform distributions may also exhibit the same discrepancy. For example, in Figure 2(b) objects have non-uniform distributions. O_1 is the NN based on expected distance to q, while O_2 is the NN based on marginal probabilities, since the probability of O_2 being the NN is at least 0.6 (the probability of O_2 being inside the shaded area).

Approximating marginal NN probabilities using sampling techniques, e.g., [16], suffers from an inherent slow diminishing of approximation error when increasing sample size. For example, in Monte-Carlo sampling, quadrupling the number of samples only halves the error [18].

Threshold-based probabilistic NN queries are addressed in [6], where objects with marginal NN probabilities above a given threshold are reported. Threshold-based queries have inherent problems in selecting a suitable threshold. Setting the threshold too high may lead to empty results, and hence the query needs to be restarted with a lower threshold. Alternatively, setting the threshold too low may produce too many results and increase query response time.

Current proposals are lacking with regard to two points:

- Most proposals assume limited uncertainty models that do not support all possible uncertainty sources. For example, in [7], data objects have uncertain attributes, while their existence in the database is certain. In [8], data objects have deterministic attributes, and uncertain existence in the database. In [16], data and query objects are uncertain, while their existence in database is certain. To the best of our knowledge, integrating all uncertainty sources within the same processing framework has not been addressed before.
- Current proposals separate the I/O operations (i.e., object retrieval) and CPU operations (i.e., probability computation) of probabilistic NN queries into two isolated stages, as in [7, 6]. No current work addresses interleaving these operations during query processing, or integrating their costs into a unified cost model.

Integrating all uncertainty sources in the same model adds further complexity to the problem. For example, if all objects have deterministic existence in the database, a large number of objects can be pruned using *spatial* properties [7, 6]. Specifically, an object cannot be the NN if its minimum distance to the query point is larger than the maximum distance of another object. Such pruning criterion is not directly applicable when objects' existence is uncertain.

Interleaving I/O and CPU operations, based on a cost model, allows for addressing the trade-offs among the cost factors of NN queries. This is particularly important if a small number of answers, e.g., the top-k answers, is required.

1.2 Contributions

We summarize our contributions as follows:

- We introduce Top$k$-PNN query, a novel formulation of NN queries combining both data/query uncertainty and distance-based criteria (Section 2).
- We study object retrieval orders of Top$k$-PNN queries, and give new results regarding the I/O optimality of different orders. We further analyze the cost factors of Top$k$-PNN queries, and construct a unified cost model and efficient query evaluation techniques (Section 3).
- We address Top$k$-PNN queries with uncertain query objects and both uncertain (Section 4) and deterministic (Section 5) data objects.
- We give multiple extensions to our techniques including handling dependencies among data objects, answering top-k queries over uncertain data (Appendix A), and supporting threshold queries (Section 6).

2. PROBLEM DEFINITION

We next describe the uncertainty model we adopt in this paper followed by our formal problem definition.

Uncertainty Model. We assume a database of objects $\mathcal{O} = \{O_1, \ldots, O_n\}$ such that the existence (membership) of objects in $\mathcal{O}$ is uncertain, i.e., $\Pr(O_i \in \mathcal{O}) \leq 1$. We denote with $\Pr(O_i)$, and $\Pr(\neg O_i) = 1 - \Pr(O_i)$ the existence and absence probabilities of O_i, respectively. Each object $O_i \in \mathcal{O}$ has a probabilistic attribute, defined as follows:

Definition 2.1. **Probabilistic Attribute.** *A probabilistic attribute A in object O_i is a random variable drawn from a distribution with density function f_i^A.* □

We assume f_i^A has a bounding *uncertainty region* R_i^A so that $\forall x \notin R_i^A$, $f_i^A(x) = 0$ and $\int_{R_i^A} f_i^A(x)\ dx = 1$. If f_i^A is an unbounded probability density function (PDF), e.g., Gaussian PDF, we truncate PDF tails with negligible probabilities and normalize the resulting PDF. This procedure is also used in other related works (e.g., [6, 5]), and in other contexts such as such as econometrics (e.g., [12]). We show in our experiments (Section 6.8) the effect of PDF truncation on the accuracy of the results. For a probabilistic attribute A whose domain is an n-dimensional space, we assume R_i^A is an n-dimensional hyper-rectangle. For example, the *location* attribute of an object moving in 2D plane, has a rectangular uncertainty region with 2D PDF. We restrict our discussions to 2D space for clarity. However, our techniques are applicable to n-dimensional spaces.

The query object q is an uncertain object with uncertainty region R_q, and a PDF f_q. The existence of query object is always certain, i.e., $\Pr(q) = 1$. We assume that our queries involve a single probabilistic attribute, e.g., location, and hence we use the object O_i to directly refer to its queried attribute $O_i.A$. We omit the superscript A for brevity.

Top$k$-PNN Queries. Let d be a distance metric, e.g., Euclidian distance. Since O_i has different possible values, there is a range of possible distances between O_i and the query object q. Let S_i be an interval enclosing all possible distances between O_i and q. We denote with $P_{nn}(O_i, q)$ the *marginal probability* of O_i to be the NN to q. The value of $P_{nn}(O_i, q)$ is found by summing all probabilities where $d(O_i, q)$ is the least among all other objects.

$$P_{nn}(O_i,q) = \int_{S_i} \Pr(O_i \wedge d(O_i,q) = s \\ \wedge\ \forall j \neq i(\neg O_j\ \vee\ d(O_j,q) > s)\ \ ds \tag{1}$$

Equation 1 integrates the probabilities of all settings where O_i is the NN to q.

We describe how to compute $P_{nn}(O_i, q)$ when all objects are independent. Given a point x, let $F_j(x, dist)$ be the probability that O_j does not exist or $d(O_j, x) > dist$. This probability is formulated as follows:

$$\begin{aligned} F_j(x, dist) &= \Pr(\neg O_j\ \vee\ d(x, O_j) > dist) \\ &= \Pr(\neg O_j)\ +\ \Pr(O_j) \cdot \int_{y \in R_j : d(x,y) > dist} f_j(y) dy \end{aligned} \tag{2}$$

Based on Equation 2, we formulate $P_{nn}(O_i, q)$ under independence as follows:

$$P_{nn}(O_i,q) = \Pr(O_i) \int_{R_q} \int_{R_i} f_q(x) \cdot f_i(y) \cdot \prod_{j \neq i} F_j(x, d(x,y))\ dy\ dx \tag{3}$$

Note that $\sum_{O_i} P_{nn}(O_i, q) \leq 1$. The value $(1 - \sum_{O_i} P_{nn}(O_i, q))$ corresponds to the probability of the configuration in which no object exists. The probability of such configuration is equal to $\prod_{O_i} \Pr(\neg O_i)$.

We assume that $P_{nn}(O_i, q)$ values are distinct across objects in $\mathcal{O}$ using a deterministic tie-breaker (a typical assumption in top-k algorithms, e.g., [10]). We next give our query definition.

Definition 2.2. **Top-k Probable NN (Top$k$-PNN) Query.** *Given a database $\mathcal{O}$ and a query object q, a Topk-PNN query returns a vector $V =< O_{(1)} \dots O_{(k)} > \subseteq \mathcal{O}$ such that: $P_{nn}(O_{(1)}, q) > \cdots > P_{nn}(O_{(k)}, q)$, and $\nexists O_i \in (\mathcal{O} - V)$ where $P_{nn}(O_i, q) > P_{nn}(O_{(k)}, q)$.* □

In general, Topk-PNN queries incur two cost factors:

1. I/O cost incurred by object retrieval, which can be the cost of reading objects from disk or transferring objects' details, e.g., a PDF histograms, over the network if objects are obtained from remote sources.
2. CPU cost incurred by computing a complex nested integral to evaluate the P_{nn} values of different objects.

Our techniques are based on optimizing the two above cost factors by exploiting general properties in top-k queries: (1) most database objects are not part of the query answer, and hence many I/O operations can be avoided; and (2) the scores of retrieved objects can be bounded, i.e., not fully computed, while still being able to rank query answers at reduced computational costs. We focus on the case of independent objects and discuss extensions to handle object dependencies in Appendix A.

3. QUERY EVALUATION

In this section, we describe our techniques to compute Top$k$-PNN queries with uncertain data objects and a single (deterministic) query point q. We discuss handling uncertain query objects in Sections 4 and 5.

Based on Equation 3, when q is a single point, $P_{nn}(O_i, q)$ is computed as follows:

$$P_{nn}(O_i,q) = \Pr(O_i) \int_{R_i} f_i(x) \prod_{j \neq i} F_j(q, d(q,x))\ dx \tag{4}$$

We discuss in Section 3.1 optimizing the number of I/O operations. We show in Section 3.2 how to perform computation lazily to optimize the CPU cost. Further, we describe in Section 3.3 techniques to optimize the combined cost of I/O and CPU.

3.1 Optimizing Object Retrieval

Let $\underline{d}(O_i, q)$ and $\overline{d}(O_i, q)$ denote the minimum and maximum distances between O_i and q, respectively. Figure 3(a) shows such distances. Further, let *min-dist order* denote the ascending order of objects based on $\underline{d}(., q)$.

Incremental Access Assumption. The main assumption we make on the class of algorithms we consider in this section is that objects are incrementally retrieved from the database with no prior information on the PDFs of non-retrieved objects. This assumption applies particularly to the case of retrieving objects from remote sources.

Incremental access of objects in the order of an arbitrary measure defined on objects boundaries, e.g., $\underline{d}(O_i, q)$ or $\overline{d}(O_i, q)$, can be made using an index over objects uncertainty regions (e.g., an R-tree). Building and traversing such index does not require knowledge about objects PDFs.

