Evaluating Probability Threshold k-Nearest-Neighbor Queries over Uncertain Data

Reynold Cheng
The University of Hong Kong
ckcheng@cs.hku.hk

Lei Chen
Hong Kong Univ. of Sci. & Tech.
leichen@cse.ust.hk

Jinchuan Chen
Hong Kong Polytechnic University
csjcchen@comp.polyu.edu.hk

Xike Xie
The University of Hong Kong
xkxie@cs.hku.hk

ABSTRACT
In emerging applications such as location-based services, sensor monitoring and biological management systems, the values of the database items are naturally imprecise. For these uncertain databases, an important query is the Probabilistic k-Nearest-Neighbor Query (k-PNN), which computes the probabilities of sets of k objects for being the closest to a given query point. The evaluation of this query can be both computationally- and I/O- expensive, since there is an exponentially large number of k object-sets, and numerical integration is required. Often a user may not be concerned about the exact probability values. For example, he may only need answers that have sufficiently high confidence. We thus propose the Probabilistic Threshold k-Nearest-Neighbor Query (T-k-PNN), which returns sets of k objects that satisfy the query with probabilities higher than some threshold T. Three steps are proposed to handle this query efficiently. In the first stage, objects that cannot constitute an answer are *filtered* with the aid of a spatial index. The second step, called *probabilistic candidate selection*, significantly prunes a number of candidate sets to be examined. The remaining sets are sent for *verification*, which derives the lower and upper bounds of answer probabilities, so that a candidate set can be quickly decided on whether it should be included in the answer. We also examine spatially-efficient data structures that support these methods. Our solution can be applied to uncertain data with arbitrary probability density functions. We have also performed extensive experiments to examine the effectiveness of our methods.

1. INTRODUCTION
Uncertainty is inherent in many emerging applications. In the Global-Positioning System (GPS), for example, the location values collected from the mobile devices have measurement errors and it is difficult to remove them due to the lacking of domain knowledge [1, 2]. As another example, consider a habitat monitoring system where data like temperature, humidity, and light intensity are acquired from sensors. Due to the impreciseness of sensing devices, the data obtained are often noisy [3]. In biometric and biological databases, the attribute values of the extracted feature vectors are again not perfect [4, 5] due to the limitation of extraction methods. Recent works also propose to introduce a controlled amount of uncertainty to a user's location data, as a means of reducing resource utilization [1, 6] or improving the user's location privacy [7, 8]. To deal with the increasing needs of managing data uncertainty and providing high-quality services, researchers have recently proposed the use of "uncertain databases", where uncertainty is treated as a "first-class citizen". In particular, these uncertain data are evaluated by *probabilistic queries*, which produces answers with probabilistic and statistical guarantees [9, 6, 10, 11].

A widely-used data model assumed by uncertain databases is the *attribute uncertainty*, where the actual attribute value is located inside a closed area, or the *uncertainty region*. A non-zero probability density function (*pdf*) is associated with the uncertainty region, such that the integration of pdf inside the region equals to one. Figure 1(a) shows that in a location-based service, the uncertainty of a moving object's location can be treated as a normalized Gaussian distribution [1, 2]. The uncertainty region is a circular area, with a radius called the "distance threshold". The newest location is reported to the system when it deviates from the old one by more than this threshold (Figure 1(a)). Gaussian distributions are also used to model values of a feature vector in biometric databases [4]. Figure 1(b) shows the histogram of temperature values recorded by a sensor network deployed in a geographical area observed in a week. The pdf, as a histogram, depicts an arbitrary distribution between 30^oF and 40^oF.

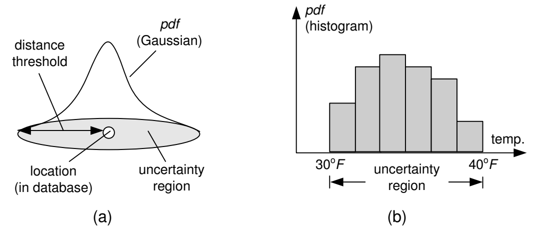


Figure 1: Location and sensor uncertainty.

In this paper, we study the *Probabilistic k-Nearest Neighbor Query* (k-PNN) for databases with attribute uncertainty. This query returns the non-zero probability (called *qualification probability*) of each set of k objects for being the nearest neighbor of a given point q. Given an uncertain database D of n uncertain objects, where $D = \{o_1, ..., o_n\}$, the k-PNN query can be defined as follows:

DEFINITION 1.1. *A* **Probabilistic k-NN Query (k-PNN)** *returns a list of answers $\{(S, p(S))\}$, where S is a subset of D of cardinality k, and $p(S)$ is the non-zero probability that S consists of the k nearest neighbors of q.*

Figure 2 shows an example of k-PNN, evaluated over eight uncertain objects ($o_1, \ldots, o_8$). If $k = 3$, then the query returns a set of 3-ary tuples, together with their chances for satisfying the query. In this example, $\{o_1, o_2, o_5\}$ and $\{o_1, o_2, o_3\}$ have qualification probabilities of 0.05 and 0.3 respectively. Notice in this definition, the number of k-subsets that satisfy the query may be exponential, and it may be necessary to have additional constraints (e.g., return objects whose probabilities are higher than some threshold) in order to limit the size of the answer set.

The k-PNN can be considered as a version of the k-nearest neighbor query (k-NN) evaluated on uncertain data. The k-NN query has been widely used in different applications, including location-based services [12], natural habitat monitoring [3], network traffic analysis [13], knowledge discovery [14], and CAD/CAM systems [15]. For example, in mobile e-commerce, a driver is supplied with the location information of the nearest gas stations. A CAM system uses k-NN queries to discover similar patterns over multi-dimensional data obtained from sensors installed in production lines [15]. A k-NN query can also be used to answer other ranking queries, such as k-min and k-max queries. For one-dimensional data, a k-min (k-max) query can be considered as a k-NN query by setting q to $-\infty$ (respectively $+\infty$). Such queries can be used in scientific monitoring applications to answer questions like: "What are the k bird nests that yield the highest temperature?" [3].

Most works about k-NN queries assume that the data being queried is precise. However, as we can see from the applications mentioned before, data on which k-NN is evaluated (e.g., locations of moving objects and sensor values) are often imprecise. To our best knowledge, few researches (e.g., [16, 5]) have studied the evaluation of k-NN queries over uncertain data. Therefore, our goal is to investigate efficient methods for evaluating k-NN queries for these databases.

Computing a k-PNN is usually more complex than its precise counterpart. Consider, for example, the computation of the probability that $\{o_1, o_2, o_5\}$ are the three closest neighbors to q in Figure 2. Since each object's value is not exactly known, we need to consider the values in its uncertainty region. Moreover, the qualification probability of $\{o_1, o_2, o_5\}$ depends not just on the three objects' values, but also on the relative values of other objects (e.g., o_3). If there is a chance that two objects have the same distance from q (e.g., o_2 and o_3), then their pdfs must be considered in order to derive the probabilities. The problem is further aggravated by the large number of combinations of objects. For example, for a 3-PNN evaluated over eight objects in Figure 2, we may have to compute the probabilities for $C_3^8 = 56$ possible answers. The number of answers that satisfy the query can also be exponential. Clearly, we need better semantics and methods to handle this query.

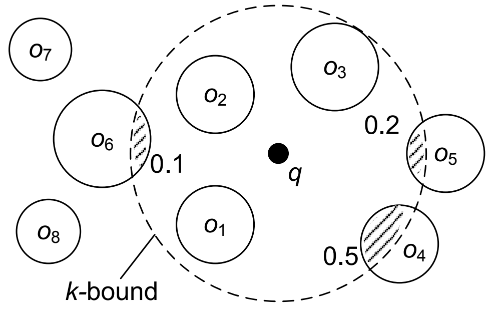


Figure 2: Probabilistic k-NN Query (k-PNN) with $k = 3$.

1.1 Solution Overview

We observe that a user may not always be interested in getting the precise probability values. He may only require answers with confidence that meets some predefined condition. For example, a user may only require answers with confidence higher than some fixed value. In Figure 2, for instance, if an answer with at least 20% probability is needed, then the sets $\{o_1, o_2, o_3\}$ and $\{o_1, o_2, o_4\}$ would be the only answers. We term the variant of k-PNN with a probability threshold constraint, T (e.g., 20%), as the *Probability Threshold k-Nearest-Neighbor Query* (or T-k-**PNN** in short). The threshold constraint allows the user to control the desired confidence required in a query answer. In Figure 2, for example, a 0.2-3-PNN returns $\{o_1, o_2, o_3\}$ and $\{o_1, o_2, o_4\}$ as the query answer. Such a query answer also allows a user to extract some useful information (e.g., o_1 and o_2 appear in both 3-subsets in this example). Notice that with a moderate value of T, the number of k-subsets returned is quite small in practice. For instance, in our experiments, at $T = 0.1$, two k-subsets are returned on average.

