VLDB Journal manuscript No.
(will be inserted by the editor)

Scalable Processing of Snapshot and Continuous Nearest-Neighbor Queries over One-Dimensional Uncertain Data

Jinchuan Chen · Reynold Cheng · Mohamed Mokbel · Chi-Yin Chow

Abstract In several emerging and important applications, such as location-based services, sensor monitoring and biological databases, the values of the data items are inherently imprecise. A useful query class for these data is the Probabilistic Nearest-Neighbor Query (PNN), which yields the IDs of objects for being the closest neighbor of a query point, together with the objects' probability values. Previous studies showed that this query takes a long time to evaluate. To address this problem, we propose the Constrained Nearest-Neighbor Query (C-PNN), which returns the IDs of objects whose probabilities are higher than some threshold, with a given error bound in the answers. We show that the C-PNN can be answered efficiently with *verifiers*. These are methods that derive the lower and upper bounds of answer probabilities, so that an object can be quickly decided on whether it should be included in the answer. We design five verifiers, which can be used on uncertain data with arbitrary probability density functions. We further develop a *partial evaluation* technique, so that a user can obtain some answers quickly, without waiting for the whole query evaluation process to be completed (which may incur a high response time). In addition, we examine the maintenance of a long-standing, or *continuous* C-PNN query. This query requires any update to be applied to the result immediately, in order to reflect the changes to the database values (e.g., due to the change of the location of a moving object). We design an incremental update method based on previous query answers, in order to reduce the amount of I/O and CPU cost in maintaining the correctness of the answers to such a query. Performance evaluation on realistic datasets show that our methods are capable of yielding timely and accurate results.

Jinchuan Chen
Key Labs of Data Engineering and Knowledge Engineering, MOE
School of Information, Renmin University of China
Bei Jing, P.R.China
(Work done in University of Hong Kong)
E-mail: csjcchen@gmail.com

Reynold Cheng
Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong
E-mail: ckcheng@cs.hku.hk

Mohamed Mokbel · Chi-Yin Chow
Department of Computer Science and Engineering
University of Minnesota-Twin Cities
200 Union Street SE, Minneapolis, MN, 55455
E-mail: {mokbel,cchow}@cs.umn.edu

1 Introduction

In several important and emerging applications, the values of the data items are inherently imprecise. As an example, a habitat monitoring system employs sensors to acquire temperature, humidity, and UV-light index values from the environment. Due to the imperfection of these sensing devices, the data obtained are noisy [1]. A similar situation happens in the Global-Positioning System (GPS), where the location values collected have some measurement error [2,3]. In biometric databases, the attribute values of the feature vectors stored are also not certain [4]. In some scenarios, data impreciseness is introduced deliberately. In Location-Based Services (LBS), for instance, the "dead-reckoning" approach is often used, where each mobile device sends an update to the system when its value has changed significantly. This reduces the need for the mobile object to report its location change. Here, the location is modeled in the database as a range of possible values [2,5]. Recently, the idea of injecting location

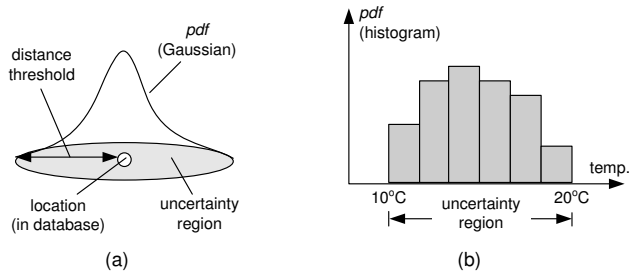

Fig. 1 Location and sensor uncertainty.

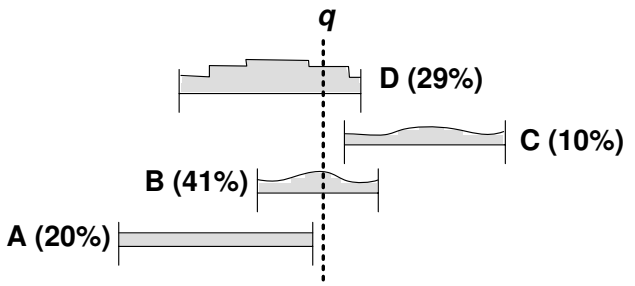

Fig. 2 Probabilistic NN Query (PNN).

uncertainty to a user's location in an LBS has been proposed [6,7], in order to protect a user's location privacy.

To provide high-quality query answers, it is important to capture and manage data uncertainty carefully. A well-studied model assumes that the actual data value is located within a closed area, known as the *uncertainty region*. In this region, a probability density function (*pdf*) of the value is defined, and the integration of the pdf inside the region is equal to one [2,3,5,4]. The exact form of the *pdf* is application-dependent. For example, in a sensor-monitoring system, we can collect a number of temperature-value samples over a week, model these observations as a Gaussian pdf, and compute the mean and variance values of these samples.

Gaussian distributions are also used to model the measurement error of vehicles' locations in moving object environments [3], and also the feature vectors in biometric databases [4].

Figure 1(b) shows the histogram of temperature values in a geographical area observed in a week. The pdf, represented as a histogram, is an arbitrary distribution between 10^oC and 20^oC. In this paper, we assume that a pdf is represented as a histogram. Also, we focus on uncertain objects in the one-dimensional space (i.e., a pdf defined inside a closed interval). However, our techniques are generally applicable to multi-dimensional data.

An important query for uncertain objects is the *Probabilistic Nearest Neighbor* Query (PNN in short) [5]. This query returns the non-zero probability (called *qualification probability*) of each object for being the nearest neighbor of a given point q. The qualification probability augmented with each object allows us to place confidence onto the answers. Figure 2 illustrates an example of PNN on four uncertain objects (A, B, C and D). The query point q and the qualification probability of each object are also shown.

A PNN could be used in a scientific application, where sensors are deployed to collect the temperature values in a natural habitat. For data analysis and clustering purposes, a PNN can be executed to find out the district(s) whose temperature values is (are) the closest to a given centroid. Another example is to find the IDs of sensor(s) that yield the minimum or maximum wind-speed from a given set of sensors [5,1]. A minimum (maximum) query is essentially a special case of PNN, since it can be characterized as a PNN by setting q to a value of $-\infty$ (∞).

Although PNN is useful, its evaluation is not trivial. Particularly, since the exact value of a data item is not known, one needs to consider the item's possible values in its uncertainty region. Moreover, an object's qualification probability depends not just on its own value, but also the relative values of other objects. If the uncertainty regions of the objects overlap, then their pdfs must be considered in order to derive their corresponding probabilities. In Figure 2, for instance, evaluating A's qualification probability (20%) requires us to consider the pdfs of the other three objects, since each of them has some chance of overtaking A as the nearest neighbor of q. Existing solutions either perform numerical integration on pdfs [5,8,1] or use Monte-Carlo sampling methods [9]. These solutions can be quite costly. Also, the accuracy of the solution depends on the precision of the integration or the number of samples used. It is worth mention that the indexing solution proposed in [8] successfully prunes away a large fraction of objects (with zero qualification probabilities). Nevertheless, their experiments show that the time for evaluating the probabilities for the remaining objects is still expensive. As a result, the user has to wait for a long time before obtaining the query answer from the system.