We propose `IO-Centric`, an algorithm to compute Top$k$-PNN queries with optimality guarantees on the number of I/O's. The idea is to incrementally retrieve objects from the database, while bounding $P_{nn}(O_i, q)$, for each retrieved object O_i, using an interval $[lo(O_i), up(O_i)]$. In addition,

Algorithm 1 `IO-Centric` ($\mathcal{O}$:database, q: query point)

```
$up(\phi) \leftarrow 1.0$ {upper bound of $P_{nn}$ value of non-retrieved objects}
Create an empty candidate set $\mathcal{C}$
{start growing phase}
while ($\mathcal{O}$ not exhausted AND $\nexists O_j \in \mathcal{C}$ with $lo(O_j) > up(\phi)$) do
    $O_i \leftarrow$ next object in $\mathcal{O}$ based on min-dist to $q$
    Add $O_i$ to $\mathcal{C}$
    Compute $lo(O_i)$ {Equ. 5}
    $up(\phi) \leftarrow up(\phi) - lo(O_i)$
end while
For each $O_j \in \mathcal{C}$, compute $up(O_j)$ {Equ. 6}
{start shrinking phase}
while ($\mathcal{O}$ not exhausted AND
        $\nexists O^* \in \mathcal{C}$ with $lo(O^*) > max(up(\phi), up(O_r))$ for any
        other object $O_r \in \mathcal{C}$) do
    Retrieve the next object based on min-dist to $q$
    For each $O_j \in \mathcal{C}$, update $lo(O_j)$, $up(O_j)$ {Equ. 5 and 6}
    For each $O_j \notin \mathcal{C}$, update $lo(O_j)$ {Equ. 5}
    Update $up(\phi)$ as $1 - \sum lo(.)$ of all retrieved objects
    Remove from $\mathcal{C}$ any object $O_j$ with $up(O_j) < lo(O_r)$ for
      some other object $O_r \in \mathcal{C}$
end while
return the object $O^*$ with max $up(.)$ in $\mathcal{C}$
```

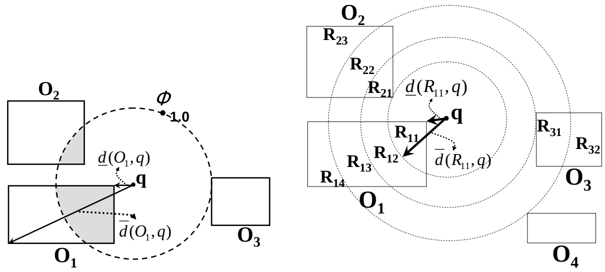
$P_{nn}(\phi, q)$, where ϕ represents any non-retrieved object, is upper-bounded with a function $up(\phi)$. Query answer is reported by reasoning about probability bounds to guarantee result correctness.

Algorithm 1 gives the details of `IO-Centric`. For illustration, we assume a Top$k$-PNN query with $k = 1$. We show at the end of this section how to compute queries with $k > 1$. The algorithm goes through two consecutive phases: (1) a growing phase, where a set of *candidates* grows by retrieving objects in min-dist order and updating their $lo(.)$ values until some object O_i satisfies $lo(O_i) > up(\phi)$ and hence any yet non-retrieved object cannot be the query answer; and (2) a shrinking phase, where candidates are pruned by retrieving new objects, and using these objects to update the bounds of candidates. The algorithm terminates when a candidate object O^*, retrieved in the growing phase, satisfies $(lo(O^*) > max(up(\phi), up(O_r))$, for any other candidate O_r. At this point, `IO-Centric` guarantees that O^* is the most probable NN to q. If the database is exhausted before entering the shrinking phase, the algorithm directly reports the candidate with the highest $up(.)$, since in this case $up(.)$ is the exact P_{nn} value, as discussed below.

We note that the behavior of `IO-Centric` is similar to the adaptation of the NRA algorithm in [22].

Bound Computation. Let $\acute{\mathcal{O}} \subseteq \mathcal{O}$ be the current set of retrieved objects, and $O_l \in \acute{\mathcal{O}}$ be the last retrieved object in min-dist order. The value of $lo(O_i)$ is given by Equation 4 by *pessimistically* estimating $P_{nn}(O_i, q)$ by assuming a non-retrieved object ϕ as a deterministic point (with probability 1) located at $\underline{d}(O_l, q)$. This setting maximizes the possible P_{nn} value of ϕ, and consequently minimizes the possible P_{nn} values of all retrieved objects (since the summation of P_{nn} values is ≤ 1). Formally, $lo(O_i)$ is computed as follows:

$$lo(O_i) = \Pr(O_i) \cdot \int_{x \in R_i : d(q,x) < \underline{d}(q, O_l)} f_i(x) \cdot \prod_{O_j \in \acute{\mathcal{O}} \wedge j \neq i} F_j(q, d(q,x))\ dx \tag{5}$$



(a) Lower-bounding $P_{nn}(.)$ (b) Initial Object Partitions

Figure 3: Computing Probability Bounds

For example, Figure 3 (a) shows how to compute $lo(O_i)$ by integrating the shaded areas where O_l is O_3.

Similarly, the value of $up(O_i)$ is given by Equation 4 by *optimistically* estimating $P_{nn}(O_i, q)$ by assuming that the minimum distance between any non-retrieved object and q is greater than $\overline{d}(O_i, q)$, and hence it does not affect $P_{nn}(O_i, q)$. Formally, $up(O_i)$ is computed as follows:

$$up(O_i) = \Pr(O_i) \int_{R_i} f_i(x) \prod_{O_j \in \acute{\mathcal{O}} \wedge j \neq i} F_j(q, d(q,x))\ dx \tag{6}$$

Since the summation of P_{nn} values of all database objects is ≤ 1, we have $up(\phi) = 1 - \sum_{O_i \in \acute{\mathcal{O}}} lo(O_i)$, which is the maximum P_{nn} value a non-retrieved object could obtain, while $lo(\phi)$ is always zero.

Algorithm Analysis. We justify the way we compute $lo(O_i)$ and $up(O_i)$ by showing that they are the tightest bounds on $P_{nn}(O_i, q)$ under the *Incremental Access Assumption.* Then, we analyze the growing and shrinking phases of `IO-Centric`.

Lemma 3.1. *Let $\acute{\mathcal{O}}$ be a subset of objects retrieved in min-dist order under the* Incremental Access Assumption. *For $O_i \in \acute{\mathcal{O}}$, $lo(O_i)$ and $up(O_i)$ are the tightest lower and upper bounds of $P_{nn}(O_i, q)$ based on $\acute{\mathcal{O}}$, respectively.* □

We include the proof of Lemma 3.1 in Appendix B. It follows from Lemma 3.1 that any Top$k$-PNN algorithm working under the *Incremental Access Assumption*, and retrieving objects in min-dist order cannot terminate while retrieving less objects than `IO-Centric`.

Growing Phase. Our main result, represented in Theorem 3.2, is that Algorithm `IO-Centric` is I/O-optimal in the growing phase among the class of algorithms that work under *Incremental Access Assumption*, and use arbitrary retrieval orders (not necessarily min-dist). We include the proof of Theorem 3.2 in Appendix B.

Theorem 3.2. *Under the* Incremental Access Assumption, *any Topk-PNN algorithm $\mathcal{A}$ must retrieve all objects retrieved by Algorithm `IO-Centric` in the growing phase.* □

Shrinking Phase. Based on Equation 5, $lo(O_i)$ can only increase by retrieving the next object in min-dist order and adding this object to the set $\acute{\mathcal{O}}$. This is because for any other non-retrieved object $\acute{O}$, we have $\underline{d}(q, \acute{O}) \geq \underline{d}(q, O_l)$, and hence $\acute{O}$ does not change the value of the integral in

Equation 5. On the other hand, based on Equation 6, retrieving any new object $\acute{O}$ can be used to decrease $up(O_i)$ since Equation 6 does not restrict the values of $x \in R_i$. We hence make the following observations: (1) Arbitrary object retrieval order can result in decreasing $up(O_i)$, and (2) Only min-dist retrieval order can result in increasing $lo(O_i)$, and hence decreasing $up(\phi)$.

Since the shrinking phase starts when $lo(O_i) > up(\phi)$ is satisfied for some object O_i, any retrieved object in the shrinking phase is not a query answer. Retrieved objects in the shrinking phase tighten the probability bounds of candidate objects by either increasing the $lo(.)$ values and decreasing the $up(.)$ values of different candidates (which is only possible under min-dist retrieval), or by only decreasing the $up(.)$ values (which is possible using arbitrary retrieval order). Hence, the question is whether we can find an optimal retrieval order that leads to query termination with the least number of retrieved objects. Theorem 3.3 gives a negative result on the existence of such optimal order. We include the proof in Appendix B.

Theorem 3.3. *There exist two database instances D_1 and D_2 where, in the shrinking phase and under the* Incremental Access Assumption*, min-dist is the optimal retrieval order in D_1 but not the optimal retrieval order in $D2$.* □

Retrieval orders in the shrinking phase lead to query termination based on different factors including the overlap of uncertainty regions of different objects, and how objects' PDFs interact to decide their P_{nn} values. The effect of these factors cannot be known under our *Incremental Access Assumption* without actually retrieving the objects. We thus resort to heuristics to choose the object retrieval order in the shrinking phase. Since only min-dist order has the property of changing all $lo(.)$ and $up(.)$ values, as well as $up(\phi)$, we adopt min-dist retrieval in the shrinking phase.

Throughout the remainder of this paper, min-dist order is used in incremental object retrieval.

Top$k$-PNN Queries with $k > 1$. Algorithm `IO-Centric` is extended to answer Top$k$-PNN queries with $k > 1$ by changing the condition that ends the growing phase to: $\exists \mathcal{O}' \subseteq \mathcal{C}, |\mathcal{O}'| = k$ such that $\forall O' \in \mathcal{O}'(lo(O') > up(\phi))$, and continuing the shrinking phase until the condition in Definition 2.2 is satisfied.

3.2 Lazy Computation of Bounds

The bounds $lo(O_i)$ and $up(O_i)$ use nested integration on R_i. While these bounds are tight (Lemma 3.1), they involve high computation cost in return of optimal I/O cost. We note that using looser bounds and extra object retrievals may yield better overall query evaluation cost.