Moreover, we present three methods to efficiently process a T-k-PNN query. The first method, called *k-bound filtering*, effectively removes all objects that have no chance to be a query answer. Let us consider Figure 2 again, which shows the "k-bound" (as a dotted circle centered at q) that completely encloses the three objects o_1, o_2 and o_3. The radius of the 3-bound is defined as the third minimum of the maximal distances of the objects from q (in this example, the maximum distance of o_3 from q). With the k-bound, objects o_7 and o_8 can be pruned immediately, since they have no chance to overtake any of the objects o_1, o_2 or o_3 to become part of the answer to the 3-PNN query. Generally, with the k-bound, a lot of objects can be removed, and as we show in the paper, its usage can be easily leveraged to a spatial index (e.g., R-tree). For convenience, we call the objects that are not pruned by the k-bound filtering (i.e., those that overlap the k-bound) the *candidate objects*.

After k-bound filtering, we still need to consider the k-subsets of the candidate objects. In Figure 2, for instance, $C_3^6 = 20$ of sets of cardinality 3 may need to be considered. To further reduce the search space, we propose the second method, namely *Probabilistic Candidate Selection* (or **PCS**), which can efficiently detect k-subsets (i.e., subsets of database D with cardinality k) whose qualification probabilities are less than T, also called *unqualified k-subsets*. While k-bound filtering utilizes distance information for pruning, the PCS makes use of the probability information of uncertain data to remove unqualified k-subsets. The rationale behind PCS is that given the probability of a candidate object that lies within the k-bound (called *cutoff probability*), the qualifica-

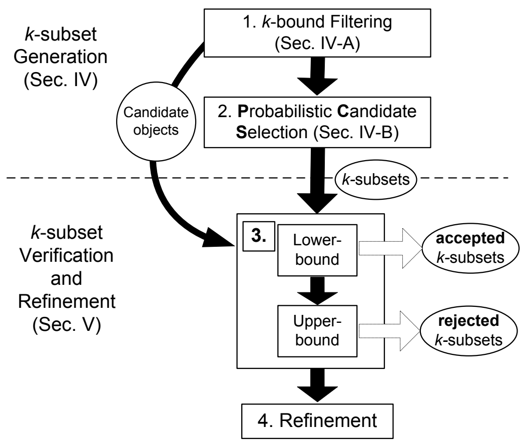

k-subset Generation (Sec. IV)
1. k-bound Filtering (Sec. IV-A)
Candidate objects
2. Probabilistic Candidate Selection (Sec. IV-B)
k-subsets
k-subset Verification and Refinement (Sec. V)
3.
Lower-bound
accepted k-subsets
Upper-bound
rejected k-subsets
4. Refinement

Figure 3: Solution Framework of T-k-PNN.

tion probability of a k-subset must be lower than the product of the cutoff probabilities of its subsets. In Figure 2, for example, the cutoff probabilities of o_4, o_5, and o_6 are 0.5, 0.2 and 0.1 respectively (shown as the shaded area). The qualification probability of the 3-subset $\{o_2, o_4, o_5\}$ must be lower than the product of the cutoff probabilities of $\{o_4, o_5\}$, or $0.5 \times 0.2 = 0.1$. If $T = 0.2$, then $\{o_2, o_4, o_5\}$ can be pruned. Based on this useful fact, the PCS algorithm constructs k-subsets by growing the list of the i-subsets with respect to i (where $i = 1, 2, .., k$). At each iteration i, the product of the cutoff probabilities of each i-subset are checked on whether it is less than T, and the i-subset is pruned if that is true. In addtion, we also design a technique, called "seed-pruning", to further improve the performance of PCS by removing the unqualified k-subsets with the "seeds" – objects that are located within the k-bound. Moreover, an efficient data compression method that suppresses the amount of intermediate storage overhead required by PCS is presented. Our experiments show that PCS reduces a significant portion of k-subsets to be examined.

The third method, called *verification*, is useful for handling k-subsets that are not filtered by the previous two methods. This technique determines whether a k-subset is a query answer, by making use of the uncertainty pdf of objects returned by k-bound filtering. We propose two kinds of verification: *lower-bound* and *upper-bound* verification, which quickly computes the lower and upper bounds of qualification probabilities of k-subsets. These bounds can then be used to determine how the k-subset should be handled. For example, a k-subset can be removed if its upper bound probability is smaller than T; it should be included in the query answer if its lower bound probability is higher than T. We will show the detailed design, complexity analysis, as well as correctness proofs for these methods.

Figure 3 depicts the framework of our solution, which consists of four steps. First, the k-bound filtering removes objects that must not be part of the k-nearest neighbor of q. All candidate objects are then passed to the PCS, which derives k-subsets based on the cutoff probability information. Next, the lower (upper) bounds of the qualification probabilities of the k-subsets are used to accept (reject) the k-subsets. Those that still cannot be determined are sent for *refinement*, whose exact probabilities are computed. While refinement is expensive, it can utilize the information generated during verification. Thus this is still faster than computing the qualification probability of k-subsets directly.

To summarize, we propose a computationally- and I/O- efficient solution for evaluating a T-k-PNN query. Our solution reduces I/O overhead by using a spatial index (e.g., R-tree) to prune away a large number of objects. To alleviate the large computational overhead, we propose PCS for reducing the number of k-subsets to be examined, as well as verification/refinement for avoiding exact probability computation. We propose a framework to connect these techniques in order to provide an efficient solution. We further investigate storage-efficient data structures to support our solution. Our experiments show that our approach can significantly improve the performance of query evaluation. For example, at $T = 0.1$ and $k = 5$ the time required by our method is only 1.6% of the time needed by calculating qualification probabilities directly.

The rest of this paper is organized as follows. We discuss the related work in Section 2. In Section 3, we present the formal semantics of the T-k-PNN, and our solution framework. Section 4 explains the filtering and the probabilistic candidate selection process. The details of verification and refinement are developed in Section 5. We present the experimental results in Section 6. The paper is concluded in Section 7.

2. RELATED WORK

Recently, a few uncertain database prototypes (e.g., [17, 18, 19, 20, 3]) have been developed. In these systems, two major classes of uncertainty models are assumed: tuple- and attribute-uncertainty. Tuple-uncertainty records the probability that a given tuple is part of a relation [17]. Attribute-uncertainty represents the inexactness of an attribute value as an uncertainty region and a pdf bounded in the region [1, 2, 6, 5, 4]. A formal database model for combining tuple and attribute uncertainty has also been proposed [21]. In this paper, we use the attribute uncertainty model.

A number of studies have been focused on the evaluation of Probabilistic Nearest-neighbor (PNN) queries on attribute uncertainty. A PNN, which can be regarded as a 1-PNN (in Definition 1.1), returns the probability of a single object for being the closest to a given query point q. In [22], an R-tree-based indexing solution for PNN has been presented. In this paper, we develop an indexing solution (called k-bound filtering) for k-PNN. In [6, 22], qualification probabilities of objects for satisfying a PNN are obtained by transforming the uncertainty of each object into two functions: pdf and cdf of an object's distance from the query point. They show how this conversion can be done for 1D uncertainty (intervals) and 2D uncertainty (circle and line). The qualification probabilities are then derived by evaluating an integral of an expression that involves distance pdfs and cdfs of multiple objects. We show how the computation of k-PNN can be performed using distance pdfs and cdfs. Another method for evaluating a PNN is proposed in [23], where each object is represented as a set of points sampled from the object's continuous pdf. More recently, the probability that an object exists in the database (called *existential probability* is used to derive lower and upper bounds and pruning for nearest-neighbors [16, 24]. In [25], the authors address how to efficiently retrieve data objects that have the minimum aggregate distance from a set of query points.

In order to improve the evaluation of qualification probabilities for 1-PNN, [26] has proposed a variant of 1-PNN that uses probability threshold as an answering criterion, and has developed efficient verification methods for deriving lower and upper bounds of an object's qualification probabilities. These methods are not readily used by a k-PNN (with $k \geq 1$) for three reasons. First, the evaluation of k-PNN faces the additional problem of examining a large number of k-subsets. To handle this problem, we develop new methods to significantly reduce the number of candidate k-subsets Secondly, the probability bound verification is designed for 1-PNN queries only. We develop new lower/upper bound computation methods for k-PNN queries. Thirdly, the solution of [26] can only be used to handle distance pdfs of the candidate objects represented as arbitrary histograms. Our techniques, on the other hand, are not restricted to histogram pdfs.