1.1 Probabilistic Verifiers

Although calculating qualification probabilities is expensive, a query user may not always be interested in the precise probability values. A user may only require answers with confidence that are higher than some fixed value. In Figure 2, for instance, if an answer with probability higher than 30% is required, then object B (41%) would be the only answer. If a user can tolerate with some approximation in the result (e.g., he allows an

object's actual probability to be 2% less than the 30% threshold), then object D (29%) can be another answer.

Here the *threshold* (30%) and *tolerance* (2%) are requirements or *constraints* imposed on a PNN. We denote this variant of PNN as the **Constrained Probabilistic Nearest-Neighbor Query** (or **C-PNN** in short). A C-PNN allows the user to control the desired confidence and quality in the query. The answers returned, which consists of less information than PNN, may also be easier to understand. In Figure 2, the C-PNN only includes objects (B,D) in its result, as opposed to the PNN which returns the probabilities of all the four objects.

The C-PNN has another advantage: its answers can be more efficiently evaluated. In particular, we have developed *probabilistic verifiers* (or *verifiers* in short), which can assist in making decisions on whether an object is included in the final answer, *without* computing the exact probability values. The verifiers derive a bound of qualification probabilities with algebraic operations, and test the bound with the threshold and tolerance constraints specified in the C-PNN. For example, one of our verifiers may use the objects' uncertainty regions to deduce that the probability of object A in Figure 2 is less than 25%. If the threshold is 30% and the tolerance is 2%, then A must not be the answer, even though we do not know about A's exact probability. In this paper, we have developed two classes of verifiers. The first group of verifiers (called *subregion-based verifiers*) utilize an object's pdf information to derive the lower and upper probability bounds. Another type of verifiers (called *result-based verifiers*) make use of the information produced by the subregion-based verifiers. All these verifiers can handle arbitrary pdfs. We also propose a framework for stringing these verifiers together, in order to provide an efficient solution. We will also explain the data structure required for this solution. We remarked that even if an object cannot be decided by the verifiers (and so that object's qualification probability has to be evaluated), the knowledge generated by the verifiers can still facilitate the probability computation process. We show experimentally that the price paid for using verifiers is justified by the lower cost of refinement. In some cases, the evaluation time of the C-PNN is an order of magnitude faster with the use of verifiers.

The C-PNN solution requires the knowledge of distance probability function information of the objects from q. Obtaining these values can be expensive. To save these costs, we devise a method that derives these function values only when they are needed. We also extend the C-PNN solution to handle the situation when the main memory cannot hold all the objects that are not pruned. In particular, we propose a simple extension to support efficient disk-based access.

1.2 Partial Evaluation and Incremental Update

The second problem we study is related to the "timeliness" of reporting PNN and C-PNN query answers. Since these queries may take a long time to complete, it is desirable for part of the query answers to be visible to the user before the completion of the whole query computation process. For instance, a certain C-PNN query yields 100 objects as its answer. The traditional option is to wait for the probabilistic query to complete (which may need a lot of time) and return 100 objects together to the user. The other alternative (which we call *partial evaluation*) returns the object to the user once it is confirmed to be the answer. The latter method could be more preferable since the user can see his results more quickly without waiting for the whole query to be completed. In other words, this technique can shorten the *response time* of a query.

Partial evaluation is also useful for the computation of *continuous* queries, which reside in the system for a substantial amount of time [1]. A continuous query is useful in applications like intrusion detection, road-traffic control, and advertisement of messages to customers in a shopping mall [10]. For these queries, any update to the database can invalidate the previous query answers, requiring the query to be re-evaluated. Waiting for a PNN/C-PNN query to finish may not be a very wise option, since an update to the object may arrive at the system during query computation. Thus, the query needs to be recomputed immediately after its evaluation on the old data values has just been completed. Partial evaluation allows the user to see fragments of answers during query computation; upon receiving any data update, the user can decide whether to finish or restart the query. In this paper, we show how partial evaluation can be implemented in the probabilistic verifier framework. We propose two methods, which differ in their ways of evaluating the objects. We also propose two metrics to measure the responsiveness of these queries.

As we have mentioned, a continuous query needs to have its results recomputed in order to reflect the database changes. Recomputing the whole query can be expensive (in terms of both I/O and computation), especially for a continuous C-PNN. We tackle this problem by two means. First, we propose an *incremental evaluation* technique, which *reuses* the query results

[1] We call a query which is only executed once the "snapshot query".

before update arrival, in order to reduce the I/O overhead in executing the query. Secondly, to reduce the computation cost, we observe that when an object gets updated, the changes in probabilities of other objects in the query answer are bounded. For example, in Figure 2, if the qualification probability of C increases from 10% to 20%, we show that the probabilities of other objects cannot decrease by more than 10%. For example, B's probability is at least 31% after the update of C. If the probability threshold is 0.3, then B must still satisfy the query after this update, and so we do not need to change the answer. Essentially, we use the change of C to derive the knowledge about B, without recomputing B's actual probability. We call this a *lazy* approach, since an object can be determined on whether it still satisfies the query without going through the whole verification/refinement process. We illustrate how this approach can efficiently handle insertion, deletion, and updates of objects. Our experiments indeed show that this method significantly reduces the chance that a C-PNN has to be recomputed.

To our best knowledge, there are no past studies on partial evaluation techniques on probabilistic queries. We are also not aware of any work on efficient update algorithms for continuous probabilistic queries. Our contributions are summarized as follows:

- Introduce the notion of C-PNN;
- Design five different verifiers for efficient evaluation of C-PNN;
- Study partial evaluation techniques for shortening query response time;
- Present an on-demand method for computing distance probability functions;
- Extend the query evaluation method to support disk-based access;
- Develop efficient approaches for reducing I/O and computation of continuous C-PNN; and
- Perform experimental evaluations on both real and synthetic datasets.

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 C-PNN, and its basic evaluation method. We then present the details of the probabilistic verifiers and partial evaluation in Section 4. The details of continuous query evaluation are presented in Sections 5. We describe the experimental results in Section 6, and conclude the paper in Section 7.

2 Related Work

Uncertain Databases Recently, database systems for managing uncertainty have been proposed [11–13, 1,14]. Two major types of uncertainty are assumed in these works: tuple- and attribute-uncertainty. Tuple-uncertainty refers to the probability that a given tuple is part of a relation [11,15–17]. Attribute-uncertainty generally represents the inexactness in the attribute value as a range of possible values and a pdf bounded in the range [2,3,5,18,4]. The imprecise data model studied here belongs to the attribute uncertainty.