We propose a lazy bound-refinement technique that starts with the coarse granularity of the whole uncertainty region R_i, and lazily tightens the bounds by considering the finer granularity of subregions in R_i. Consider Equation 4. Starting with the granularity of R_i, we upper-bound $F_j(q, d(q, x))$ by replacing $d(q, x)$ with its smallest possible value $\underline{d}(q, O_i)$, resulting in $F_j(q, \underline{d}(q, O_i))$ which is independent of x. Hence, $P_{nn}(O_i, q)$ is upper-bounded as follows:

$$\begin{aligned}\overline{P_{nn}}(O_i, q) &= \Pr(O_i) \cdot \prod_{j \neq i} F_j(q, \underline{d}(O_i, q)) \cdot \int_{R_i} f_i(x)\ dx \\ &= \Pr(O_i) \cdot \prod_{j \neq i} F_j(q, \underline{d}(O_i, q))\end{aligned} \tag{7}$$

Similarly, $P_{nn}(O_i, q)$ is lower-bounded as follows:

$$\underline{P_{nn}}(O_i, q) = \Pr(O_i) \cdot \prod_{j \neq i} F_j(q, \overline{d}(O_i, q)) \tag{8}$$

Our bound-refinement procedure has the insight that partitioning R_i into smaller subregions gives tighter bounds on $P_{nn}(O_i, q)$, since we exploit further information in f_i. We prove this fact in Theorem 3.6. We start our description of the refinement procedure by defining partitions.

Definition 3.4. **Object Partition.** *A partition $\mathcal{P}_i$ for object O_i is a set of disjoint subregions $\{R_{i1} \dots R_{in}\}$ of R_i, such that $\mathcal{P}_i$ totally* covers *R_i.* □

For example, in Figure 3(b), $\{R_{31}, R_{32}\}$ is a possible partition of O_3. Let $\Pr(R_{ij}) = \int_{R_{ij}} f_i(x)dx$. It follows that $\sum_{R_{ij} \in \mathcal{P}_i} \Pr(R_{ij}) = \int_{R_i} f_i(x)dx = 1$. We use min-dist order to create initial object partitions, where a newly retrieved object O_l splits the partitions of already retrieved objects based on $\underline{d}(O_l, q)$, as shown in Figure 3(b).

Let $\underline{d}(R_{ij}, q)$ and $\overline{d}(R_{ij}, q)$ be the minimum and maximum distances between subregion R_{ij} and q, respectively. These distances are shown in Figure 3(b). A lower-bound of $P_{nn}(O_i, q)$, given $\mathcal{P}_i$, is computed as follows:

$$\underline{P_{nn}}(O_i, q|\mathcal{P}_i) = \Pr(O_i) \cdot \sum_{R_{ij} \in \mathcal{P}_i} \Pr(R_{ij}) \cdot \prod_{k \neq i} F_k(q, \overline{d}(q, R_{ij})) \tag{9}$$

Similarly, an upper-bound of $P_{nn}(O_i, q)$, given $\mathcal{P}_i$, is computed as follows:

$$\overline{P_{nn}}(O_i, q|\mathcal{P}_i)^* = \Pr(O_i) \cdot \sum_{R_{ij} \in \mathcal{P}_i} \Pr(R_{ij}) \cdot \prod_{k \neq i} F_k(q, \underline{d}(q, R_{ij})) \tag{10}$$

Since $\sum_{O_i} P_{nn}(O_i, q) \leq 1$, another valid upper-bound for $P_{nn}(O_i, q)$ is $1 - \sum_{j \neq i} \underline{P_{nn}}(O_j, q|\mathcal{P}_j)$. The overall upper-bound is thus computed as follows:

$$\overline{P_{nn}}(O_i, q|\mathcal{P}_i) = min(\overline{P_{nn}}(O_i, q|\mathcal{P}_i)^*, 1 - \sum_{j \neq i} \underline{P_{nn}}(O_j, q|\mathcal{P}_j)) \tag{11}$$

The above bounds provide multiple optimization opportunities of CPU cost. Bound computation cost is mainly dominated by the cost of computing $F_k(.)$, which can be efficiently done using a PDF index, e.g., [15]. Furthermore, it is possible to cache $F_k(q, dist)$ at different $dist$ values to be used with multiple bound computation.

While the above bounds assume all database objects are retrieved, they can still be used with partial retrieval of database objects, and hence supporting our incremental retrieval strategy. We discuss how this can be done by first distinguishing two types of object's subregions.

Definition 3.5. **Inner and Outer Subregions.** *Let O_l be the last retrieved object in min-dist order. A subregion R_{ij} is an inner subregion if $\overline{d}(R_{ij}, q) \leq \underline{d}(O_l, q)$, otherwise R_{ij} is an outer subregion.* □

For example, R_{11}, R_{12} and R_{13} in Figure 3(b) are inner subregions, given that the last retrieved object is O_4. On the other hand, R_{14} is an outer subregion.

Equations 9 and 10 can be adopted with partial retrieval of database objects, based on min-dist order, using the virtual object ϕ which represents any non-retrieved object. Specifically, to compute $\underline{P_{nn}}(.)$, the object ϕ is assumed to be located at the nearest possible distance to q, i.e., at $\underline{d}(O_l, q)$, with probability 1. In Equation 9, $\Pr(R_{ij})$, for an outer subregion R_{ij}, would thus be multiplied by $F_\phi(q, \overline{d}(q, R_{ij})) = 0$, while $\Pr(R_{ij})$, for an inner subregion, would be multiplied by $F_\phi(q, \overline{d}(q, R_{ij})) = 1$. Hence, only inner subregions contribute to $\underline{P_{nn}}(.)$. On the other hand, in Equation 10, in order to compute $\overline{P_{nn}}(.)$, the object ϕ is assumed to be located at the maximum possible distance from q (i.e., positive infinity), or equivalently, no more objects are available. Hence, $F_\phi(q, \underline{d}(q, R_{ij})) = 1$ for any subregion R_{ij}, and thus all object's subregions contribute to $\overline{P_{nn}}(.)$.

Partition Refinement. A partition $\mathcal{P}_i^1$ is refined by partitioning one of its subregions into two smaller subregions to create a *finer* partition $\mathcal{P}_i^2$. For example, in Figure 3(b), $\mathcal{P}_3^1 = \{R_3\}$ is refined by splitting R_3 to R_{31} and R_{32}, resulting in $\mathcal{P}_3^2 = \{R_{31}, R_{32}\}$. Theorem 3.6 shows that partition refinement leads to tightening the bounds of $P_{nn}(O_i, q)$. We include the proof in Appendix B.

THEOREM 3.6. *If partition* $\mathcal{P}_i^2$ *is* finer *than partition* $\mathcal{P}_i^1$*, then (1)* $\underline{P_{nn}}(O_i, q|\mathcal{P}_i^2) \geq \underline{P_{nn}}(O_i, q|\mathcal{P}_i^1)$*; and (2)* $\overline{P_{nn}}(O_i, q|\mathcal{P}_i^2) \leq \overline{P_{nn}}(O_i, q|\mathcal{P}_i^1)$. □

A partition $\mathcal{P}_i$ is refined by splitting either an inner or an outer subregion. Selecting which subregion to split is controlled by the cost model in Section 3.3. Let O_l be the last retrieved object in min-dist order, we consider two cases: **(1) Splitting inner subregions**. Finding the optimal split location, i.e., the location that would result in the largest bound tightening at the least cost, is by itself an optimization problem. The cost of such optimization may outweigh the benefit of finding the optimal refinement, since an optimization algorithm would effectively try many candidate split locations (we discuss splitting cost and benefit in Section 3.3). We thus adopt a heuristic to split an inner subregion at its *middle* distance to q. Our heuristic allows the bounds to converge to the exact P_{nn} value at a rate comparable to the optimal splitting method as shown in Appendix C. **(2) Splitting outer subregions**. Split location is set at $\underline{d}(O_{l+1}, q)$, i.e., by retrieving the next object O_{l+1}. This results in two smaller subregions; an *inner* and an *outer* subregion. Note that O_{l+1} initially has one *outer* subregion covering its entire uncertainty region.

Splitting subregions to the finest level, i.e., each subregion has the minimal width supported by numerical precision, reduces Equations 9 and 10 to the integral in Equation 4, which is the exact P_{nn} value, under the same precision.

3.3 Optimizing Total Cost

In this section, we show how to compute Top$k$-PNN queries while optimizing the combined I/O and CPU cost. We adopt a benefit-cost approach, where a *benefit* is obtained by refining the bounds of $P_{nn}(O_i, q)$, since such refinement leads to query termination, while a cost is incurred in bound computation and object retrieval. We thus view bound refinement as an *expensive predicate* with a *cost* and a *benefit*. Finding the optimal predicate evaluation order, i.e., the order that results in query termination at the least cost, is equivalent to the *optimal scheduling problem* [4], which is NP-hard. We propose a technique to *rank* bound refinement operations based on estimated benefit and cost, and iteratively apply the refinement with the highest rank.

Let $\mathcal{O}_{R_{ij}}$ be the set of objects overlapping with R_{ij} based on distance to q. For example, in Figure 3(b), $\mathcal{O}_{R_{13}} = \{O_2, O_3\}$. The cost of using R_{ij} in bound refinement is estimated as follows. For $O_k \in \mathcal{O}_{R_{ij}}$, let C_k be the cost of integrating the density function f_k over the subregion in O_k that overlaps with R_{ij} based on distance to q. For example, to use R_{13} in bound refinement, we need to integrate the density functions f_2 and f_3 over the subregions R_{22} and R_{31}, respectively. We use PDF indexing (e.g., aggregate R-tree [17]) to speed up the computation of integrals. Estimating the cost C_k depends on the type of PDF index. For example, when using an aggregate R-tree (aR-Tree) to index objects' PDFs, the number of visited index nodes is used to reflect the cost [29].

On the other hand, the benefit of a subregion R_{ij} is estimated as the difference between R_{ij}'s contributions to upper and lower bounds of $P_{nn}(O_i, q)$. The intuition is that subregions with large differences are expected to tighten $P_{nn}(O_i, q)$ bounds considerably when refined.