To our best knowledge, few work has addressed k-NN queries over uncertain data. Soliman et al. [16] proposed a query that ranks the probability each object is the nearest neighbor of q, and returns the k objects with the highest probabilities. Notice that the ranking criterion is based solely on each object's probability of being the nearest-neighbor of q. This is not the probability that all objects returned in the query answer are the k nearest neighbors of q (by considering all the possible worlds). In other words, the k object answer returned by [16] may not appear in the same possible world. On the other hand, the query studied in this paper is a "true" k-nearest-neighbor query, where we consider the probability that a set of objects are the k nearest neighbors of q. In [5], Ljosa et al. proposed an efficient index structure, called APLA-tree, for evaluating k-NN queries. They used the expected distance (under L1-norm) of an object's uncertainty pdf from q as a ranking criterion. Thus, their k-NN query is based on the expected distance, and does not have probabilities in their answers.

The evaluation and indexing methods for other probabilistic queries on attribute uncertainty have been studied. This includes range queries [27], location-dependent queries [7], skyline queries [28, 29], and top-k queries [30, 31]. The issue of uncertainty have also been considered in the domain of biometric databases [4] and access control [32]. More recently, the issues of conditioning and cleaning a probabilistic database have been studied [33, 34].

3. PRELIMINARIES

We now present the semantics of the T-k-PNN query (Section 3.1). Then we explain a simple solution for this query (Section 3.2).

3.1 Definition of T-k-PNN

Let $p(S) \in [0, 1]$ be the probability that the elements of a k-subset S are the k nearest neighbors of query point q (i.e., qualification probability). Then, a T-k-PNN can be defined as follows.

DEFINITION 3.1. *A* **Probability Threshold k-NN Query (T-k-PNN)** *returns a set S, such that $\{S | S \subseteq D \wedge |S| = k\}$ and $p(S) \geq T$, where $T \in (0, 1]$.*

We call T the *threshold* parameter. A k-subset S is allowed to be returned as an answer if its qualification probability is not less than T. Compared with k-PNN (Definition 1.1), this query does not return the actual qualification probability of S to the user. Also, we can further use other constraints (e.g., the maximum number of answers) to limit the number of k-subsets returned to the user. Table 1 summarizes the symbols used in the definition of T-k-PNN.

3.2 Basic Evaluation of T-k-PNN

Symbol	Meaning
D	Uncertain database
o_i	Uncertain object i of D ($i = 1, \ldots, \lvert D\rvert$)
r_i	$\lvert o_i - q\rvert$
$d_i(r)$	pdf of r_i (*distance pdf*)
$D_i(r)$	cdf of r_i (*distance cdf*)
q	Query point
T	Probability Threshold
S	$\{o_i \lvert o_i \in D\}$
$p(S)$	Qualification prob. of S

Table 1: Symbols for T-k-PNN.

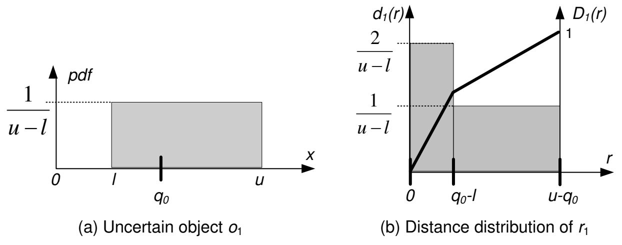


Figure 4: Distance pdf and cdf

Let us now present a simple solution for answering the T-k-PNN, which forms the basis for further discussions. This method utilizes the probability distribution of each object's distance from q. Formally, let $r_i \in \Re$ be the absolute distance of an uncertain object o_i from q. That is, $r_i = |o_i - q|$. We assume that r_i takes on a value $r \in \Re$. Then, the distance pdf and cdf of o_i are defined below [6, 22].

DEFINITION 3.2. *Given an uncertain object o_i, its* **distance pdf**, *denoted by $d_i(r)$, is a pdf of R_i; its* **distance cdf**, *denoted by $D_i(r)$, is a cdf of R_i.*

We use an example to illustrate these two functions. Figure 4(a) shows a one-dimensional uncertain object, o_1, which has a uniform pdf of value $\frac{1}{u-l}$ in uncertainty region $[l, u]$. A query point, q_0, is also shown. In Figure 4(b), the distance pdf (shaded) of $r_1 = |o_1 - q_0|$, ranging from zero to $u - q_0$, is illustrated. The distance pdf in $[0, q_0 - l]$ is the sum of the pdf on both sides of q_0, which is equal to $\frac{2}{u-l}$. In $[q_0 - l, u - q_0]$, the distance pdf of r_1 is $\frac{1}{u-l}$. The distance cdf of r_1, drawn as a solid line, is found by integrating the distance pdf.

For the detailed procedures of deriving distance pdf and cdf, readers are referred to [26] (for 1D arbitrary histogram uncertainty pdf) and [22] (for 2D circle and line segment uncertainty regions with uniform pdf).

Based on the uncertainty pdf and cdf of each object, the qualification probability of a k-subset S (i.e., $p(S)$) can be computed. The probability, $p(S)$, is then used to compare against T; if $p(S) \geq T$, then S becomes an answer. Now let us take a look at how $p(S)$ is computed:

$$p(S) = \sum_{o_i \in S} \int_0^{+\infty} d_i(r) \prod_{o_j \in S \wedge o_j \neq o_i} D_j(r) \prod_{o_h \in D-S} (1 - D_h(r))dr \quad (1)$$

To understand Equation 1, observe that for S to be a query answer, the distance of any object o_h (where $o_h \notin S$) from q must be greater than that of o_i (where $o_i \in S$). Now, at distance r, the pdf that object $o_i \in S$ has the k-th shortest distance from q is the product of the following factors:

- the pdf that o_i has a distance of r from q, i.e., $d_i(r)$;
- the probability that all objects in S other than o_i have shorter distances than r, i.e., $\prod_{o_j \in S \wedge o_j \neq o_i} D_j(r)$; and
- the probability that objects in $D - S$ have longer distances than r, i.e., $\prod_{o_h \in D-S} (1 - D_h(r))$.

The integration function in Equation 1 is essentially the product of the above three factors. By integrating this function over $(0, +\infty)$, we obtain the probability that S contains the k nearest neighbors with o_i as the k-th nearest neighbor. Finally, by summing up this probability value for all objects $o_i \in S$, Equation 1 is obtained.

Equation 1 is inefficient to evaluate. First, the distance pdf and cdf of each object has to be computed. Secondly, Equation 1 involves costly numerical integration, and has to be performed over a large range. Thirdly, the probability of each k-subset S has to be computed, and the number of these k-subsets is exponential. However, we found that T-k-PNN can be handled in a better way. Specifically, later, in Section 4, we exploit the extent of the objects' uncertainty regions and probability threshold to significantly prune the number of k-subsets to be examined. Then, in Section 5, we derive the lower and upper bounds of k-subsets' qualification probabilities, so that the decision of whether a k-subset should be accepted as a query answer can be made without computing its actual probability.

4. GENERATING K-SUBSETS

In this section, we examine two efficient methods for generating k-subsets that can potentially satisfy the T-k-PNN queries. Section 4.1 discusses the design and implementation of k-bound filtering. We then present the Probabilistic Candidate Selection (PCS) process in Section 4.2. We investigate a spatially-efficient compression method for supporting PCS in Section 4.3.

4.1 k-bound Filtering

Given a query point, a naive solution is to enumerate all possible combinations of sets of size k from the uncertain database D and compute their probabilities according to Equation 1. If the probability is greater than T, the set will be returned as a result. Clearly, this is an inefficient approach with respect to computation and I/O costs. In fact, given a T-k-PNN, we can first utilize the distance information of the objects to prune those that are not qualified for the answers. Specifically, we propose an efficient filter based on the k-th minimum of maximal distance f_k, called k-bound filter, to remove objects that have zero probability to be the answers of T-k-PNN. The rationale behind the k-bound filter is stated in the following Lemma.

LEMMA 4.1. *Given an object $o_i \in D$, a query point q, if $min(r_i) > f_k$, then o_i will not appear in any answer data set, where f_k is the k^{th} minimum of maximal distance (k-bound) among all r_j's for $(j = 1, ..., n)$, and r_i is the distance between q and o_i.*

PROOF. If $min(r_i) > f_k$ holds, o_i cannot belong to the k nearest neighbors of q since there always exist at least k objects with distances smaller than or equal to f_k. Therefore, no answer set S contain such an o_i according to the T-k-PNN definition. In other words, the fact that S is the k nearest neighbors of q implies $min(r_i) \leq f_k (\forall o_i \in S)$. □

Lemma 4.1 offers a filtering method by that uses k-bound as illustrated in Figure 2. In particular, with k-bound filtering, objects o_7 and o_8's minimum distances to q are larger than the bound, and so they can be excluded for further consideration. The rest of the objects that overlap with or are contained in the k-bound are used to generate k-subsets. Another advantage of k-bound filtering is, after filtering, probability computation is easier (compared to Equation 1), since the integration range $[0, \infty]$ is reduced to $[0, f_k]$ for numerical integration.