An R-tree-based solution for PNN over attribute uncertainty has been presented in [8]. The main idea is to prune tuples with zero probabilities, using the fact that these tuples' uncertainty regions must not overlap with that of a tuple whose maximum distance from the query point is the minimum in the database. [5, 8] discuss the evaluation of qualification probabilities by transforming each uncertainty region 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 probabilities are then derived by evaluating an integral of an expression involving distance pdfs and cdfs from multiple objects. While our solution also uses distance pdfs and cdfs, its avoids a significant number of integration operations with the aid of verifiers. This paper is based on our previous work in [19], where the C-PNN query is proposed and evaluated by using a verification-refinement framework. This paper is not only a detailed version of [19], but also contains several new issues such as the continuous C-PNN query, partial evaluation, and several new verifiers.

Another method for evaluating a PNN is proposed in [9], where each object is represented as a set of points (sampled from the object's pdf). In this method, the samples of each object are grouped into clusters using the k-means algorithm. Then, the qualification probabilities of the objects are computed by comparing the minimum and maximum distances of the samples (clusters) from the query point. Compared with that work, our solution is tailored for a constrained version of PNN, where threshold and tolerance conditions are used to avoid computation of exact probabilities. Also, we do not need the additional work of sampling the pdf into points. Notice that this sampling process may introduce another source of error if there are not enough samples. We will experimentally evaluate and compare the method in [9] with ours in Section 6. In [20], a method for evaluating the probability that an object (represented as a histogram) is before another object in the time domain is presented. Their result could be adapted to answer a PNN that involves two objects, by viewing the time domain as the space domain. Our solution, on the other hand, addresses the PNN problem involving two or more objects.

Some related works about PNN can also be found in [21,22], where the existential probabilities are used to derive lower and upper bounds and pruning for nearest-neighbors, and [23], where the authors address how to efficiently retrieve data objects that minimize the aggregate distance to a query set.

Besides PNN, probabilistic top-k queries (i.e., queries that return k tuples with the highest probabilities) have been considered for tuple uncertainty [11,15–17]. These work considered issues like efficient query evaluation and query plans. The evaluation of k-NN queries for attribute uncertainty have been considered in [24]. Ljosa et al. [18] investigated the problem of evaluating k-NN queries over multi-dimensional uncertain data, where the query answers are ranked according to expected Manhattan distance from the query point. Other probabilistic queries studied includes range queries [25] and location-dependent queries [6]. The issues of uncertainty have also been considered in similarity matching in biometric databases [4].

Continuous Query Evaluation Recently, there are plenty of works on continuous queries, which can be used in numerous applications, including intrusion detection, road-traffic control, and advertisement of messages to customers in some confined area (e.g., a shopping mall). Since the answers to a continuous query may be affected by data updates, they have to be recomputed periodically [10,26]. The handling of frequent data updates and continuous queries place heavy burden on the system. Thus, a number of approaches have been proposed to address these problems. In particular, efficient indexing schemes have been proposed that can adapt to the high frequency of location updates, including [10,27,26]. [28] developed an efficient incremental algorithm for k-nearest-neighbour queries. Its main idea is to use the fact that the newest query result can be obtained by applying the location update on the original result by using an "incremental algorithm". This method has been proven to be more effective than re-evaluating the whole query. To our best knowledge, there is no prior attempts for saving recomputation costs due to data update for continuous probabilistic nearest-neighbor queries. In this paper, we propose a lazy update approach to reduce the chance of query recomputation. We also study how probabilistic query answers can be partially returned to the user, and this has also not been investigated before.

3 Snapshot and Continuous PNN Queries

Let us now present the semantics of the snapshot C-PNN and its continuous counterpart (Section 3.1). We then present the solution framework in Section 3.2.

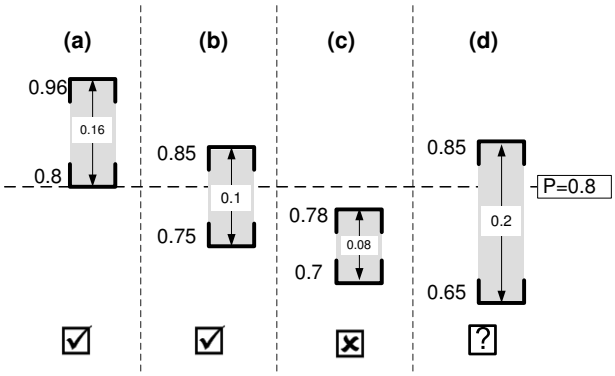

Fig. 3 C-PNN with $P = 0.8$ and $\Delta = 0.15$.

3.1 C-PNN and CC-PNN

Let X be a set of uncertain objects in 1D space (i.e., an arbitrary pdf defined inside a closed interval), and X_i be the i^{th} object of X (where $i = 1, 2, \ldots, |X|$). Essentially, each object X_i has an arbitrary pdf defined inside a closed region, as illustrated in Figure 1. At each discrete time instant, a set S of objects (where $S \subseteq X$) reports its new information (e.g., a new region and a new pdf) to the system. In a LBS, for example, an update may be generated and reported by the moving object and reported to the system.

Next, suppose that $q \in \Re$ is the query point, and $p_i \in [0, 1]$ is the probability (i.e., qualification probability) that X_i is the nearest neighbor of q at time t. We call $p_i.l \in [0, 1]$ and $p_i.u \in [0, 1]$ the *lower* and *upper probability bound* of p_i respectively, such that $p_i.l \leq p_i.u$, and $p_i \in [p_i.l, p_i.u]$. In essence, $[p_i.l, p_i.u]$ is the range of possible values of p_i, and $p_i.u - p_i.l$ is the error in estimating the actual value of p_i. We denote the range $[p_i.l, p_i.u]$ as a *probability bound* of p_i.

Definition 1 A **Constrained Probabilistic Nearest Neighbor Query (C-PNN)** returns a set $\{X_i | i = 1, 2, \ldots, |X|\}$ such that p_i satisfies both of the following conditions:

- $p_i.u \geq P$
- $p_i.l \geq P$, or $p_i.u - p_i.l \leq \Delta$

where $P \in (0, 1]$ and $\Delta \in [0, 1]$.

Here P is called the *threshold* parameter. An object is allowed to be returned as an answer if its qualification probability is not less than P. Another parameter, called *tolerance* (Δ), limits the amount of error allowed in the estimation of qualification probability p_i.