DEFINITION 3.7. **Refining Cost and Benefit.** *Let* C_{IO} *be object retrieval cost. Then,*

- $cost(R_{ij}) = \begin{cases} \sum_{O_k \in \mathcal{O}_{R_{ij}}} C_k & \textit{if } R_{ij} \textit{ is inner} \\ C_{IO} + \sum_{O_k \in \mathcal{O}_{R_{ij}}} C_k & \textit{if } R_{ij} \textit{ is outer} \end{cases}$
- $benefit(R_{ij}) = \Pr(R_{ij}) \cdot \left(\prod_{k \neq i} F_k(q, \underline{d}(R_{ij}, q)) - \prod_{k \neq i} F_k(q, \overline{d}(R_{ij}, q)) \right)$

□

The rank of R_{ij} is defined as follows:

$$rank(R_{ij}) = \frac{benefit(R_{ij})}{cost(R_{ij})} \tag{12}$$

Search Algorithm. Since $\sum_{O_i} P_{nn}(O_i, q) \leq 1$, tightening the bounds of one object affects the bounds of other objects. We are unaware of how much an object O_i affects other objects before actually tightening the bounds of $P_{nn}(O_i, q)$. We cannot thus ideally order objects for processing in a deterministic way (similar to selective predicates with dependencies [23]). Consequently, we choose to process objects in $\overline{P_{nn}}(O_i, q)$ order, following the upper-bound principle, which is widely-used in optimal top-k algorithms [10].

We now formulate `Find-Top`k`-PNN`, a search algorithm to find Top$k$-PNN query answer, while optimizing the total cost based on Definition 3.7. The details are given in Algorithm 2. The algorithm maintains an object priority queue $\mathcal{Q}$ based on $\overline{P_{nn}}(O_i, q)$. The queue is initialized with the virtual object ϕ. At each step, the algorithm removes $\mathcal{Q}$'s top object O_t (line 5). If the top object happens to be ϕ, a new object O_l is retrieved in min-dist order, its bounds are computed, and it is inserted in $\mathcal{Q}$ (lines 7-9). If $O_t \neq \phi$, we identify the subregion R_{ti} with the highest-rank in O_t (line 11). If R_{ti} is an *outer* subregion, a new object is retrieved to split R_{ti} (lines 12-14). Alternatively, if R_{ti} is an *inner* subregion, R_{ti} is split as discussed in Section 3.2 (line 15). New bounds of $P_{nn}(O_t, q)$ are computed and O_t is re-inserted in $\mathcal{Q}$ (line 17). An object O^* is reported once $\underline{P_{nn}}(O^*, q)$ is greater than upper-bounds of all other objects in $\mathcal{Q}$ including ϕ (lines 18-21). The algorithm terminates upon reporting k objects.

Figure 4 illustrates Algorithm `Find-Top`k`-PNN` using an example. In Figure 4(a), O_1, the first object in min-dist

Algorithm 2 `Find-Top`k`-PNN` ($\mathcal{O}$:database, q: query point, k:answer size)

```
Create a priority queue $\mathcal{Q}$ based on $\overline{P_{nn}}(O_i, q)$
Add $\phi$ to $\mathcal{Q}$, with bounds [0,1]
$reported \leftarrow 0$
while ($reported < k$) do
    $O_t \leftarrow$ remove top object in $\mathcal{Q}$
    if ($O_t$ is $\phi$ AND not all objects are retrieved) then
        $O_l \leftarrow$ next object in $\mathcal{O}$ based on min-dist to $q$
        $bounds(O_l) \leftarrow$ use Equations 7 and 8
        Add $O_l$ to $\mathcal{Q}$
    else
        $R_{ti} \leftarrow$ highest-rank subregion in $\mathcal{P}_t$ (Section 3.3)
        if ($R_{ti}$ is an outer subregion AND not all objects are retrieved) then
            Retrieve next object $O_l$, compute its bounds and add to $\mathcal{Q}$
        end if
        Split $R_{ti}$ into two subregions (Section 3.2)
    end if
    Update bounds of $O_t$, re-insert $O_t$ into $\mathcal{Q}$
    $O^* \leftarrow$ peek at top object in $\mathcal{Q}$
    if $(\forall O_i \in \mathcal{Q}, O_i \neq O^* : \underline{P_{nn}}(O^*, q) > \overline{P_{nn}}(O_i, q))$ then
        Report $O^*$, and remove it from $\mathcal{Q}$
        $reported \leftarrow reported + 1$
    end if
end while
```

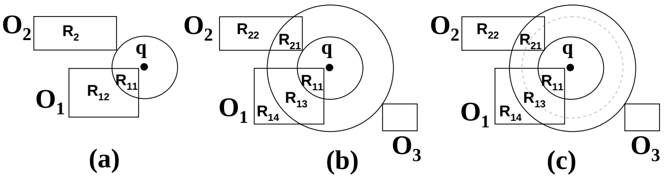


Figure 4: `Find-Top`k`-PNN` Processing Steps

order, is partitioned based on min-dist between the next object O_2 and q. Assume that O_1 has the highest $\overline{P_{nn}}(O_i, q)$, and that R_{12} is its highest-rank subregion, we thus need to partition R_{12}. In Figure 4(b), partitioning R_{12} involves retrieving another object O_3, which splits R_{12} into R_{13} and R_{14}, and O_2 into R_{21} and R_{22}. In Figure 4(c), assume that the next object to be processed is O_1 again and that R_{13} has the highest rank. Now, we further partition R_{13} without retrieving a new object. The algorithm continues to find query answer.

4. UNCERTAIN QUERY, UNCERTAIN DATA

We discuss how to extend our techniques to allow uncertainty in both the query and data objects.

4.1 Computing Probability Bounds

We extend our lazy bound refinement procedure (Section 3.2) to consider uncertain query object. We start by describing object partitioning.

For a data object O_i and a query object q, let partition $\mathcal{P}_i$ denote a set of subregion-pairs (R_{ij}, R_{qs}), where $R_{ij} \subseteq R_i$ and $R_{qs} \subseteq R_q$, such that the distinct R_{ij}'s in pairs in $\mathcal{P}_i$ are disjoint and totally cover R_i, and similarly the distinct R_{qs}'s in pairs in $\mathcal{P}_i$ are disjoint and totally cover R_q. That is, $\mathcal{P}_i$ is the cross-product of individual partitions of O_i and q. The subregions in each pair are defined as rectangles. For example in Figure 5, (R_{11}, R_{q1}) is a pair of subregions in $\mathcal{P}_1$ (shown as shaded rectangles). In the same figure, $\mathcal{P}_1$ contains a total of 16 pairs. The partition of q may be different with different data objects. For example in Figure 5, O_1 may use the partition $\{R_{q1}, \ldots, R_{q4}\}$, while O_2 uses the partition $\{R_q\}$. We thus have $(R_{11}, R_{q1}) \in \mathcal{P}_1$ and $(R_2, R_q) \in \mathcal{P}_2$. We show how to lazily construct these partitions in Section 4.2.

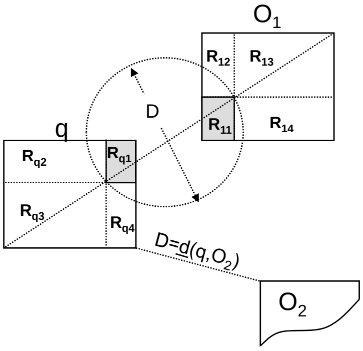


Figure 5: Partitioning Subregion Pairs

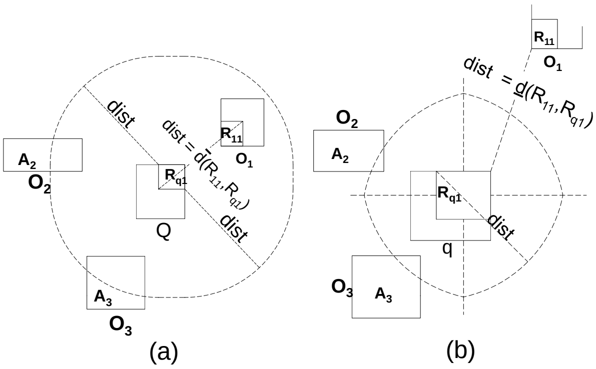


Figure 6: (a) Lower Bounds (b) Upper Bounds

The justification of the above partitioning method is that when q is uncertain, rectangular subregions facilitate computing the minimum and maximum distances among objects, as widely-used in classical NN techniques, e.g., [14].

Given the above partitioning scheme, we extend our bound computation procedure as follows:

$$\underline{P_{nn}}(O_i, q|\mathcal{P}_i) = \Pr(O_i) \cdot \sum_{(R_{ij}, R_{qs}) \in \mathcal{P}_i} \Pr(R_{ij}).\Pr(R_{qs}). \prod_{k \neq i} \underline{F_k}(R_{qs}, \overline{d}(R_{qs}, R_{ij})) \tag{13}$$

where $\underline{F_k}(R_{qs}, dist)$ is computed as follows:

$$\begin{aligned} \underline{F_k}(R_{qs}, dist) &= \Pr(\underline{d}(O_k, R_{qs}) > dist \vee \neg O_k) \\ &= \Pr(O_k) \cdot \Pr(\underline{d}(O_k, R_{qs}) > dist) + \Pr(\neg O_k) \end{aligned} \tag{14}$$

The lower-bound in Equation 13 is correct due to the following inequality:

$$\begin{aligned} &\forall x \in R_{ij}, y \in R_{qs} : \\ &\quad F_k(y, d(x, y)) \geq \underline{F_k}(R_{qs}, d(x, y)) \geq \underline{F_k}(R_{qs}, \overline{d}(R_{qs}, R_{ij})) \end{aligned} \tag{15}$$

We use Minkowski sum [3] to limit the integration area in $\underline{F_k}(R_{qs}, dist)$. The Minkowski sum of two areas results from summing every point in the first area with every point in the second. That is, $Minkowski\ sum(A, B) = \{x + y | x \in A, y \in B\}$. Let M be a Minkowski sum defined over R_{qs} and a circle centered at (0,0) with radius $dist$. For example, in Figure 6(a), we show the Minkowski sum for subregion-pair (R_{11}, R_{q1}), where $dist = \overline{d}(R_{11}, R_{q1})$. For any point x outside M, we have $\underline{d}(x, R_{qs}) > dist$. Thus, we find the value of $\underline{F_k}(R_{qs}, dist)$ by integrating the density function f_k outside M. For example, in Figure 6 (a), $\underline{F_2}(R_{q1}, \overline{d}(R_{q1}, R_{11})) = \Pr(O_2) \cdot \int_{A_2} f_2(x)\ dx + \Pr(\neg O_2)$.