The performance of k-bound filtering can be sped up with the help of a spatial index structure. In this work we index the uncertainty region of each data object in the R-tree [35], on which the k-bound filtering can be conducted. The reason we choose R-tree is due to its popularity. However other spatial index structures can also be used. The R-tree recursively groups uncertain data objects with *minimum bounding rectangles* (MBRs) until one final node (root) is obtained. The process of filtering over R-tree is detailed in Algorithm 1 (k-bound_Processing). The algorithm maintains a minimum heap $\mathcal{H}$ which contains the entry of form (v, key), where key is the $min(dist(q, v))$. $\mathcal{H}$ is first initialized (line 1). The candidate object set is emptied and f_k is set as infinity (lines 2 and 3). Then, the root node is loaded and stored in $\mathcal{H}$ (line 4). Each time we pop out an entry (v, key) from heap $\mathcal{H}$ (line 6), and check whether key is smaller than f_k (line 7). If the answer is negative, this entry is discarded (Lemma 4.1). Otherwise, we then check whether v is a leaf node (line 8). If the answer is yes, we insert this founded candidate object into C (line 9). If v is an intermediate node, for each entry v_i in v, we compute its minimum distance from q (lines 11 and 12). If this minimum distance is also smaller than f_k, we insert v_i into the heap $\mathcal{H}$ and update f_k if necessary (lines 13 to 15). This process repeats until the queue is empty.

4.2 Probabilistic Candidate Selection

After k-bound filtering, assume we obtain $m \in [k, n]$ objects, i.e. $C = \{o_1, ..., o_m\}$, there could still be C_k^m possible k-subset answers. Directly computing these answer sets will result in exponential cost in both memory and computation. In fact, it is not necessary to generate all the k-subsets. In this section, we propose a candidate set generation method based on the probability information, namely *probabilistic candidate selection* (PCS). Specifically, we make use of the probability of an object that lies within the k-bound, called *cutoff probability* (CP) (shown in Figure 5(a)) and the fact that the qualification probability of a k-subset must be less than the product of the cutoff probabilities of its members. The following lemma states this fact.

LEMMA 4.2. $p(S) \leq UBProb(S')$, $\forall S' \subseteq S$,

where $UBProb(S') = \prod_{o_i \in S'} Pr(r_i \leq f_k)$

PROOF. According to Lemma 4.1, the fact "S contains k nearest neighbors" requires that all member objects of S to have distances from q not larger than f_k. That means $p(S)$ must not be larger than

```
input : R-tree $\mathcal{I}$ constructed over $D$
input : $q$: the query point
input : $k$
output: the candidate object set $C$ and $f_k$
1  initialize min-heap $\mathcal{H}$ accepting entries in the form
   $(v, key)$;
2  $C \leftarrow \emptyset$;
3  $f_k \leftarrow +\infty$;
4  insert $(root(\mathcal{I}), 0)$ into heap $\mathcal{H}$;
5  while $\mathcal{H}$ is not empty do
6      $(v, key) \leftarrow$ de-heap $\mathcal{H}$;
7      if $key < f_k$ then
8          if $v$ is an uncertain object then
9              insert $v$ into $C$;
10         else
11             for each entry $v_i$ contained in $v$ do
12                 compute $key \leftarrow min(dist(q, v_i))$;
13                 if $key < f_k$ then
14                     insert($v_i, key$) to $\mathcal{H}$;
15                     update $f_k$ according to $key$;
16 return $C$, $f_k$;
```
Algorithm 1: k-bound_Processing

$\prod_{o_i \in S} Pr(r_i \leq f_k)$, i.e. the product of the cutoff probabilities of its member objects. Since the cutoff probabilities are always smaller than or equal to 1, $UBProb(S')$ gives an upper bound of $\prod_{o_i \in S} Pr(r_i \leq f_k)$. Thus Lemma 4.2 is proved.

□

Based on the cutoff probability of each candidate object within the k-bound, the PCS algorithm constructs $k-$subsets by growing the list of i-subsets with respect to i (where $i = 1$ to $k-1$). The steps of PCS are listed in Algorithm 2. First, the algorithm generates 1-subsets based on the candidate set C (line 1). Then, the $(i+1)$-subsets are generated by unioning i-subsets and C (lines 2 to 13). The value of $UBProb(S')$ could be obtained by Lemma 4.2 (line 10). All those subsets with $UBProb(S')$ smaller than the threshold will be pruned (line 11). Therefore, many intermediate subsets are pruned, and the number of k-subsets will be greatly reduced. When we extend S to S' by adding o_j, and find that S' should be pruned, then it is no need to check the extensions with $o_{j+1}, ..., o_m$ (line 13), since the data objects are sorted in descending order of their cutoff probabilities.

Figures 5(a)-5(c) show an example of generating candidate $k-$subsets based on the cutoff probability of each candidate object within the k-bound. Figure 5(a) lists the cutoff probability (CP) of each object. We can safely remove candidate object o_6 since its CP is less than the threshold $T = 0.2$. Then, in the second round, as shown in Figure 5(b), the subset $\{o_4, o_5\}$ can be removed. Similarly, in the third round (Figure 5(c)) the candidate subsets $\{o_1, o4, o_5\}$, $\{o_2, o_4, o_5\}$ and $\{o_3, o_4, o_5\}$ can be safely removed.

In the previous discussions, in each round i, we have used CP and T to determine whether the generated i-subset should be kept for further extension. Here we propose an enhancement that utilizes the i-th minimum maximum distance (i.e., f_i) to further remove the

```
input : $C = \{o_1, ..., o_m\}$, $q$,$k$, $T$
output: $C_k$: the set of candidate object sets
1  $C_1 \leftarrow \{\{o_1\}, ..., \{o_m\}\}$;
2  for $i \leftarrow 1$ to $k-1$ do
3      $C_{i+1} \leftarrow \emptyset$;
4      for each $S \in C_i$ do
5          $z \leftarrow max_{o_i \in S}(i)$ ;
6          for $j \leftarrow z+1$ to $c$ do
7              if $o_j \notin S$ then
8                  $S' \leftarrow S \cup \{o_j\}$;
9                  if $S' \notin C_{i+1}$ then
10                     if UBProb($S'$)$\geq T$ then
11                         $C_{i+1}.add(S')$
12                     else
13                         break;
14 return $C_k$
```
Algorithm 2: Prob_Cand_Sel

unqualified subsets generated in each round. We can obtain these f_i values by slightly changing the Algorithm k-bound_Processing to return all the f_i values. Next, we suppose all candidate objects have been sorted in ascending order of their maximum distances from q. We put the objects with the k lowest values of maximum distance into an array called $seeds$, and derive the following lemma.

LEMMA 4.3. *If the lower bound of r_j of data object o_j is larger than f_i (i-th minimum maximum distance of $seeds[i]$), any k-subset S containing o_j cannot be the answer to the T-k-PNN if $\exists o_t \in \{seeds[1], ..., seeds[i]\}$ and $o_t \notin S$.*

PROOF. The fact that "the lower bound of r_j is larger than the f_i" implies that all objects in $\{seeds[1], ..., seeds[i]\}$ must have shorter distances than r_j. Therefore, if o_j happens to be inside a probabilistic k-nearest neighbors answer set, say S, all objects in $\{seeds[1], ..., seeds[i]\}$ must also be contained in S. □

Lemma 4.3 indicates that a qualified k-subset should contain some specific seeds to become a valid result. This rule can help us prune many unqualified subsets without estimating their qualification probabilities. The detailed steps are listed in Algorithm 3 (Seed_Pruning). In order to use seed pruning method to remove unqualified intermediate subsets generated in each round of PCS, we can invoke Algorithm Seed_Pruning immediately after line 4 of Algorithm Prob_Can_Sel.

4.3 A Storage-Efficient Compression Method

The PCS algorithm can be quite expensive in terms of memory consumption, since in each round i, we have to store all the i-subsets whose CPs are greater than T, and the number of such i-subsets could be exponentially large.