Fig. 3 illustrates the semantics of C-PNN. The probability bound $[p_j.l, p_j.u]$ of some object X_j (shaded) is shown in four scenarios. Let us assume that the C-PNN has a threshold $P = 0.8$ and tolerance $\Delta = 0.15$. Case

Symbol	Meaning
X_i	Uncertain object i of a set X ($i = 1, 2, \ldots, \|X\|$)
q	Query point
p_i	Prob. that X_i is the NN of q (qualification prob.)
$[p_i.l, p_i.u]$	Lower & upper probability bounds
P	Threshold
Δ	Tolerance

Table 1 Symbols for C-PNN and CC-PNN.

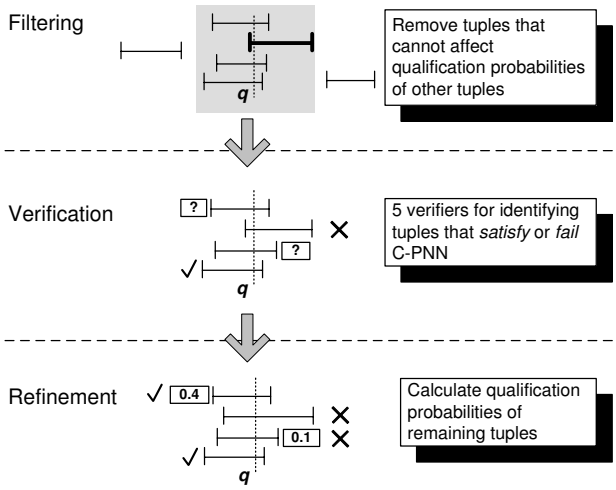

Fig. 4 Solution Framework of C-PNN.

(a) shows that the actual qualification probability p_j of some object X_j (i.e., p_j) is within a closed bound of $[p_j.l, p_j.u]$=[0.8, 0.96]. Since p_j must not be smaller than P, according to Definition 1, X_j is the answer to this C-PNN. In (b), X_j is also a valid answer since the upper bound of p_j (i.e., $p_j.u$) is equal to 0.85 and is larger than P. Moreover, the error of estimating p_j (i.e., 0.85-0.75), being 0.1, is less than $\Delta = 0.15$. Thus the two conditions of Definition 1 are satisfied. For case (c), X_j cannot be the answer, since the upper bound of p_j (i.e., 0.78) is less than P, and so the first condition of Definition 1 is violated. In (d), although object X_j satisfies the first requirement ($p_j.u = 0.85 \geq P$), the second condition is not met. According to Definition 1, it is not an answer to the C-PNN. However, if the probability bounds could later be "shrunken" (e.g., by verifiers), then the conditions can be checked again. For instance, if $p_j.l$ is later updated to 0.81, then X_j will be the answer.

We also study the *continuous* C-PNN query (or **CC-PNN** in short), which resides in the system for an extensive amount of time. Whenever an update is received by the system, this query may have to be recomputed, in order to reflect the change in the database. Formally, the CC-PNN is defined as follows:

Definition 2 Given a query point q, a **Continuous Constrained Probabilistic Nearest Neighbor Query (CC-PNN)** returns $\mathbb{X}(t) = \{X_i | i = 1, 2, \ldots, |X|\}$, where $\mathbb{X}(t)$ is the result of the C-PNN query for q evaluated over X at time instant t.

Table 1 summarizes the symbols used in the definition of (snapshot) C-PNN and CC-PNN.

3.2 Solution Overview

Now let us outline our approach for answering C-PNN. As shown in Figure 4, our solution consists of three phases. The first step is to prune or *filter* objects that must not be the nearest neighbor of q, using an R-tree based solution [8]. The objects with non-zero qualification probabilities (shaded) are then passed to the *verification* phase, where verifiers are used to decide if an object satisfies the C-PNN. In this figure, two objects have been determined in this stage. Objects that still cannot be determined are passed to the *refinement* phase, where the exact probability values are computed. We can see that the object with 0.4 probability is retained in the answer, while the other object (with 0.1 chance) is excluded.

In this paper, we assume that objects that are not pruned by the filtering phase can be stored in the main memory.

As our experiments and previous studies (e.g., [5, 8]) show, few (less than 100) objects remain after filtering. It is also worth mention that some applications (e.g., LBS) assume that the data about the moving objects can be stored in the main memory [29]. We focus on the verification and refinement issues of C-PNN. In particular, we present five verifiers, which utilize an object's uncertainty information in different ways, in order to derive lower/upper bounds of qualification probabilities in a cost-effective manner. In the next section, we will examine the principles of these verifiers, and how they are integrated in our solution. We will also discuss how the knowledge derived by the verifiers can facilitate the refinement phase. We then study the adapation of the verifier framework for yielding partial C-PNN query answers, in order to improve query response time. We also propose useful metrics to quantify query responsiveness. In Section 5, we study the problem of updating CC-PNN queries due to update arrivals. By using previous query results, we propose an incremental update method for reducing the I/O required in the filtering phase. We also present a lazy update method, so that the verification and refinement effort of re-evaluating a CC-PNN can be reduced without violating tolerance requirements.

4 Verification and Refinement

Let us now have an overview of the verification phase for evaluating the C-PNN. As we have discussed, uncertain objects that cannot be filtered (shaded in Figure 4) require further processing. This set of unpruned objects, called the *candidate set* (or C in short), are passed to *probabilistic verifiers*, which report a list of probability bounds of these objects. This list is sent to the *classifier*, which labels an object by checking its probability bounds against the definition of the C-PNN. In particular, an object is marked *satisfy* if it qualifies as an answer (e.g., Figure 3(a), (b)). It is labeled *fail* if it cannot satisfy the C-PNN (Figure 3(c)). Otherwise, the object is marked *unknown* (Figure 3(d)). This labeling process can be done easily by checking an object's probability bounds against the two conditions stated in Definition 1.

Figure 5 shows the five verifiers (in shaded boxes), as well as the classifier. During initialization, all objects in the candidate set are labeled *unknown*, and their probability bounds are set to $[0, 1]$. Other information like the distance pdf and cdf is also precomputed for the candidate set objects. The candidate set is then passed to the first verifier (RS) for processing. The RS produces the newly computed probability bounds for the objects in the candidate sets, and sends this list to the classifier to label the objects. Any objects with the label *unknown* are transferred to the next verifier (L-SR) for another round of verification. The process goes on (with the order of execution of verifiers numbered in Figure 5) until all the objects are either labeled *satisfy* or *fail*. When this happens, all the objects marked *satisfy* are returned to the user, and the query is finished. Thus, it is not always necessary for all verifiers to be executed.

Notice that a verifier only adjusts the probability bound of an *unknown* object if this new bound is smaller than the one previously computed. The subregion-based verifiers are arranged in the order of their running time, so that if a low-cost verifier (e.g., the RS verifier) can successfully determine all the objects, there is no need to execute a more costly verifier (e.g., the L-SR verifier). In the end of verification, objects that are still labeled *unknown* are passed to the refinement stage for computing their exact probabilities.