Similar to the case of deterministic query point, we define an upper-bound of $P_{nn}(O_i, q)$ as follows:

$$\overline{P_{nn}}(O_i, q|\mathcal{P}_i) = Min(1 - \sum_{j \neq i} \underline{P_{nn}}(O_j, q|\mathcal{P}_j), \overline{P_{nn}}^{\,*}(O_i, q|\mathcal{P}_i)) \tag{16}$$

where $\overline{P_{nn}}^{\,*}(O_i, q|\mathcal{P}_i)$ is computed as follows:

$$\begin{aligned} &\overline{P_{nn}}^{\,*}(O_i, q|\mathcal{P}_i) = \\ &\Pr(O_i) \cdot \sum_{(R_{ij}, R_{qs}) \in \mathcal{P}_i} \Pr(R_{ij}) \cdot \Pr(R_{qs}) \cdot \prod_{k \neq i} \overline{F_k}(R_{qs}, \underline{d}(R_{qs}, R_{ij})) \end{aligned} \tag{17}$$

where $\overline{F_k}(R_{qs}, dist)$ is computed as follows:

$$\begin{aligned} \overline{F_k}(R_{qs}, dist) &= \Pr(\overline{d}(O_k, R_{qs}) > dist \vee \neg O_k) \\ &= \Pr(O_k) \cdot \Pr(\overline{d}(O_k, R_{qs}) > dist) + \Pr(\neg O_k) \end{aligned} \tag{18}$$

The correctness of the upper-bound given by Equation 17 can be proved similar to the correctness of lower-bound. We compute $\overline{F_k}(R_{qs}, dist)$ by integrating the density function f_k outside S, where S is defined using four arcs with radii equal to $dist$, and the center of each arc is the furthest opposite corner of R_{qs}. For example, Figure 6(b) shows the four arcs drawn from the corners of R_{q1} with radii $dist$. For any point x outside S, we have $\overline{d}(x, R_{qs}) > dist$. For example, in Figure 6(b), we compute the value of $\overline{F_2}(R_{q1}, \underline{d}(R_{q1}, R_{11}))$ as $\Pr(O_2) \cdot \int_{A_2} f_2(x)\ dx + \Pr(\neg O_2)$.

4.2 Refining Objects' Partitions

Refining a partition $\mathcal{P}_i$ has to take into consideration query uncertainty. We extend our definition of inner/outer subregions as follows. Let O_l be the last retrieved object. We call a subregion-pair (R_{ij}, R_{qs}) *inner* if $\overline{d}(R_{ij}, R_{qs}) \leq \underline{d}(O_l, q)$, otherwise we call it an *outer* subregion-pair. Definition of inner and outer subregions in this case allows using retrieved objects for bound computation in Equations 13 and 16 based on the same discussion in Section 3.2.

Splitting an outer subregion-pair. Splitting an *outer* subregion-pair (R_{ij}, R_{qs}) is performed upon retrieving a new object O_{l+1} based on min-dist order. To generate a smaller *inner* subregion-pair (R'_{ij}, R'_{qs}) from this split, we need to select the split location such that $\overline{d}(R'_{ij}, R'_{qs}) \leq \underline{d}(O_{l+1}, q)$. Any circle with diameter less than or equal to $\underline{d}(O_{l+1}, q)$ intersecting with both R_{ij} and R_{qs} can be used for such splitting. Such circle would enclose the new subregions R'_{ij} and R'_{qs} (e.g., the dotted circle in Figure 5. There is potentially an infinite number of circles that satisfy these two requirements. Our strategy is to find a split that maximizes the probability of the resulting inner subregions in order to maximize their effect on the bounds (Section 3.2).

We heuristically obtain such split by locating two points along the line that connects the farthest points of R_{ij} and R_{qs} such that the distance between each point and the line's midpoint is equal to $\underline{d}(O_{l+1}, q)/2$. The obtained points represent the two ends of a diameter for the circle used for splitting. For example, Figure 5 shows how to split (R_1, R_q) based on object O_2. We obtain one inner subregion-pair (R_{11}, R_{q1}), and 15 outer subregion-pairs which are the combinations of other subregions in R_1 and R_q.

Splitting an inner subregion-pair. Splitting an *inner* subregion-pair (R_{ij}, R_{qs}), is performed by extending our *middle* distance method (Section 3.2). Specifically, we select the subregion (either R_{ij} or R_{qs}) that has the larger density, and split it at the middle of its largest dimension. The intuition is to decrease the difference between the minimum and maximum distances between R_{ij} and R_{qs}, and thus allow for obtaining tighter bounds.

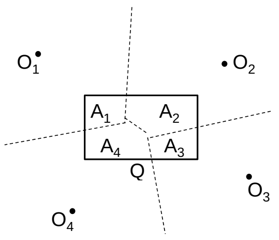


Figure 7: Voronoi Diagram for Uncertain Query

5. UNCERTAIN QUERY, CERTAIN DATA

In this section, we discuss answering Topk-PNN queries when data objects have deterministic attributes and certain membership (i.e., $\Pr(O_i) = 1, \forall O_i \in \mathcal{O}$), while the query object is uncertain. For example, in Figure 7, the data objects are represented by points in the space, while the query object is bounded by the uncertainty region R_q which is shown as a solid rectangle. Based on this setting, $P_{nn}(O_i, q)$ is computed as follows:

$$P_{nn}(O_i, q) = \int_{R_q} f_q(x) \cdot \prod_{j \neq i} F_j(x, d(x, O_i))\ dx \tag{19}$$

where $F_j(x, dist) = 1$ if $d(x, O_j) > dist$, and 0 otherwise.

To compute $P_{nn}(O_i, q)$, we propose an efficient algorithm based on Voronoi diagram [2], which is widely-used in (reverse) nearest neighbors queries. A Voronoi diagram divides the space into disjoint cells, each of which is associated with one object O_i such that O_i is the NN to any point in this cell. For example, each one of the four objects shown in Figure 7 exists in a separate subregion of the space.

Therefore, the NN to q would be O_i whenever q resides in the area A_i, which is the intersection of R_q and the Voronoi cell of O_i. Hence, $P_{nn}(O_i, q) = \Pr(q \in A_i) = \int_{A_i} f_q(x)dx$.

Voronoi diagram can be constructed off-line for stationary data objects [2]. We show how to answer Top$k$-PNN queries under these settings. We limit the number of retrieved objects by using an index over data objects (e.g., R-tree) to prune any object whose minimum distance to q is greater than the maximum distance between q and some other object. We compute $P_{nn}(O_i, q)$ for retrieved objects by simple integration of $f_q(.)$ over each subregion A_i of R_q. We maintain a virtual object, ϕ, that represents non-retrieved objects, where $\overline{P_{nn}}(\phi, q) = 1 - \sum_i P_{nn}(O_i, q)$. We update

$\overline{P_{nn}}(\phi, q)$ when an object is retrieved. An object O^* is reported if $\underline{P_{nn}}(O^*, q)$ is greater than all other $P_{nn}(.)$ values as well as $\overline{P_{nn}}(\phi, q)$. The algorithm terminates when k objects are reported.

6. EXPERIMENTS

In addition to Algorithms Find-Top$k$-PNN and IO-Centric, we implemented the following algorithms for performance comparison:

- CPU-Centric: A variant of Find-Topk-PNN that optimizes CPU cost only by loading all objects that survive spatial pruning (Section 1) into memory, and tightening their bounds lazily until query answer is reported.
- Baseline [7]: An algorithm that filters objects using spatial pruning, computes the exact $P_{nn}(.)$ values for all objects, and returns the top-k objects. The computation of $P_{nn}(.)$ values is improved by restricting integration to the overlapping areas of objects.
- Probabilistic Verifiers [6]: A threshold-based probabilistic NN algorithm. The algorithm filters objects using spatial pruning, then partitions objects into subregions that are used to bound objects' $P_{nn}(.)$ values. If computed bounds do not allow termination, nested integration is used per each subregion to compute exact probabilities.
- Find-Threshold-PNN: A threshold based version of Algorithm Find-Top$k$-PNN that reports all objects whose $P_{nn}(.)$ values are above a given threshold τ. We extend Algorithm 2 as follows. The stopping criterion (line 4) is modified so that the algorithm terminates when the object O_t on the top of the queue has $\overline{P_{nn}}(O_t, q)$ below τ, since in this case no other object can have $P_{nn}(.)$ value above τ. The answer reporting criterion (line 19) is modified such that an object O^* is reported, and removed from $\mathcal{Q}$, if $\underline{P_{nn}}(O^*, q)$ is above τ.

We compare our techniques to Algorithms CPU-Centric and Baseline. We additionally compare between Algorithms Find-Threshold-PNN and Probabilistic Verifiers. We use real and synthetic data in our comparisons. Our performance metrics are query response time, and the number of retrieved objects.

6.1 Experimental Setup

All experiments are conducted on a SunFire X4100 server with 2.2GHz processor, and 2GB of RAM. We used an open source R*-tree implementation [13] to index the bounding rectangles of objects. Objects are retrieved in min-dist order using best-first tree traversal. An object PDF is represented as a histogram of 300 bins indexed using an aggregate R-tree [17] which allows efficient density aggregation.

We used 'Los Angeles' dataset, a real dataset available in [1], with 60K geographical objects described by ranges of longitudes and latitudes. We used Uniform and Normal distributions as the objects' PDFs. For Synthetic data, we generated data objects in a 2-dimensional grid of size 1000×1000. The PDFs f_i's can be either uniform, normal or skewed, where skewed PDFs are generated by shifting the mean of a normal distribution based on a *skewness* parameter $\in [-1, 1]$, and normalizing the resulting distribution. Positive skewness means that PDF is biased towards q, while negative skewness means that PDF is biased away from q. We additionally truncate unbounded PDFs (i.e., normal and skewed) such that the probability of the truncated region is less than 0.003. The truncated PDFs are then normalized. Our problem parameters are the following:

- Number of Objects: The number of objects ranges from 100,000 to 300,000 (default is 100,000).
- Size of Objects: The size of each object ranges from 10×10 to 100×100 (default is 100×100).
- Data Distribution: We experimented with uniform, normal, positively skewed and negatively skewed objects' PDFs (default is normal).
- Source of Uncertainty: We experimented with (1) certain query object and uncertain objects with certain membership (CQ,UO), (2) certain query and uncertain objects with uncertain membership (CQ,UEO), and (3) uncertain query and uncertain objects with certain membership (UQ,UO) (default is (CQ,UO)).
- k: The value of k changes from 1 to 10 (default is 1).