To reduce the memory cost we propose a simple but effective compression method. We first present our observations on the i-subsets generated by the PCS algorithm which forms the basis of our discussions. As shown in Figures 5(a)-5(c), the elements of the generated subsets in each round using Algorithm 2 ($Prob\_Cand\_Set$)

1-subset	CP
$\{o_1\}$	1
$\{o_2\}$	1
$\{o_3\}$	1
$\{o_4\}$	0.5
$\{o_5\}$	0.2
$\{\mathbf{o_6}\}$	**0.1**

(a) Round 1

2-subset	CP
$\{o_1, o_2\}$	1
$\{o_1, o_3\}$	1
$\{o_1, o_4\}$	0.5
$\{o_1, o_5\}$	0.2
$\{o_2, o_3\}$	1
$\{o_2, o_4\}$	0.5
$\{o_2, o_5\}$	0.2
$\{o_3, o_4\}$	0.5
$\{o_3, o_5\}$	0.2
$\{\mathbf{o_4, o_5}\}$	**0.1**

(b) Round 2

3-subset	CP
$\{o_1, o_2, o_3\}$	1
$\{o_1, o_2, o_4\}$	0.5
$\{o_1, o_2, o_5\}$	0.2
$\{o_1, o_3, o_4\}$	0.5
$\{o_1, o_3, o_5\}$	0.2
$\{\mathbf{o_1, o_4, o_5}\}$	**0.1**
$\{o_2, o_3, o_4\}$	0.5
$\{o_2, o_3, o_5\}$	0.2
$\{\mathbf{o_2, o_4, o_5}\}$	**0.1**
$\{\mathbf{o_3, o_4, o_5}\}$	**0.1**

(c) Round 3

Figure 5: Step-by-step generating candidate subsets based on CP

Size-1 Set	CP
$\{o_1\}$	1
$\{o_2\}$	1
$\{o_3\}$	1
$\{o_4\}$	0.5
$\{o_5\}$	0.2

(a) Round 1

Size-2 Set	CP
$\{o_1, o_5\}$	1
$\{o_2, o_5\}$	1
$\{o_3, o_5\}$	1

(b) Round 2

Size-3 Set	CP
$\{o_1, o_2, o_5\}$	1
$\{o_1, o_3, o_5\}$	1
$\{o_2, o_3, o_5\}$	1

(c) Round 3

Figure 6: Compressed candidate subsets based on CP

```
input  : $seeds$, $S$ and $f_i, \ldots, f_k$.
output: A boolean value indicating whether $S$ is a possible
        result.
1 for each $o_j \in S$ do
2     if $min(r_j) < f_1$ then
3         $\gamma \leftarrow 0$;
4     else
5         $\gamma \leftarrow$ the largest $i$ satisfying $min(r_j) \geq f_i$;
6     if $\gamma > 0$ then
7         if not $\{seeds[1], \ldots, seeds[\gamma]\} \subseteq S$ then
8             return False
9 return True
```

Algorithm 3: Seed_Pruning

are sorted in the descending order of their CPs. In addition, we can also find that many subsets of the same size share a common prefix. For example, in Figures 5(b), the first four 2-subsets share the common prefix $\{o_1\}$. Similarly, in Figure 5(c), the first three 3-subsets share the common prefix $\{o_1, o_2\}$. Based on these observations, we propose to compress the subsets of the same size that share a common prefix. Specifically, for the subsets of the same size, we store the common prefix of the subsets and the last element of the subset that has the minimum product of cutoff probability greater than T. We also call this element a *boundary element*. For example, given the first four 2-subsets in Figures 5(b), after compression, we only store $\{o_1, o_5\}$ as shown in the first entry of compressed storage in Figure 6(b). In this entry $\{o_1\}$ is the common prefix and o_5 is the last element of subset $\{o_1, o_5\}$, whose subset has the minimum product probability among first four 2-subsets in Figure 5(b). As shown in Figure 6(b), in addition to the compressed item $\{o_1, o_5\}$, we also store the product probability of the common prefix, here it is $\{o_1\}$'s CP, which is 1. Thus, in our compression scheme, for each compressed entry, we store the common prefix, the boundary element, and the product probability of the prefix. As another example, the first three 3-subsets in Figure 5(c) are compressed into $\{o_1, o_2, o_5\}$, as shown in Figure 6(c). The whole compressed entry is $\{\{o_1, o_2, o_5\}, 1\}$, where 1 is the product probability of prefix $\{o_1, o_2\}$. Figures 6(a)-6(c) show the whole compressed results of subsets in Figures 5(a)-5(c). Note that entries in bold fonts of Figures 5(a)-5(c) are unqualified entries.

Whenever it is necessary to decompress an compressed entry, we can generate the uncompressed subsets by appending all the possible elements starting from the *immediate successor* of the last element in the prefix to the bounding element. Let us use Figure 6(b) as an example. Given the compressed entry $\{\{o_1, o_5\}, 1\}$, the prefix is $\{o_1\}$, the bounding element is o_5, the immediate successor element of o_1 is o_2, as all the candidate objects are sorted in the descending order according to their CPs. Then the decompressed set is $\{o_1, o_2\}$, $\{o_1, o_3\}$, $\{o_1, o_4\}$, and $\{o_1, o_5\}$. The corresponding CP of each decomposed subset is the product of the compressed entry's CP (now is 1) and the appended element's CP. Similarly, for the compressed entry $\{\{o_1, o_2, o_5\}, 1\}$ shown in Figure 6(c), o_3 is the immediate successor of last element in prefix $\{o_1, o_2\}$, so we can generate $\{o_1, o_2, o_3\}$, $\{o_1, o_2, o_4\}$, and $\{o_1, o_2, o_5\}$, the corresponding CPs for these decompressed entries are: $1 * 1$, $1 * 0.5$, and $1 * 0.2$ respectively. Our experiments show that this compression scheme reduces the storage required in this phase significantly.

5. VERIFICATION AND REFINEMENT

Based on the k-subsets generated by PCS, we now present efficent techniques for handling the k-subsets. We discuss techniques for deriving lower and upper bounds of the qualification probabilities of k-subsets in Section 5.1. We then study how these techniques facilitate probability computation (in Section 5.2).

Symbol	Meaning
C	*candidate set*
Q	Set of k-subsets yielded by PCS
$[p(S).l, p(S).u]$	Lower & upper prob. bounds of $p(S)$
$m(S)$	$\max(\{r_i \mid o_i \in S\})$
e_j	The j-th end point
P_j	The j-th partition, where $P_j = [e_j, e_{j+1}]$
M	Total no. of partitions
$y_j(S)$	Prob$(m(S) \in P_j)$
$p_j(S)$	Qualification prob. of S, given $m(S) \in P_j$
$[p_j(S).l, p_j(S).u]$	Lower & upper bounds of $p_j(S)$

Table 2: Symbols used by verification.

```
input : Partition info. $X$, set $Q$ of $k$-subsets
output: set $A$ of answers, set $U$ of $k$-subsets to be refined
1 $A \leftarrow \emptyset$; $U \leftarrow \emptyset$;
2 for each $S \in Q$ do
3     if UB$(X, S) \geq T$ then
4         if LB $(X, S) \geq T$ then
5             insert $S$ into $A$;
6         else
7             insert $S$ into $U$;
8 return $A$, $U$
```

Algorithm 4: Verification.

5.1 Lower and Upper Bound Verification

Partitions. Let us first sort the set of objects C retained after the k-bound filtering in ascending order of their shortest distances from the query point q. For convenience, let us assume that $o_1, o_2, \ldots, o_{|c|}$ are sorted in this order. Figure 7(a) illustrates three distance pdfs with respect to q, where $k = 2$. As illustrated, the range between e_1 (i.e., the closest distance of all objects from q) and the point f_2 obtained from k-bound filtering (i.e., e_5) is subdivided into non-overlapping fragments called *partitions*. We call each fragment P_j, where P_j is embraced by two end-points, namely, e_j and e_{j+1}, circled in the figure. In this example, there are four partitions, e.g., $P_1 = [e_1, e_2]$, $P_2 = [e_2, e_3]$.

The partitions can have variable sizes. Thus, the system has flexibility in deciding the number of partitions to be used. The more partitions are defined, the more accurate will be the lower and upper verification, with the need of more overhead for storing the partition information and prolonging the verification process. We found that the verification performs well when the boundaries of the objects' distance pdfs are used as end-points.

The number above each range indicates the probability that an uncertain object has that range of distance from the query point. For each partition P_j of an object o_i, we evaluate the distance cdf of P_j's upper end-point (i.e., $D_i(e_{j+1})$). In general, the distance cdf of P_j's lower end-point is the same as that of P_{j-1}'s upper end point. For partition P_1, its lower end-point has a distance pdf of zero. Figure 7(b) illustrates the distance cdf values extracted from (a). For example, for r_1 in P_3, $D_1(e_4) = 0.8$. We store the values of $D_i(e_{j+1})$ in a two-dimensional array of dimensions $|C| \times M$, where M is the number of partitions, so that they can be accessed in $O(1)$ times. The time for sorting and initializing this array is $O(|C| \log |C| + M|C|)$. Table 2 lists the symbols used by verification.

Verification Process. We now demonstrate how partitions can be used to efficiently derive lower and upper bounds of each k-subset's qualification probability (i.e., $p(S)$). Let $[p(S).l, p(S).u]$ be the lower and upper bounds of $p(S)$. Let X be the data structure that stores the partition information, and Q be the set of k-subsets generated from the PCS algorithm. Given these inputs, the verification algorithm (Algorithm 4 judges whether a k-subset S should be considered as a query answer. It produces an answer set A (i.e., k-subsets that satisfy the query) and a refinement set U (i.e., k-subsets that need to be further investigated).