From Figure 5, we can also see that the verifiers are classified into *subregion-based* and *result-based*. The main difference between these two classes of verifiers lies in the way they derive probability bounds. Subregion-based verifiers uses the information of *subregions* to compute the probability bounds. A subregion is essentially a partition of the space derived from the uncertainty regions of the candidate set objects. This information is used by the RS, L-SR and U-SR verifiers. On the other hand, the result-based verifiers (i.e., U-PB and L-PB) uses results derived by the subregion-based verifiers to infer the bounds. For example, the U-PB verifier uses the lower-bound information derived by the L-SR verifier. As we will explain, a subregion-based verifier is more powerful than a result-based verifier, but are also more costly to use.

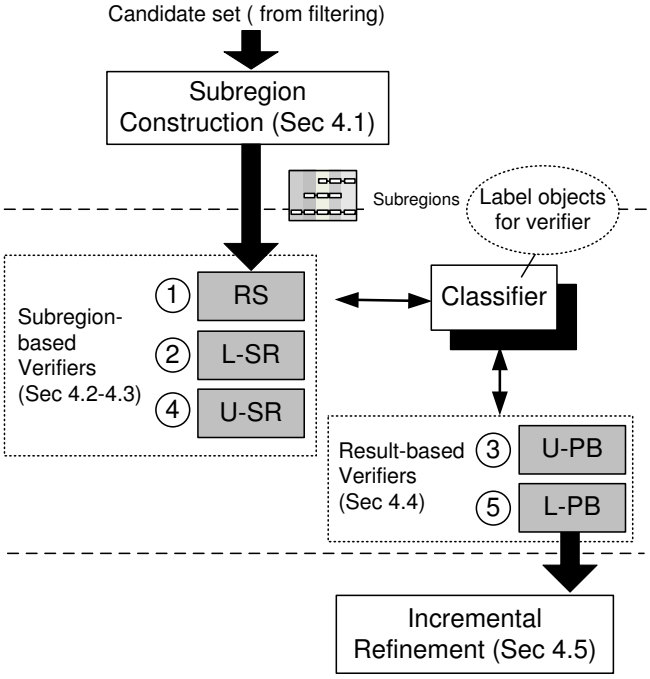

Fig. 5 The Verification Framework.

The rest of this section is organized as follows. Section 4.1 discusses how subregions are produced. We then present the RS-verifier in Section 4.2, followed by the L-SR and U-SR verifiers in Section 4.3. In Section 4.4 we examine the result-based verifiers. We then describe the "incremental refinement" method, which uses the verifiers' information to improve the refinement process, in Section 4.5. We also discuss how to extend our solution to handle partial query evaluation and other issues in Section 4.6.

4.1 Computing Subregion Probabilities

The initialization phase in Figure 5 performs two tasks: (1) computes distance pdf and cdf for each object in the candidate set, and (2) derives *subregion probabilities*.

We start with the derivation of distance pdf and cdf. Let $R_i \in \Re$ be the absolute distance of an uncertain object X_i from q. That is, $R_i = |X_i - q|$. We assume that R_i takes on a value $r \in \Re$. Then, the distance pdf and cdf of X_i are defined as follows [5,8]:

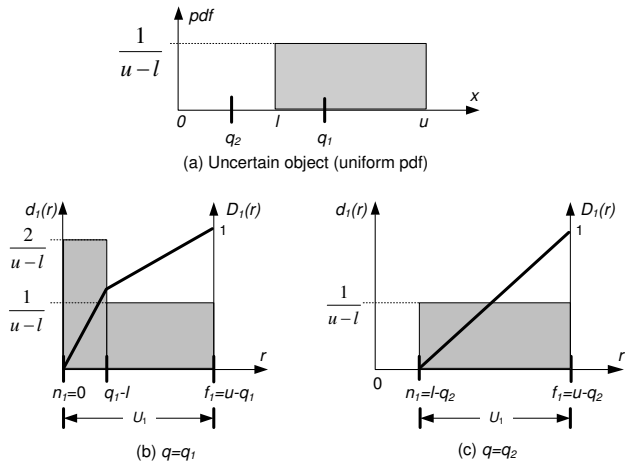

Fig. 6 Distance pdf and cdf

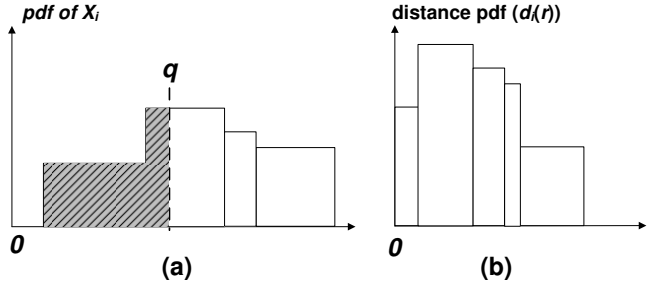

Fig. 7 Histogram pdf.

Definition 3 Given an uncertain object X_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.

Figure 6(a) illustrates an uncertain object X_1, which has a uniform pdf with a value of $\frac{1}{u-l}$ in an uncertainty region $[l, u]$. Two query points (q_1 and q_2) are also shown. Figure 6(b) shows the corresponding distance pdf (shaded) of $R_1 = |X_1 - q_1|$, with q_1 as the query point. Essentially, we derive the pdf of X_1's distance from q_1, which ranges from 0 to $u-q_1$. In $[0, q_1-l]$, the distance pdf is obtained by summing up the pdf on both sides of q_1, which equals to $\frac{2}{u-l}$. The distance pdf in the range $[q_1 - l, u - q_1]$ is simply $\frac{1}{u-l}$. Figure 6(c) shows the distance pdf for query point q_2. For both queries, we draw the distance cdf in solid lines. Notice that the distance cdf can be found by integrating the corresponding distance pdf. From Figure 6, we observe that the distance pdf and cdf for the same uncertain object vary, and depend on the position of the query point.

We represent the distance pdf of each uncertain object as a histogram. Note that this distance pdf/cdf can conceptually be found by first decomposing the histogram into a number of "histogram bars", where the pdf in the range of each bar is the same. We can then compute the distance pdf/cdf of each bar using the methods (for uniform pdf) described in the previous paragraph, and combine the results to yield the distance pdf/cdf for the histogram. Figure 7(a) illustrates the histogram pdf of an uncertain object, and its corresponding distance pdf (in (b)). The corresponding distance cdf is a piecewise linear function. In practice, we store a histogram as an array of the histogram bars' end-points and their corresponding pdfs. Then, given a query q, we split the histogram into two halves (as shown in the two shaded parts in Figure 7). A merge sort is then performed on these end-points, during which their distance pdfs and cdfs are computed. If there are a total of h end-points in a histogram pdf, the process of generating distance pdfs and cdfs can be done in $O(h)$ time.