In each experiment, we change the value of one parameter, while setting all other parameters at their default values.

6.2 Algorithms General Behavior

In general, IO-Centric retrieves the least number of objects among all algorithms, followed by Find-Top$k$-PNN. However, IO-Centric has slower running time than Find-Top$k$-PNN in most cases since IO-Centric only minimizes the I/O cost. If the number of retrieved objects is significantly small, e.g., PDFs with positive skewness as in Figure 11, IO-Centric incurs almost the same cost as Find-Top$k$-PNN, since both algorithms process a few objects.

CPU-Centric is more efficient when most data objects need to be retrieved, e.g., (UQ,UO) in Figure 12 and Skewed(-1) distribution in Figure 11, since in this case CPU-Centric benefits from ignoring the overhead of scheduling object retrievals. Find-Top$k$-PNN has the best running time in almost all experiments, since it takes into consideration the combined cost of CPU and I/O.

Baseline is typically an order of magnitude slower than Find-Top$k$-PNN. For example, for 100,000 objects in Figure 10, Baseline terminates in 2063 seconds, while Find-Top$k$-PNN terminates in 141 seconds. The main reason is the significant computational overhead incurred in the full integral evaluation.

Figure 10 shows that increasing the number of objects does not severely degrade the performance of Find-Top$k$-PNN. For example, tripling the number of objects from 100,000 to 300,000 results in less than one order of magnitude increase in the running time (from 141 to 885 seconds). Increasing the total number of objects has less impact on the number of retrieved objects (went from 616 to 1750 objects in the same example).

6.3 Real vs. Synthetic Data

The running times of our techniques with real data (Figure 8(a)) is significantly smaller than the time with synthetic data (Figure 9 (a)). The reason is that the majority of objects in the real data are scattered and hardly overlapping

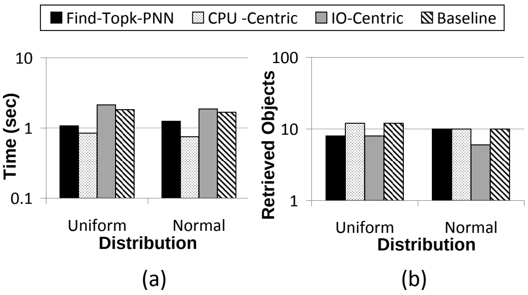


Figure 8: Performance Using Real Dataset

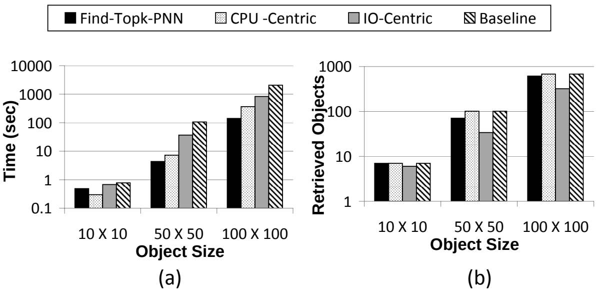


Figure 9: Effect of Object Size

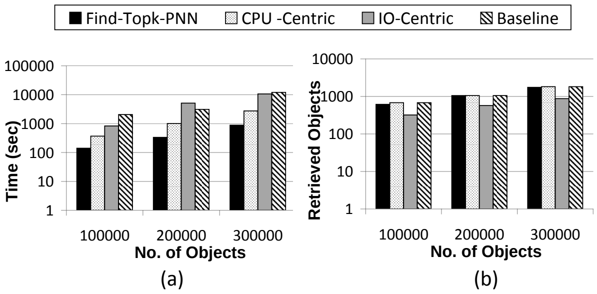


Figure 10: Effect of No. of Objects

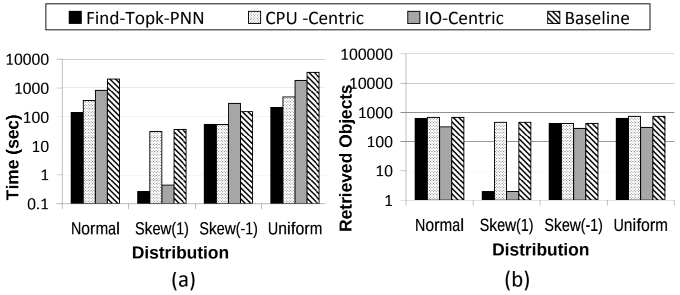


Figure 11: Effect of PDF

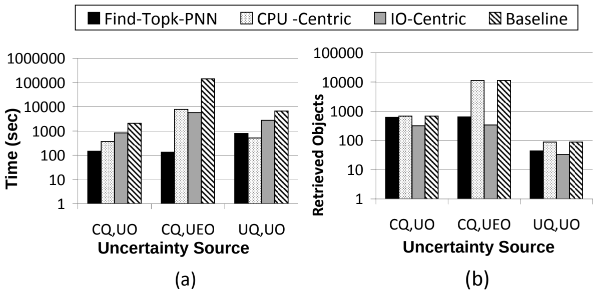


Figure 12: Effect of Uncertainty Source

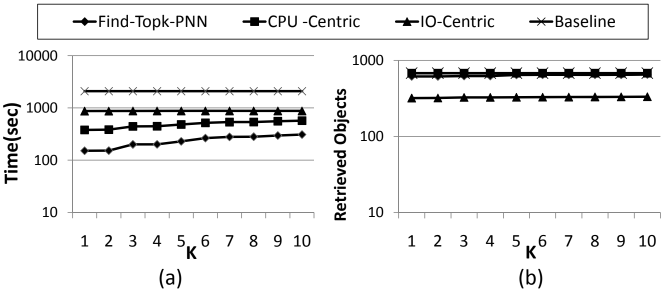


Figure 13: Effect of K Value

(each object covers less than 0.001% of the space), while in the synthetic data, objects are heavily overlapping (each object covers 10% of the space in default configuration). Similarly, our techniques retrieve a larger number of objects with synthetic data (Figure 10 (b)) since a large number of objects are candidate answers. The inverse relationship between objects' overlapping and query response time is also illustrated in Figure 9, where the degree of overlapping is controlled by varying object size in synthetic data.

6.4 Effect of Data Distribution

Figure 11 shows that the number of retrieved objects is relatively small when objects' distribution is positively skewed (2 object retrieved by `Find-Top`k`-PNN` compared to 616 objects in the default configuration). This is due to the fact that positively skewed PDFs increase $\underline{P_{nn}}(.)$ values rapidly with a small number of retrievals. Consequently, a rapid decrease in $\overline{P_{nn}}(\phi, q)$ occurs, and both `IO-Centric` and `Find-Top`k`-PNN` quickly terminate. `CPU-Centric` and `Baseline` do not gain much from positive skewness since they initially retrieve all candidate answers. Negatively skewed PDFs result in relatively larger number of retrievals for the opposite reason. For example, `Find-Top`k`-PNN` retrieves the same number of objects as `Baseline` in this case.

6.5 Effect of Uncertainty Source

Figure 12 (a) shows that the uncertainty of objects existence leads to increasing the running times. The reason is that pruning objects based on spatial properties is not applicable unless a retrieved object has a membership probability of 1, which results in a large number of retrievals as shown in Figure 12 (b). This leads to significant increase in the running times of `Baseline` and `CPU-Centric` since they mainly depend on spatial pruning. For example, the number of retrievals in these algorithms is 11299 objects (11% of all objects) at (CQ,UEO), compared to 680 (0.6%) at (CQ,UO). The running time significantly increases when the query object is uncertain (UQ,UO). The reason is that query uncertainty results in looser $P_{nn}(.)$ bounds compared to the deterministic query point, and hence, additional computation is needed.

6.6 Scalability with k

Figure 13 shows the performance with different k values. The running times of `Find-Top`k`-PNN`, `CPU-Centric`, and `IO-Centric` slightly increase with k, e.g., from 141 seconds for k=1 to 309 seconds for k=10 in `Find-Top`k`-PNN`. `Baseline` has the same running time for all k values since it always computes the exact $P_{nn}(.)$ values of all objects. The number of retrieved objects in `CPU-Centric` and `Baseline` are the same for all k values since they both avoid object retrievals based only on spatial pruning.

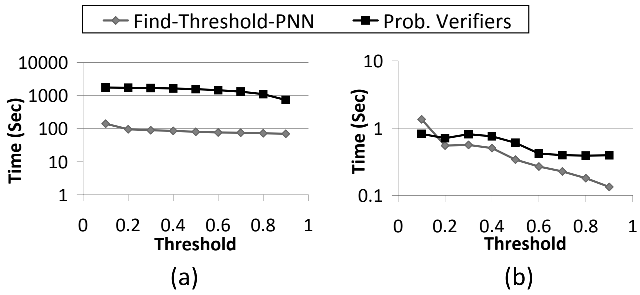


Figure 14: `Find-Threshold-PNN` **Vs.** `Probabilistic Verifiers` **(a) Synthetic Data (b) Real Data**

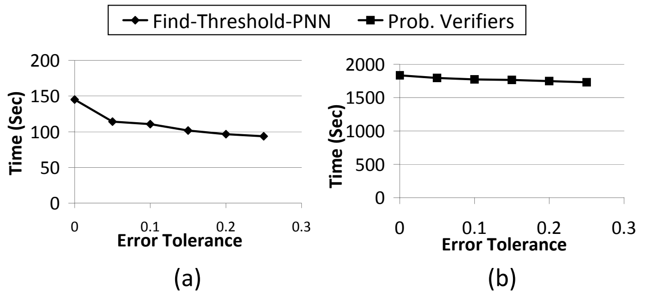


Figure 15: Effect of Error Tolerance (a) `Find-Threshold-PNN` **(b)** `Probabilistic Verifiers`

6.7 Comparison with Other Approaches

We compare `Find-Threshold-PNN`, our threshold-based extension with `Probabilistic Verifiers`. `Probabilistic Verifiers` resorts to expensive integration when objects cannot be judged to be in query result based on their initial bounds. This approach shows longer running times when compared to `Find-Threshold-PNN` in both synthetic data (one order of magnitude difference in Figure 14(a)) and real data (Figure 14(b)). The difference in running time is smaller (less than one second) with real data since objects are hardly overlapping, and thus integration cost is small. The running times of both algorithms decrease when the threshold approaches 1, since high threshold values allow pruning large number of objects using their initial bounds, i.e., without performing expensive refinement. `Find-Threshold-PNN` slightly outperforms `Probabilistic Verifiers` w.r.t number of retrieved objects.