In Algorithm 4, Step 1 initializes the two sets, A and U, to empty sets. For every k-subset $S \in Q$, Step 3 uses subroutine UB to find the upper bound of S's qualification probability (i.e., $p(S).u$). If $p(S).u$ is smaller than T, then S cannot satisfy the query and is pruned. Next, subroutine LB is invoked to find $p(S).l$. If this value is not less than T, then S is inserted to the answer set (Steps 4,5). Otherwise, S is put into the set U for further processing (Step 7). Step 8 returns the sets A and U.

It is worth mention that in Step 3, we put UB *before* LB. This is because in our experiments, a large number of k-subsets can be pruned by UB (in Step 3). By testing the k-subsets with UB first, we avoid applying the LB test to the k-subsets, which have to be pruned anyway. We will revisit this issue in Section 6.

We now explain the design of LB and UB, which returns $p(S).l$ and $p(S).u$. Suppose $m(S)$ is the maximum distance between all objects in S and q, i.e., $\max(\{r_i \mid o_i \in S\})$. Let $y_j(m(S))$ be the probability that $m(S)$ is within the partition $P_j = [e_j, e_{j+1}]$. Let $p_j(S)$ be the qualification probability of $p(S)$, given that $m(S)$ lies in P_j. Let $[p_j(S).l, p_j(S).u]$ be the lower and upper bounds of $p_j(S)$. If there are M partitions, we have:

$$p(S).l = \sum_{j=1}^{M} p_j(S).l \cdot y_j(m(S)) \tag{2}$$

$$p(S).u = \sum_{j=1}^{M} p_j(S).u \cdot y_j(m(S)) \tag{3}$$

Moreover,

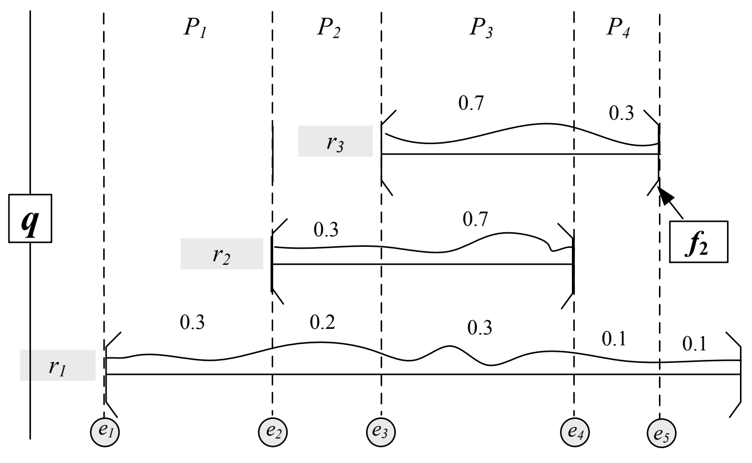
$$y_j(m(S)) = \prod_{o_i \in S} D_i(e_{j+1}) - \prod_{o_i \in S} D_i(e_j) \tag{4}$$

This is because the term $\prod_{o_i \in S} D_i(e_{j+1})$ is the probability that all objects in S have distance from q not larger than the end-point e_{j+1}. By subtracting $\prod_{o_i \in S} D_i(e_j)$ from it, we obtain the probability that at least one object is located inside $P_j = [e_j, e_{j+1}]$. This is also the chance that the maximum distance of all objects in S (i.e., $m(S)$) is within P_j, as shown in Equation 4. The following describes the formulas for $p_j(S).l$ and $p_j(S).u$.

LEMMA 5.1. *Given that $m(S) \in P_j$, the lower and upper bounds of qualification probabilities of k-subset, S, are:*

$$p_j(S).l \geq \prod_{o_i \in C-S} (1 - D_i(e_{j+1})) \tag{5}$$

$$p_j(S).u \leq \prod_{o_i \in C-S} (1 - D_i(e_j)) \tag{6}$$



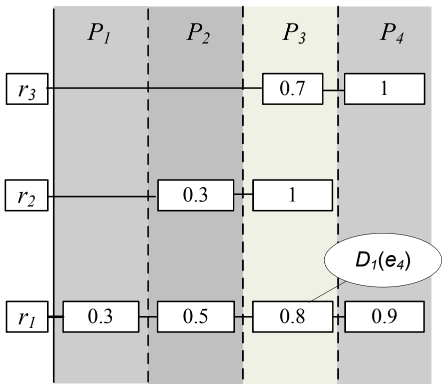


Figure 7: Illustrating the distance pdfs and partition probabilities (for $k = 2$).

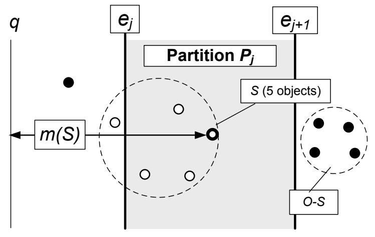


Figure 8: Correctness proofs for $p_j(S).l$ and $p_j(S).u$.

PROOF. **Equation 5:** Since the maximum distance of all objects in S from q is less than e_{j+1}, S must be the answer if all other objects in $C - S$ have distances more than e_{j+1}. For example, Figure 8 shows that S, a 5-subset, has a maximum distance ($m(S)$) not more than e_{j+1}. If the remaining objects ($C - S$) are on the right of e_{j+1} (circled), then S must constitute a query answer. The probability that this event happens is $\prod_{o_i \in O-S}(1-D_i(e_{j+1}))$ (the right side of Equation 5), which is also the lower bound of $p_j(S)$.

Equation 6: If any object in $C - S$ has distance from q shorter than e_j, then S could not be the set of k closest neighbors of q. In Figure 8, for example, since an object (colored black) is on the left of e_j, it is certainly closer to q then at least one object in S. So, S cannot be a query answer for $k = 5$. The event that all objects in $C - S$ have distance from q more than e_j is thus a precondition for S to be the query answer. The probability that this event happens, i.e., $\prod_{o_i \in O-S}(1 - D_i(e_j))$ (the right side of Equation 6), is therefore the upper bound of $p_j(S)$. $\square$

With Equation 4 and Lemma 5.1, the lower and upper bounds of $p(S)$ (i.e., Equations 2 and 3) can be estimated. If the partition data structure presented earlier is used, retrieving $D_i(e_j)$ (given i and j) needs $O(1)$ times. Evaluating Equation 4 thus needs $O(k)$ times. Computation of Equations 5 and 6 both requires $O(|C|)$ times. Thus the LB and UB functions have a complexity of $O(kM|C|)$. The total complexity of the verification algorithm is $O(kM|C||Q|)$.

5.2 Incremental Refinement

After verification, objects stored in the set U (Step 3 of Figure 4) require further processing, whose exact qualification probabilities need to be computed. This can be expensive, since numerical integration may be needed (see Equation 1). Interestingly, we can speed up this process with the information obtained during verification. This main idea is to treat the probability of an object as a sum of qualification probabilities inside partitions. By using the bound information of probabilities in each partition, the answer probabilities can be gradually computed.

Specifically, observe that the probability bounds of each k-subset S in each partition P_j (i.e., $[p_j(S).l, p_j(S).u]$) have been obtained during verification. For each P_j, once we get the value of $p_j(S)$ (by Equation 1), we can collapse $[p_j(S).l, p_j(S).u]$ into $p_j(S)$, update the probability bound of $p(S)$ (i.e., $[p(S).l, p(S).u]$), and test this new bound against the threshold T. This process is repeated for the next partition until we can decide whether S should be included in the answer. As shown in our experiments, "incremental refinement" is usually faster than computing probabilities directly, since performing numerical integration on a partition is faster than on $[0, f_k]$, which has a larger area of integration.

6. RESULTS

We have performed extensive experiments on a real data set to examine the effectiveness of our solution. We first describe the experimental setup in Section 6.1. Then we present the results in Section 6.2.

6.1 Experimental Setup

We use the *Long Beach* dataset[1] which includes 53,144 rectangles, distributed in the two-dimension space of $10K \times 10K$ units. Each rectangle is treated as an uncertainty region, with a uniform pdf as the default. We also perform experiments on Gaussian pdf (represnted as a histogram). For each T-k-PNN query, the default values of probability threshold (T) and k are 0.1 and 6 respectively. The query point is randomly chosen from the 2D space. Each data point

[1]Available at http://www.census.gov/geo/www/tiger/.