Although we focus on 1D uncertainty, we only need distance pdfs and cdfs. In fact, our solution can conceptually be used in multi-dimensional space, as long as the function for converting an uncertainty pdf into distance pdf/cdf is provided. For example, the problem of generating distance pdf in 2D space was studied in [8], where the authors proposed to derive the exact form of the distance cdf of an object by calculating the probability that the distance between the object and the query point is less than a given value. The issue of efficiently converting multi-dimensional pdf into distance pdf/cdf is left for future study. Next, we describe the definitions of near and far points of R_i, as defined in [5,8]:

Definition 4 A **near point** of R_i, denoted by n_i, is the minimum value of R_i. A **far point** of R_i, denoted by f_i, is the maximum value of R_i.

We use U_i to denote the interval $[n_i, f_i]$. Figure 6(b) shows that when q_1 is the query point, $n_1 = 0$, $f_1 = u - q_1$, and $U_1 = [0, u - q_1]$. When q_2 is the query point, $U_1 = [l - q_2, u - q_2]$ (Figure 6(c)). We also let f_{min} and f_{max} be the minimum and maximum values of all the far points defined for the candidate set objects. We assume that the distance pdf of X_i has a non-zero value at any point in U_i.

Subregion Probabilities. Upon generating the distance pdfs and cdfs of the candidate set objects, the next step is to generate *subregions*. Let us first sort these objects in the ascending order of their near points. We also rename the objects as $X_1, X_2, \ldots, X_{|C|}$, where $n_i \leq n_j$ iff $i \leq j$. Figure 8(a) illustrates three distance pdfs with respect to a query point q, presented in the ascending order of their near points. The number above each range indicates the probability that an uncertain object is at that range of distance from q.

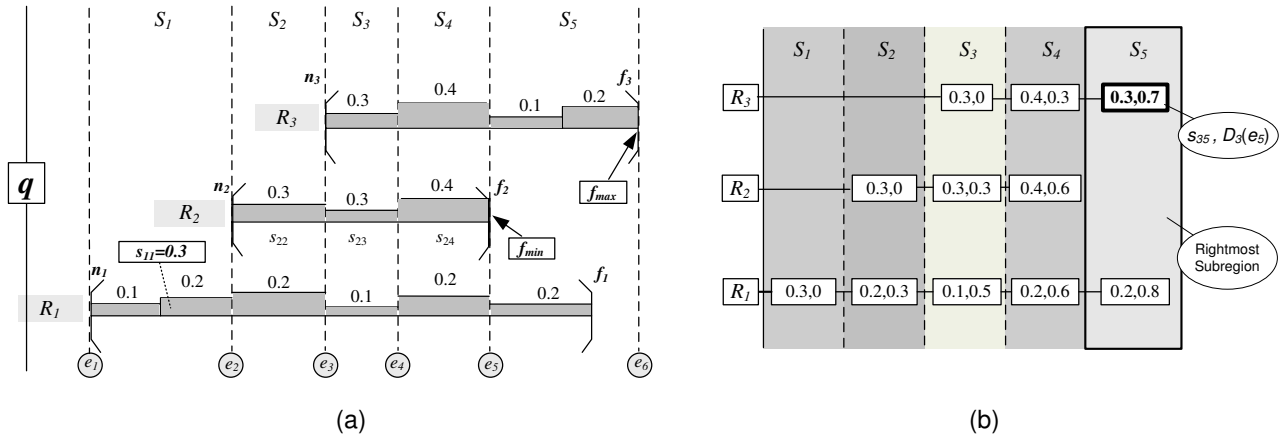

Fig. 8 Illustrating the distance pdfs and subregion probabilities.

In Figure 8(a), the circled values are called *end-points*. These end-points include all the near points (e.g., e_1, e_2 and e_3), the minimum and the maximum of far points (e.g., e_5 and e_6), and the point at which the distance pdf changes (e.g., e_4). Notice that if the probability of an object for being the nearest neighbor of q does not change when its distance from q is within an interval, no end-points are needed to further split this interval. Hence no end points are defined within (e_1, e_2), because the qualification probability of X_1 must always be equal to 1 when R_1 is inside this range. In the range (e_5, e_6), as we will explain in Section 4.2, all objects have zero chance to be the nearest neighbor of q if the distance pdf is located in this range. Hence, no end-points are needed inside (e_5, e_6). We use e_j to denote the j-th end-point, where $j \geq 1$ and $e_j < e_{j+1}$. Moreover, $e_1 = n_1$.

The adjacent pairs of end-points form the boundaries of a *subregion*. We label each subregion as S_j, where S_j is the interval $[e_j, e_{j+1}]$. Figure 8(a) shows five subregions, where $S_1 = [e_1, e_2]$, $S_2 = [e_2, e_3]$, and so on. The probability that R_i is located in S_j is called the *subregion probability*, denoted by s_{ij}. Figure 8(a) shows that $s_{22} = 0.3$, $s_{11} = 0.1 + 0.2 = 0.3$, and $s_{31} = 0$.

For each subregion S_j of an object X_i, we evaluate the subregion probability s_{ij}, as well as the distance cdf of S_j's lower end-point (i.e., $D_i(e_j)$). Figure 8(b) illustrates these pairs of values extracted from the example in (a). For example, for R_3 in S_5, the pairs $s_{35} = 0.3$ and $D_3(e_5) = 0.7$ are shown. These number pairs help the verifiers to develop the probability bounds. Note that given a subregion S_j, it is possible that $s_{ij} = 0$, for every object X_i (due to the fact that uncertainty pdf of X_i may be zero at some points). This subregion S_j can be safely removed because the qualification probability of each object in S_j must be equal to zero. In the sequel, we assume that every subregion must have at least one object with non-zero subregion probabilities. Table 2 presents the symbols used in our solution.

Next, we examine how verifiers use subregion information, in Sections 4.2 and 4.3.

Symbol	Meaning
C	$\{X_i \in X \mid p_i > 0\}$ (*candidate set*)
R_i	$\lvert X_i - q \rvert$
$d_i(r)$	pdf of R_i (*distance pdf*)
$D_i(r)$	cdf of R_i (*distance cdf*)
n_i, f_i	Near and far points of distance pdf
U_i	The interval $[n_i, f_i]$
f_{min}, f_{max}	min. and max. of far points
e_k	The k-th end point
S_j	The j-th subregion, where $S_j = [e_j, e_{j+1}]$
M	Total no. of subregions
c_j	No. of objects with non-zero subregion prob. in S_j
s_{ij}	$Pr(R_i \in S_j)$
q_{ij}	Qualification prob. of X_i, given $R_i \in S_j$
$[q_{ij}.l, q_{ij}.u]$	Lower & upper bounds of q_{ij}

Table 2 Symbols used by verifiers.