We also evaluate `Find-Threshold-PNN` with threshold queries that allow a small error in the output as proposed in [6]. That is, we also report any object O_i with $\overline{P_{nn}}(O_i, q) - \underline{P_{nn}}(O_i, q) < \Delta$ and $\overline{P_{nn}}(O_i, q) > \tau$. The value of parameter Δ represents the amout of error tolerance. We show in Figure 15 the effect of varying Δ on the performance while we fix the probability threshold at 0.1. We note that the running time of Algorithm `Find-Threshold-PNN` has improved 35% when increasing Δ from 0 to 0.25. On the other hand, the `Probabilistic Verifiers` approach experienced 5% improvement for the same change in Δ. This observation suggests that Algorithm `Find-Threshold-PNN` exploits error tolerance more efficiently to decrease the running time. Our explanation is that Algorithm `Find-Threshold-PNN` performs lazy tightening of bounds only when stopping criteria is not met and thus avoids unnecessary computations.

6.8 PDF Truncation Error

In this experiment, we study the effect of truncating the objects' PDFs (f_i's) on the accuracy of query results and on the performance of our algorithms. We denote by ϵ the value of the integral of f_i over the truncated region. We vary ϵ in the range $[0.00005, 0.01]$. All other problem parameters are set to their default values, while k is set to 10.

To measure the accuracy, we compare the vector V of ranked answers, computed at some $\epsilon \in [0.00005, 0.01]$, and the vector $\acute{V}$ of ranked answers, computed at the minimum $\epsilon = 0.00005$, using two metrics: (1) $Footrule(V, \acute{V}) = \sum_{O_i \in V} |\text{Rank}(O_i \ in \ V) - \text{Rank}(O_i \ in \ \acute{V})|$, and (2) $FalsePositives(V, \acute{V}) = |\{O_i : O_i \in V \ and \ O_i \notin \acute{V}\}|$.

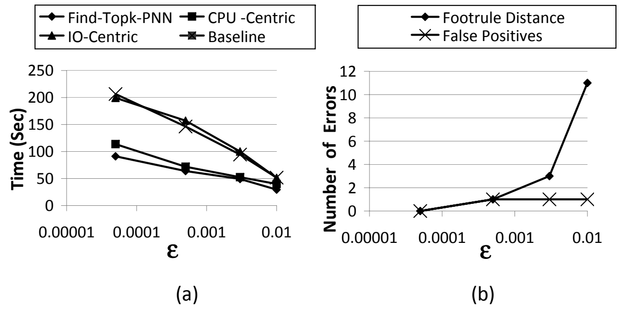


Figure 16: Effect of PDF Truncation (a) Running Time (b) Error

Figure 16 shows our results for different ϵ values. As ϵ increases, the running time improves for all algorithms due to the shrinkage of uncertainty regions, which leads to less overlapping between objects. On the other hand, larger values of ϵ produce errors in results as we ignore larger regions of PDFs. For example, going from $\epsilon = 0.00005$ to 0.0005 reduces the running time of Algorithm `Find-Top`k`-PNN` from 91 seconds to 62 seconds and the number of retrieved objects from 286 to 155, while introducing a single error.

7. RELATED WORK

Probabilistic NN queries have gained recent attention due to emerging applications that involve uncertainty. In [7], a probabilistic data model was proposed to capture objects with uncertain locations and certain membership. NN queries are defined so that all objects with non-zero probabilities of being the NN are reported, which is different from Top$k$-PNN queries. Furthermore, the proposed spatial pruning does not apply to the case of uncertain membership.

In [16], answering probabilistic NN queries using sampling methods is studied. Both query and data objects can be uncertain in this approach. The proposed algorithm also detects the cases that can be solved based only on the spatial properties of objects.

A recent approach has been introduced in [6] to solve the problem of probabilistic NN by reporting all objects with probabilities above a specific threshold, with a given error tolerance. The proposed algorithm goes through three stages to determine whether an object is part of the query answer or not. The first stage prunes objects based on their spatial properties (similar to [7]). In the second stage, space is divided into subregions based on the minimum and maximum distances between objects and the query point. Lower and upper bounds of $P_{nn}(.)$ values are then computed using coarse grained CDFs corresponding to the space partitioning to avoid performing nested integration. If the computed bounds are not enough to terminate, nested integration is incrementally performed over each subregion to compute its exact contribution to the object's $P_{nn}(.)$ value. We contrast and compare our algorithms to this approach in Section 6.

Another technique to answer probabilistic NN queries has been proposed in [8], where objects are represented as deterministic points associated with membership probabilities. However, the proposed model does not support uncertainty in objects' attributes.

Our formulation is similar to [24], where probabilistic top-k queries are addressed using Monte-Carlo multi-simulation. We refine probability bounds guided by a cost model, while [24] adopts randomized refinement. Additionally, our problem involves correlations among $P_{nn}(.)$ bounds of different objects, while the bounds computed in [24] are independent.

A related problem is answering probabilistic range queries [5]. The proposed model allows uncertain query and data objects. Addressing NN queries under the same model raises different challenges, as it involves not only the interaction between each data object and the query object, but also the interaction among objects.

PDF indexing methods (e.g., [28, 15]) can lower the cost of integrating PDFs by storing PDF synopses to allow fast pruning of objects that do not satisfy the query criteria. Although these techniques are proven to be efficient in range queries, they alone cannot provide efficient processing of Topk-PNN queries. The reason is that the execution of Topk-PNN queries is mainly influenced by the interaction among objects' PDFs. PDF indexing cannot directly be used to resolve object overlapping, where nested integration is needed (Section 2).

8. CONCLUSION

In this paper, we proposed a novel approach to efficiently compute NN queries in probabilistic databases where data and query objects are uncertain. We studied the I/O optimality of different retrieval orders. We introduced a unified cost model combining the I/O and CPU factors. We designed efficient query processing algorithms to minimize the total incurred cost. We also introduced extensions to our methods to handle dependent objects and threshold queries. Our experimental results show orders of magnitude performance gain, compared to current methods.

9. REFERENCES

[1] Topologically integrated geographic encoding and referencing (tiger) system, http://www.census.gov/geo/www/tiger/.

[2] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 1991.

[3] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. Computational geometry algorithms and applications, 2nd ed. springer verlag, 2000.

[4] K. C.-C. Chang and S. won Hwang. Minimal probing: supporting expensive predicates for top-k queries. In *SIGMOD*, 2002.

[5] J. Chen and R. Cheng. Efficient evaluation of imprecise location-dependent queries. In *ICDE*, 2007.

[6] R. Cheng, J. Chen, M. Mokbel, and C. Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *ICDE*, 2008.

[7] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. In *TKDE*, 2004.

[8] X. Dai, M. L. Yiu, N. Mamoulis, Y. Tao, and M. Vaitis. Probabilistic spatial queries on existentially uncertain data. In *SSTD*, 2005.

[9] V. de Almeida and R. Hartmut. Supporting uncertainty in moving objects in network databases. In *GIS*, 2005.

[10] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Science*, 2001.

[11] H. Franco-Lopez, A. R. Ek, and M. E. Bauer. Estimation and mapping of forest stand density, volume, and cover type using the k-nearest neighbors method. *Remote Sensing of Environment 2001.*

[12] J. Geweke. Efficient simulation from the multivariate normal and student–t distributions subject to linear constraints and the evaluation of constraint probabilities. In *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, 1991.

[13] M. Hadjieleftheriou. Spatial index library, http://research.att.com/˜marioh/spatialindex/.

[14] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *SSD*, 1995.

[15] D. V. Kalashnikov, Y. Ma, S. Mehrotra, and R. Hariharan. Index for fast retrieval of uncertain spatial point data. In *GIS*, 2006.

[16] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA*, 2007.

[17] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *SIGMOD Conference*, 2001.

[18] E. Li, D. Boos, and M. Gumpertz. Simulation study in statistics. *Journal of Interconnection Networks*, 2001.

[19] Q. Liu, W. Yan, H. Lu, and S. Ma. Occlusion robust face recognition with dynamic similarity features. In *ICPR*, 2006.

[20] V. Ljosa and A. K. Singh. Apla: Indexing arbitrary probability distributions. In *ICDE*, 2007.

[21] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, 2003.

[22] N. Mamoulis, M. L. Yiu, K. H. Cheng, and D. W. Cheung. Efficient top-k aggregation of ranked inputs. *ACM TODS.*, 32(3), 2007.

[23] K. Munagala, S. Babu, R. Motwani, and J. Widomy. The pipelined set cover problem. In *ICDT*, 2005.

[24] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.

[25] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, 1995.

[26] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[27] M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.

[28] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, 2005.

[29] Y. Tao, D. Papadias, and J. Zhang. Aggregate processing of planar points. In *EDBT*, 2002.

[30] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, 1998.

[31] R. Yamamoto, H. Matsutani, H. Matsuki, T. Oono, and H. Ohtsuka. Position location technologies using signal strengths in cellular system. In *VTC-Spring*, 2001.

APPENDIX

A. EXTENSIONS

Handling Dependencies. Our previous discussion assume independent objects. However, in some scenarios we might have object dependencies. For example, in Example 1.1, the locations of two cell phones could be correlated such that their distance cannot exceed some value.

We show how to handle object dependencies in the following setting. Assume a setting where objects are partitioned into groups G_i's, where objects in the same G_i are mutually exclusive such that $\Pr(G_m) = \sum_{O_i \in G_m} \Pr(O_i) \leq 1$, $\Pr(\neg G_m) = 1 - \Pr(G_m)$, and each object belongs to exactly one group. This type of dependency can arise in scenarios that involve uncertainty on objects' identities due to the unreliability of the sources of extracted information, e.g., low-quality images. For example, two isolated objects may be suspected to be the same entity, such that only one of them could be true, while the other is noisy data.