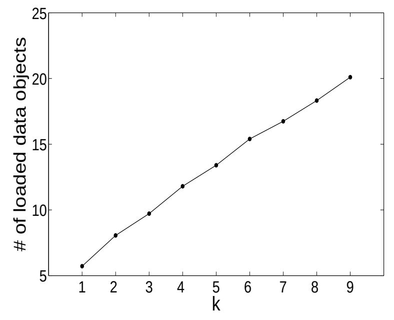


Figure 9: **# of Loaded Data Objects.**

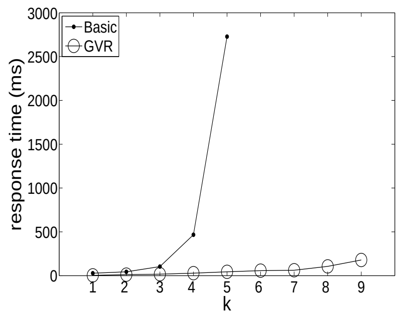


Figure 10: **Basic vs. GVR.**

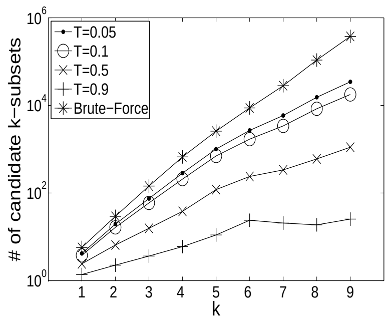


Figure 11: **Generating k-subsets.**

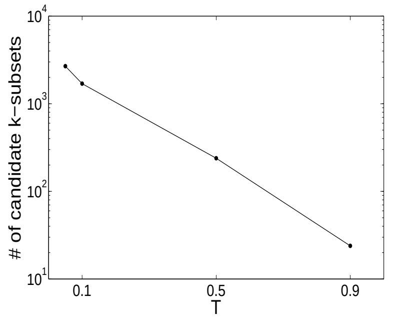


Figure 12: **Effect of T on PCS ($k = 6$).**

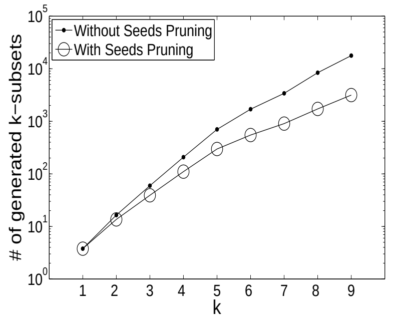


Figure 13: **Seed Pruning (# k-Subsets).**

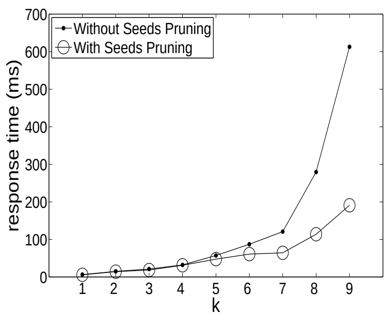


Figure 14: **Seed Pruning (Response Time).**

is an average of results for 50 runs. Under these settings, a T-k-PNN query produces two k-subsets as answers on average.

The experiments, written in Java, are executed on a PC with an Intel 2.66GHz CPU and 2GB of main memory. We have also implemented the k-bound filtering with the R-tree library in the Spatial Index Library [2].

6.2 Results

1. **k-bound Filtering.** In the first experiment, we examine the effectiveness of the k-bound filtering in pruning away unqualified objects. Fig. 9 illustrates the number of loaded data objects returned by k-bound filtering with an R-tree. As we can see, when k varies from one to nine, the size of the candidate object set increases smoothly. This is because the size of the k-bound increases with k. Consequently, the k-bound has overlap with more objects, and so more candidates need to be investigated. Another observation is that the number of candidate objects is small. In fact, the average fraction of the total database size to be examined is less than 0.04%. Thus, the pruning power of k-bound filtering is quite impressive.

On the other hand, although only a small fraction of objects are returned by k-bound filtering, the number of k-subsets generated by the candidate object set can still be very large. At $k = 9$, for instance, 22 objects are left. Out of these objects, a total of C_9^{22} (around 375K) k-subsets need to be examined. This renders a huge computational effort. To alleviate this problem, we need k-subset **G**eneration (with PCS), **V**erification, and **R**efinement techniques. Let us call these techniques collectively as the **GVR** method, and examine its effectiveness.

2. **Performance of GVR.** Here we compare the performance of GVR with that of *Basic* evaluation (described in Section 3.2). We assume that k-bound filtering has been applied first for both methods. As shown in Figure 10, the time required by *Basic* rises sharply with k, since the increase in k makes Equation 1 more expensive to compute. On the other hand, the query response time of GVR is an order of magnitude less than *Basic*. For example, when $k = 5$, GVR spends only 1.6% of the time required by *Basic*. We can thus see that GVR is important for improving the query performance. Next, let us investigate individual methods of GVR.

3. **k-subset Generation.** In this experiment, we study the performance of the PCS algorithm in generating k-subsets. Figure 11 shows the number of k-subsets produced by different techniques, in log scale. Compared with the "brute-force" method (i.e., enumerating all possible k-subsets from the candidate objects), PCS consistently generates less k-subsets under a wide range of T values. The savings are significant; at $k = 9$, for example, the improvement of PCS over the brute-force method is 90% (for $T = 0.05$) and 99% (for $T = 0.5$). Figure 12 shows that when T increases, the number of candidate k-subsets decreases sharply. Thus, the effectiveness of PCS improves with a higher value of T. It also shows that PCS can exploit the probability threshold to provide better performance.

To further enhance PCS, we have proposed **seed pruning** (in Section 4.3). As shown in Figure 13, this technique reduces the number of k-subsets produced over a wide range of k. For example, at $k = 9$, the improvement is about 80%. Figure 14 shows the corresponding effect on query response time, which addresses a saving of 69% at $k = 9$. Thus, seed pruning improves the performance of PCS significantly.

In view of the potentially large number of k-subsets generated during and after the execution of the PCS algorithm, we have designed an effective compression as discussed in Section 4.3. Figure 15 compares the storage cost with and without using this compression

[2]http://u-foria.org/marioh/spatialindex/index.html

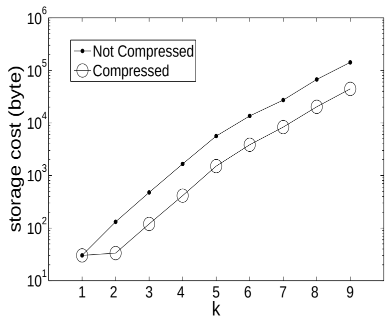


Figure 15: Efficient Storage of k-subsets.

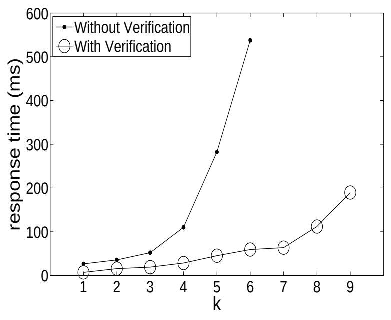


Figure 16: Effect of Verification.

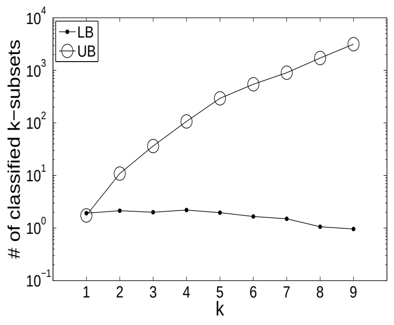


Figure 17: LB vs. UB.

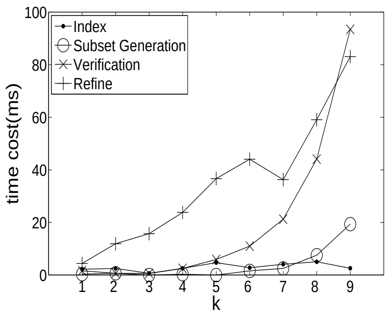


Figure 18: Time Analysis (with T=0.1)

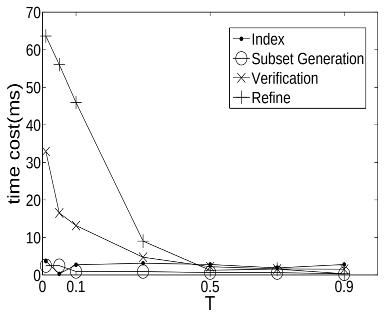


Figure 19: Time Analysis (with k=6).

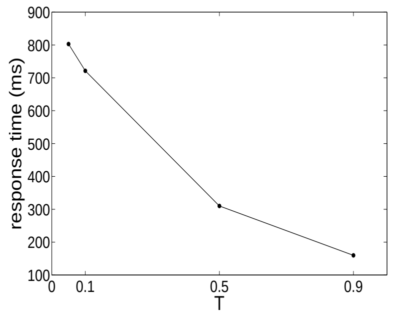


Figure 20: Various T on Gaussian Distribution

method (in log scale). We observe that under a wide range of values of k, after compression, we only need one-third of the storage that is used to store the raw k-subsets. Therefore, our compression method can greatly reduce the amount of storage required to record k-subsets for further processing.