4.2 The Rightmost-Subregion Verifier

The **Rightmost-Subregion** (or RS) verifier uses the information in the "rightmost" subregion. In Figure 8(b), S_5 is the rightmost subregion. If we let $M \geq 2$ be the number of subregions for a given candidate set, then the following specifies an object's upper probability bound:

Lemma 1 *The upper probability bound, $p_i.u$, is at most $1 - s_{iM}$, where s_{iM} is the probability that R_i is in S_M.*

The subregion S_M is the rightmost subregion. In Figure 8(b), $M = 5$. The upper bound of the qualifica-

tion probability of object X_1, according to Lemma 1, is at most $1 - s_{15}$, or $1 - 0.2 = 0.8$.

To understand this lemma, notice that any object with distance larger than f_{min} *cannot* be the nearest neighbor of q. This is because f_{min} is the minimum of the far points of the candidate set objects. Thus, there exists an object X_k such that X_k's far point is equal to f_{min}, and that X_k is closer to q than any objects whose distances are larger than f_{min}. If we also know the probability that an object is at a distance of more than f_{min} from q, then its upper probability bound can be deduced. For example, Figure 8(a) shows that the distance of X_1 from q (i.e., R_1) has a 0.2 chance of being more than f_{min}. Thus, X_1 is not the nearest neighbor of q with a probability of at least 0.2. Equivalently, the upper probability bound of X_1, i.e., $p_1.u$, is $1 - 0.2 = 0.8$. Note that 0.2 is exactly the probability that R_1 lies in the rightmost subregion S_5, i.e., s_{15}, and thus $p_1.u$ is equal to $1 - s_{15}$. This result can be generalized for any object in the candidate set, as shown in Lemma 1.

Notice that the RS verifier only handles the objects' upper probability bounds. To improve the lower probability bound, we need the L-SR verifier, as described next.

4.3 The Lower- and Upper-Subregion Verifiers

The Lower-Subregion (**L-SR**) and Upper-Subregion (**U-SR**) Verifiers uses subregion probabilities to derive the objects' probability bounds. For each subregion the L-SR (U-SR) verifier computes the lower (upper) probability bound of each object. [2]

We define the term *subregion qualification probability* (q_{ij} in short), which is the chance that X_i is the nearest neighbor of q, given that its distance from q, i.e., R_i, is inside subregion S_j. We also denote the lower bound of the subregion qualification probability as $q_{ij}.l$. Our goal is to derive $q_{ij}.l$ for object X_i in subregion S_j. Then, the lower probability bound of X_i, i.e., $p_i.l$, is evaluated. Suppose there are $c_j (c_j \geq 1)$ objects with non-zero subregion probabilities in S_j. For example, $c_3 = 3$ in Figure 8(a), where all three objects have non-zero subregion probabilities in S_3. The following lemma is used by the L-SR verifier to compute $q_{ij}.l$.

Lemma 2 *Given an object $X_i \in C$, if $e_j \leq R_i \leq e_{j+1}$ $(j = 1, 2, \ldots, M-1)$, then*

$$q_{ij}.l = \frac{1}{c_j} \prod_{U_k \cap S_j \neq \emptyset \wedge k \neq i} (1 - D_k(e_j)) + (1 - \frac{1}{c_j}) \prod_{U_k \cap S_j \neq \emptyset \wedge k \neq i} (1 - D_k(e_{j+1})) \quad (1)$$

Lemma 2 calculates $q_{ij}.l$ for object X_i by using the distance cdfs of all objects with non-zero subregion probabilities in S_j. We will prove this lemma in Section 4.3.1. To illustrate the lemma, Figure 8(a) shows that $q_{11}.l$ (for X_1 in subregion S_1) is equal to 1, since $c_1 = 1$. On the other hand, $q_{23}.l$ (for X_2 in S_3) is $\frac{(1-0.5)(1-0)}{3} + (1 - \frac{1}{3})(1 - 0.3)(1 - 0.6)$, or 0.35.

Next, we define a real constant Y_j, where

$$Y_j = \prod_{U_k \cap S_j \neq \emptyset} (1 - D_k(e_j)) \quad (2)$$

Then, Equation 1 can be rewritten as:

$$q_{ij}.l = \frac{Y_j}{c_j(1 - D_i(e_j))} + (1 - \frac{1}{c_j}) \frac{Y_{j+1}}{1 - D_i(e_{j+1})} \quad (3)$$

By computing Y_j first, the L-SR can use Equation 3 to compute $q_{ij}.l$ easily for each object in the same subregion S_j.

After the values of $q_{ij}.l$ have been obtained, the lower probability bound ($p_i.l$) of object X_i can be evaluated by:

$$p_i.l = \sum_{j=1}^{M-1} s_{ij} \cdot q_{ij}.l \quad (4)$$

The product $s_{ij} \cdot q_{ij}.l$ is the minimum qualification probability of X_i in subregion s_{ij}, and Equation 4 is the sum of this product over the subregions. Note that the rightmost subregion (S_M) is not included, since the probability of any object in S_M must be zero.

The U-SR verifier uses Lemma 3 to evaluate the upper subregion probability bound ($q_{ij}.u$) of object X_i in subregion S_j:

Lemma 3 *Given an object $X_i \in C$, if $e_j \leq R_i \leq e_{j+1}$ $(j = 1, 2, \ldots, M-1)$, then*

$$q_{ij}.u = \begin{cases} 1 & \text{if } j = 1 \\ \frac{1}{2} \cdot (\prod_{U_k \cap S_{j+1} \neq \emptyset \wedge k \neq i} (1 - D_k(e_{j+1})) & \\ + \prod_{U_k \cap S_j \neq \emptyset \wedge k \neq i} (1 - D_k(e_j))) & \text{if } j > 1 \end{cases} \quad (5)$$

Similar to L-SR, Equation 5 can be simplified to:

$$q_{ij}.u = \begin{cases} 1 & \text{if } j = 1 \\ \frac{1}{2}(\frac{Y_j}{1 - D_i(e_j)} + \frac{Y_{j+1}}{1 - D_i(e_{j+1})}) & \text{if } j > 1 \end{cases} \quad (6)$$

where Y_j and Y_{j+1} are given by Equation 2. Thus, if the Y_j's are known, we can conveniently compute $q_{ij}.u$ with Equation 6. The upper probability bound ($p_i.u$) can be computed by replacing $q_{ij}.l$ with $q_{ij}.u$ in Equation 4. Next, we present the correctness proofs of the L-SR and U-SR verifiers.

[2] The L-SR verifier presented here achieves a tighter lower bound than that presented in [19]. We will also compare them experimentally.

4.3.1 Correctness of L-SR and U-SR

We first state a claim about subregions: if there exists a set K of objects whose distances from q (i.e., R_i) are certainly inside a subregion S_j, and all other objects $(C \setminus K)$'s distances are in subregions $j+1$ or above, then the qualification probability of each object in K is equal to $\frac{1}{|K|}$. This is because all objects in $C \setminus K$ cannot be the nearest neighbor of q, and all objects in K must have the same qualification probability. In Figure 8(a), for example, if the distances R_1 and R_2 are within the range $S_2 = [e_2, e_3]$, then $p_1 = p_2 = \frac{1}{2}$, and $p_3 = 0$. The following lemma states this formally.