Let $\mathcal{G}^{-i}$ be the set of groups excluding the group that contains O_i. We next show how to compute bounds on $P_{nn}(.)$ values in this case.

$$\underline{P_{nn}}(O_i, q|\mathcal{P}_i) = \Pr(O_i) \cdot \sum_{R_{ij} \in \mathcal{P}_i} \Pr(R_{ij}) \prod_{G_m \in \mathcal{G}^{-i}} F_{G_m}(q, \overline{d}(q, R_{ij})) \tag{20}$$

$$\overline{P_{nn}}(O_i, q|\mathcal{P}_i) = Min(1 - \sum_{j \neq i} \underline{P_{nn}}(O_j, q|\mathcal{P}_j) \ , \\ \Pr(O_i) \cdot \sum_{R_{ij} \in \mathcal{P}_i} \Pr(R_{ij}) \prod_{G_m \in \mathcal{G}^{-i}} F_{G_m}(q, \underline{d}(q, R_{ij}))) \tag{21}$$

where $F_{G_m}(q, dist) = \Pr(\neg G_m) + \sum_{O_k \in G_m} \Pr(O_k) \cdot \Pr(d(q, O_k) > dist)$.

The above formulation takes into account exclusiveness among objects by summing up the probabilities of group members using the $F_{G_m}(.)$ terms, and multiplying the probabilities of different groups together, since there are no inter-group dependencies.

Our incremental retrieval model applies to the above formulation, since we can compute the bounds based on the current set of retrieved objects only. As mentioned in Section 3.2, we only include inner subregions when computing lower bounds, and we include both inner and outer subregions when we compute upper bounds.

Top-k Queries with Probabilistic Scores. The formulations and techniques presented in this paper are in the context of NN queries. However, our techniques can be extended to solve other related problems. Specifically, we consider top-k queries, where data objects have continuous score distributions. The query semantics we support is to report the k most probable top-1 answers. Such top-k queries can be mapped to probabilistic NN queries by modeling object's score as 1-dimensional uncertainty region, enclosing the possible score values, associated with the score density. The query point for such NN query is a point located in 1-dimension at the maximum possible score.

B. PROOFS

B.1 Proof of Lemma 3.1

Let $O_l \in \acute{\mathcal{O}}$ be the last retrieved object in $\acute{\mathcal{O}}$. For any $O_i \in \acute{\mathcal{O}}$, let $lo(O_i)$ be as given in Equation 5. Assume that another lower-bound $\acute{lo}(O_i) > lo(O_i)$ exists. Assume that the next non-retrieved object has a probability of 1 at $\underline{d}(O_l, q)$. In this case, $P_{nn}(O_i, q) = lo(O_i)$. Hence, $\acute{lo}(O_i)$ is an incorrect bound.

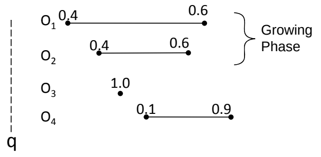


Figure 17: D_1**: Min-dist retrieval is optimal**

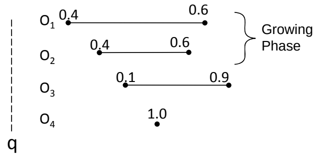


Figure 18: D_2**: Min-dist retrieval is not optimal**

Similarly, assume another upper-bound $\acute{up}(O_i) < up(O_i)$, where $up(O_i)$ is given by Equation 6, for an object $O_i \in \acute{\mathcal{O}}$. Assume that all non-retrieved objects have their minimum distance to q greater than $\overline{d}(O_i, q)$. Hence, $P_{nn}(O_i, q) = up(O_i)$. Hence, $\acute{up}(O_i)$ is an incorrect bound.

B.2 Proof of Theorem 3.2

Assume that $\mathcal{A}$ and IO-Centric have both retrieved the same objects up to some object O_l. Assume that IO-Centric has next retrieved the object O_{l+1}, while $\mathcal{A}$ has next retrieved an object different from O_{l+1}. Since $\mathcal{A}$ cannot rule out the possibility that O_{l+1} might be a deterministic point with probability 1 located at distance $\underline{d}(O_l, q)$ from q, $\mathcal{A}$ cannot increase the lower-bounds of P_{nn} values computed at the point O_l is retrieved, otherwise incorrect bounds would be assumed. Consequently, $\mathcal{A}$ cannot also change the upper-bound on the P_{nn} value of ϕ computed at the point O_l is retrieved. Hence, $\mathcal{A}$ cannot terminate before retrieving O_{l+1}.

B.3 Proof of Theorem 3.3

Let D_1 be a database instance where, after ending the growing phase, the first non-retrieved object in min-dist order, O_l, is a deterministic point with probability 1. Hence, retrieving O_l leads to direct query termination, since all candidates would have exact P_{nn} values based on Equations 5 and 6. Any other retrieval order in D_1 leads to query termination using at least one object, while min-dist order leads to termination using exactly one object O_l.

The database instance D_2 can be constructed by adjusting the PDFs of non-retrieved objects such that retrieving an object out of min-dist order leads to direct termination of the query by sufficiently shrinking the $up(.)$ values of candidates, while retrieving the next object in min-dist order has a negligible effect on the candidates P_{nn} bounds, and hence is not enough for query termination.

We illustrate the proof of Theorem 3.3 using the following example. Consider a database instance D_1, shown in Figure 17 where objects have 1-dimensional uncertain attributes. Assume that we break ties by favoring objects with smaller identifiers. The growing phase ends after retrieving O_1 and O_2 ($P_{nn}(O_1, q) \in [0.4, 0.4], P_{nn}(O_2, q) \in [0.24, 0.6]$). In D_1, the first non-retrieved object in min-dist order, O_3, is a deterministic point with probability 1. Retrieving O_3 leads to direct query termination, since all

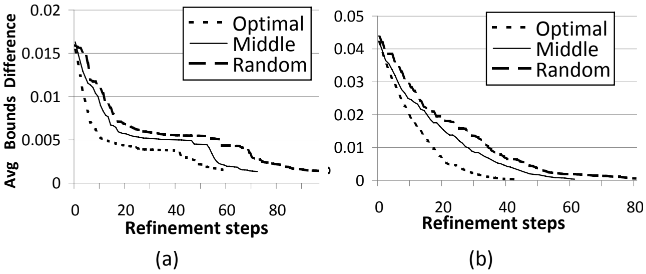


Figure 19: Convergence Rate (a) non-skewed distribution (b) negatively skewed distribution

candidates would have exact P_{nn} values (i.e., $P_{nn}(O_1, q) \in [0.4, 0.4], P_{nn}(O_2, q) = [0.24, 0.24]$). Any other retrieval order in D_1, i.e. retrieving O_4, does not lead to immediate terminating ($P_{nn}(O_1, q) = [0.4, 0.4], P_{nn}(O_2, q) \in [0.24, 0.564]$). On the other hand, consider the database instance D_2 as shown in Figure 18. Similar to D_1, the growing phase ends after retrieving O_1 and O_2. Retrieving an object in min-dist order, i.e. O_3, does not lead to termination ($P_{nn}(O_1, q) = [0.4, 0.4], P_{nn}(O_2, q) \in [0.24, 0.564]$). However, retrieval of O_4 leads to direct termination ($P_{nn}(O_1, q) = [0.4, 0.4], P_{nn}(O_2, q) = [0.24, 0.24]$).

B.4 Proof of Theorem 3.6

Assume that $\mathcal{P}_i^2$ is obtained by replacing $R_{ij} \in \mathcal{P}_i^1$ with $\{R_{ij.l}; l = 1 \dots m\}$. It follows that $\overline{d}(R_{ij}, q) \geq \overline{d}(R_{ij.l}, q)$. Therefore, $F_k(q, \overline{d}(q, R_{ij})) \leq F_k(q, \overline{d}(q, R_{ij.l}))$ for any object $O_k \neq O_i$. Based on Equation 9 and since $\sum_{R_{ij.l}} \Pr(R_{ij.l}) = \Pr(R_{ij})$, It follows that (1) is true.

Similarly, $\underline{d}(R_{ij}, q) \leq \underline{d}(R_{ij.l}, q)$. Therefore, $F_k(q, \underline{d}(q, R_{ij})) \geq F_k(q, \underline{d}(q, R_{ij.l}))$ for any object $O_k \neq O_i$. Based on Equation 10, it follows that $\sum_{R_{ij} \in \mathcal{P}_i^2} \overline{P_{nn}}(R_{ij}, q) \leq \sum_{R_{ij} \in \mathcal{P}_i^1} \overline{P_{nn}}(R_{ij}, q)$. In addition, the value of $1 - \sum_{j \neq i} \underline{P_{nn}}(O_j, q|\mathcal{P}_j))$ does not change when object O_i is partitioned. Therefore, based on Equation 11, (2) is true.

C. EVALUATING REFINING HEURISTICS

Our lazy bound refinement procedure refines object's partition by selecting the subregion with the highest rank and splitting it at its middle distance to q. We show here that the convergence rate of this heuristic to the exact integral value is comparable to the optimal refinement method. The optimal split location is found by conducting an exhaustive search over all possible split locations, and picking the location that results in tightening $P_{nn}(.)$ bounds the most. We additionally compare to a randomized strategy that splits a subregion at a random point.

Figure 19 shows the convergence of the three methods to the exact integral value with different data configurations. We plot the average width of the intervals that represent the $P_{nn}(.)$ bounds of all objects against the number of refinement steps. We use `CPU-Centric` to study the convergence, since we focus only on the efficiency of computation. The convergence rates of all methods are noticeably close. The convergence rate of our middle-distance heuristic is better than the randomized heuristic. Optimal refinement leads to the smallest number of steps where each step is much expensive than both heuristics. Thus, the overall cost of the optimal refinement is actually much higher, which makes using the optimal refinement unjustifiable as other heuristics provide close convergence rate at a significantly smaller cost. For example, with Normal distributions, middle-distance heuristic terminates in 367 seconds, while the optimal refinement terminates in 3092 seconds.