4. Verification and Refinement. Next, we investigate the advantage of verification and refinement over direct computation of qualification probabilities. Figure 16 shows that the use of this technique yields significant improvement over different values of k. For example, at $k = 6$, verification and refinement reduces the query response time by about 90%.

We further examine the effectiveness of lower- and upper-bound verification. The lower (upper) bound verification method attempts to determine whether a k-subset should be accepted (rejected). Figure 17 shows that the number of k-subsets classified by UB is much larger than that classified by LB. The reason is that in the dataset we have tested, many k-subsets have small qualification probabilities. Thus, they are more likely to be rejected through upper-bound verification. Due to this reason, we have also arranged the UB subroutine to be executed before LB in the verification algorithm, as shown in Algorithm 4.

5. Time Analysis. To get a clearer picture about the performance of each part of our solution, we measure the time costs of k-bound filtering (shown as "Index" in Figures 18 and 19), k-subset generation (with PCS), verification, as well as refinement. Figures 18 show the result under different values of k. In general, most of the time is spent on refinement. This is hardly surprising, because refinement, which performs numerical integration on Equation 1, is an expensive process. However, this is already better than doing numerical integration alone (c.f. Figure 16). The price to pay for this time drop is to verify the k-subsets before their probabilities are actually evaluated. Although the time spent on verification also increases with k, the time spent is still less than pure numerical integration (c.f. Figure 16). We also notice that that k-bound filtering and PCS require the least amount of the time. These two steps add little overhead to the overall query performance. However, their gain, as reflected by Figures 9 and 11, is significant.

Figure. 19 shows the time breakdown of the components for different values of T. Again, the time costs required by k-bound filtering and PCS are the least. For all the methods, their performance improves with an increase of T. This shows that our methods can effectively exploit the query probability threshold.

6. Gaussian Distribution. In the final experiment, we use a Gaussian distribution as the uncertainty pdf for the dataset. For each object, the uncertainty pdf has a mean equal to the center of the uncertainty region, and a variance set to be the square of one-sixth of the edge length, in both x and y dimensions. Each uncertainty pdf is represented by $10 \times 10 = 100$ histogram bars, and the probability of each bar is the integration of the pdf over the area covered. Figure 20 illustrates the result of the GVR method for various values of T. We observe that GVR shows a similar trend as that of the uniform pdf (c.f. Figure 10). More time is spent on Gaussian pdf, because more histograms are used to model the pdf, which subsequently increases the time for verification and refinement. We have also performed other experiments for Gaussian pdf, and they also reflect similar trends. We thus omit them in the paper. From these experiments, we can see that our solution is robust with respect to different types of uncertainty pdfs.

7. CONCLUSIONS

Due to the popular usage of uncertain data in many real applications, uncertainty management has become an important topic in the database community. We studied a useful query, namely, the probability threshold k-NN Query (T-k-PNN) for uncertain databases. Different from the exact database, evaluating T-k-PNN requires probability information, and performs expensive numerical integration. Thus, we proposed various pruning techniques with consideration of both distance and probability constraints. As shown by our experimental results, with the k-bound filtering technique, a lot of unqualified objects can be pruned. The number of k-subsets can be significantly reduced by the PCS algorithm. We further demonstrated the efficient computation of lower and upper bounds of probabilities with the aid of partition information. We will study how these techniques can be extended to support other queries, e.g., reverse-neighbor and skyline queries.

8. ACKNOWLEDGMENT

This work was partially supported by NSFC/RGC Joint Research Scheme N_HKUST602/08 and National Natural Science Foundation of China (NSFC) under Grant No. 60763001. Reynold Cheng was supported by the Research Grants Council of Hong Kong (Projects HKU 5138/06E, HKU 513508, HKBU 1/05C), and the Seed Funding Programme of the University of Hong Kong (grant no. 200808159002). We also thank the reviewers for their insightful comments.

9. REFERENCES

[1] P. A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao, “Querying the uncertain position of moving objects,” in *Temporal Databases: Research and Practice*, 1998.

[2] D.Pfoser and C. Jensen, “Capturing the uncertainty of moving-objects representations,” in *Proc. SSDBM*, 1999.

[3] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong, “Model-driven data acquisition in sensor networks,” in *Proc. VLDB*, 2004.

[4] C. Böhm, A. Pryakhin, and M. Schubert, “The gauss-tree: Efficient object identification in databases of probabilistic feature vectors,” in *Proc. ICDE*, 2006.

[5] V. Ljosa and A. K. Singh, “APLA: Indexing arbitrary probability distributions,” in *Proc. ICDE*, 2007.

[6] R. Cheng, D. Kalashnikov, and S. Prabhakar, “Evaluating probabilistic queries over imprecise data,” in *Proc. ACM SIGMOD*, 2003.

[7] J. Chen and R. Cheng, “Efficient evaluation of imprecise location-dependent queries,” in *Proc. ICDE*, 2007.

[8] M. Mokbel, C. Chow, and W. G. Aref, “The new casper: Query processing for location services without compromising privacy,” in *VLDB*, 2006.

[9] D. Barbara, H. Garcia-Molina, and D. Porter, “The management of probabilistic data,” *TKDE*, vol. 4, no. 5, 1992.

[10] N. Dalvi and D. Suciu, “Efficient query evaluation on probabilistic databases,” in *VLDB*, 2004.

[11] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom, “Trio: A system for data, uncertainty, and lineage,” in *VLDB*, 2006.

[12] G. Iwerks, H. Samet, and K. Smith, “Continuous k-nearest neighbor queries for continuously moving points with updates,” in *Proc. VLDB*, 2003.

[13] S. Ganguly, M. Garofalakis, R. Rastogi, and K. Sabnani, “Streaming algorithms for robust, real-time detection of ddos attacks,” in *ICDCS*, 2007.

[14] U. Fayyad, G. Piatesky-Shapiro, P. Smyth, and R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press, 1996.

[15] N. Koudas, B. Ooi, K. Tan, and R. Zhang, “Approximate NN queries on streams with guaranteed error/performance bounds,” in *Proc. VLDB*, 2004.

[16] G. Beskales, M. Soliman, and I. Ilyas, “Efficient search for the top-k probable nearest neighbors in uncertain databases,” in *VLDB*, 2008.

[17] N. Dalvi and D. Suciu, “Efficient query evaluation on probabilistic databases,” in *Proc. VLDB*, 2004.

[18] O. Mar, A. Sarma, A. Halevy, and J. Widom, “ULDBs: databases with uncertainty and lineage,” in *VLDB*, 2006.

[19] L. Antova, C. Koch, and D. Olteanu, “Query language support for incomplete information in the maybms system,” in *Prof. VLDB*, 2007.

[20] S. Singh et al, “Orion 2.0: Native support for uncertain data,” in *Prof. ACM SIGMOD*, 2008.

[21] Singh et al, “Database support for pdf attributes,” in *Proc. ICDE*, 2008.

[22] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, “Querying imprecise data in moving object environments,” *IEEE TKDE*, vol. 16, no. 9, Sept. 2004.

[23] H. Kriegel, P. Kunath, and M. Renz, “Probabilistic nearest-neighbor query on uncertain objects,” in *DASFAA*, 2007.

[24] Y. Qi, S. Singh, R. Shah, and S. Prabhakar, “Indexing probabilistic nearest-neighbor threshold queries,” in *Proc. Workshop on Management of Uncertain Data*, 2008.

[25] X. Lian and L. Chen, “Probabilistic group nearest neighbor queries in uncertain databases,” *IEEE Trans. On Knowledge and Data Engineering*, vol. 20, no. 6, 2008.

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

[27] 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 *Proc. VLDB*, 2005.

[28] J. Pei, B. Jiang, X. Lin, and Y. Yuan, “Probabilistic skylines on uncertain data,” in *Proc. VLDB*, 2007.

[29] X. Lian and L. Chen, “Monochromatic and bichromatic reverse skyline search over uncertain databases,” in *Proc. SIGMOD*, 2008.

[30] M. Soliman, I. Ilyas, and K. Chang, “Top-k query processing in uncertain databases,” in *Proc. ICDE*, 2007.

[31] M. Hua, J. Pei, W. Zhang, and X. Lin, “Ranking queries on uncertain data: A probabilistic threshold approach,” in *Proc. SIGMOD*, 2008.

[32] V. Rastogi, D. Suciu, and E. Welbourne, “Access control over uncertain data,” in *Proc. VLDB*, 2008.

[33] C. Koch and D. Olteanu, “Conditioning probabilistic databases,” in *Proc. VLDB*, 2008.

[34] R. Cheng, J. Chen, and X. Xie, “Cleaning uncertain data with quality guarantees,” in *Proc. VLDB*, 2008.

[35] A. Guttman, “R-trees: A dynamic index structure for spatial searching,” *Proc. of the ACM SIGMOD Int’l. Conf.*, 1984.