Lemma 4 *Suppose there exists a nonempty set $K (K \subseteq C)$ of objects such that $\forall X_i \in K, e_j \leq R_i \leq e_{j+1}$. If $\forall X_m \in C \setminus K, R_m > e_{j+1}$, then for any $X_i \in K$, $p_i = \frac{1}{|K|}$, where $|K|$ is the number of objects in K.*

Proof Let us consider an object $X_i \in K$. When $|K| = 1$, $p_i = 1$. This is because the other $|C|-1$ objects must be further to q than X_i. Therefore, X_i must be the nearest neighbor to q, and the lemma is proved.

Now consider $|K| > 1$. Since $d_i(r)$ is a uniform distribution within S_j, and $R_i \in [e_j, e_{j+1}]$, we have $d_i(r) = \frac{1}{e_{j+1}-e_j}$ for $r \in [e_j, e_{j+1}]$. Moreover, for any $X_k \in K$, $D_k(r) = \frac{r-e_j}{e_{j+1}-e_j}$. The remaining $|C|-|K|$ objects cannot be the nearest neighbor, since they must be farther than X_i from q. Hence p_i can be calculated by considering only the objects in K:

$$p_i = \int_{n_i}^{f_i} \frac{1}{e_{j+1}-e_j} \cdot \prod_{X_k \in K \wedge k \neq i} (1 - \frac{r-e_j}{e_{j+1}-e_j})\, dr \quad (7)$$

$$= \int_{e_j}^{e_{j+1}} \frac{1}{e_{j+1}-e_j} \cdot (\frac{e_{j+1}-r}{e_{j+1}-e_j})^{|K|-1}\, dr \quad (8)$$

Equation 7 is obtained by observing that X_i is the nearest neighbor of q if (1) it is at a distance r from q (with probability $\frac{1}{e_{j+1}-e_j}$) and (2) other objects in K are more than a distance of r from q (with probability $(1 - \frac{r-e_j}{e_{j+1}-e_j})$. The detailed explanation of this formula can be found in [5]. Note that we can change the integration bounds from $[n_i, f_i]$ to $[e_j, e_{j+1}]$, since we only consider the objects in K, which are known to be within $[e_j, e_{j+1}]$. Equation 8 leads to the conclusion $p_i = \frac{1}{|K|}$, and the lemma is thus proved. □

Lemma 4 itself can be used to obtain the qualification probabilities for the scenario when there is only one subregion (i.e., $M = 1$). Here, the distances of all objects in the candidate set C from q are located in one subregion, S_1. Using Lemma 4, we obtain $p_i = \frac{1}{|C|}, \forall X_i \in C$.

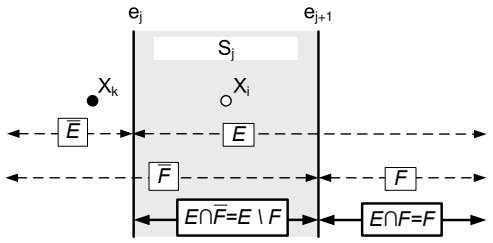

Fig. 9 Correctness proofs of L-SR and U-SR.

We can now prove the correctness of L-SR and U-SR. Let us examine when c_j, the number of objects with non-zero subregion probabilities in subregion S_j, is equal to 1. In fact, this scenario happens to subregion S_1, i.e., $j = 1$, since only this region can accommodate a single distance pdf (e.g., $d_1(r)$ in Figure 8). If we also know that distance R_i is in subregion S_j, then X_i must be the nearest neighbor. Thus, both the lower and upper subregion qualification probability bounds ($q_{ij}.l$ and $q_{ij}.u$) are equal to 1, as shown in Equations 1 and 5.

For the case $c_j > 1$, we derive the subregion qualification probability, q_{ij}. Let E denote the event that "all objects in the candidate set C have their actual distances from q not smaller than e_j". Also, let $\bar{E}$ be the complement of event E, i.e., "there exists an object whose distance from q is less than e_j". We also let F be the event "$\forall X_k \in C$, where $k \neq i$, $R_k \geq e_{j+1}$", and $\bar{F}$ be the event "$\exists X_k \in C$ s.t. $k \neq i \wedge R_k < e_{j+1}$". Figure 9 illustrates these four events.

If $Pr(E)$ denotes the probability that event E is true, then $Pr(E) = 1 - Pr(\bar{E})$. We also have $Pr(F) = 1 - Pr(\bar{F})$. Let N be the event "Object X_i is the nearest neighbor of q". Then, using the law of total probability, we have:

$$q_{ij} = Pr(N|E) \cdot Pr(E) + Pr(N|\bar{E}) \cdot Pr(\bar{E}) \quad (9)$$

If $\bar{E}$ is true, there is at least one object X_k whose distance R_k is not larger than e_j (Figure 9). Since $R_k < R_i$, object X_k must be closer to q than object X_i. Consequently, $Pr(N|\bar{E}) = 0$, and Equation 9 becomes:

$$q_{ij} = Pr(N|E) \cdot Pr(E) \quad (10)$$

which again, by using the law of total probability, can be rewritten as

$$q_{ij} = Pr(N|E \cap F) \cdot Pr(E \cap F) + Pr(N|E \cap \bar{F}) \cdot Pr(E \cap \bar{F}) \quad (11)$$

If $E \cap F$ is true, then all objects except X_i have their distances from q not smaller than e_{j+1}. Since $R_i \leq e_{j+1}$, it must be the nearest neighbor. Thus, $Pr(N|E \cap F) = 1$. We also have $E \cap F = F$ since $E \rightarrow F$. So Equation 11 is reduced to:

$$q_{ij} = Pr(F) + Pr(N|E \cap \bar{F}) \cdot Pr(E \cap \bar{F}) \quad (12)$$

Next, suppose $E \cap \bar{F}$ is true. Then, in addition to X_i, $m \geq 1$ other object(s) is (are) on the left of e_{j+1}. Since E is also true, the values of R_k for all these m objects must also be in in S_j. Using Lemma 4, we can then deduce that $Pr(N|E \cap \bar{F}) = \frac{1}{m+1}$. The minimum value of $Pr(N|E \cap \bar{F})$ is $\frac{1}{c_j}$, which happens when $\forall X_k \in C$, where $k \neq i$ and $s_{kj} > 0, X_k \in S_j$. The maximum value of $Pr(N|E \cap \bar{F})$ is $\frac{1}{2}$, which happens when $m = 1$. Thus we have

$$\frac{1}{c_j} \leq Pr(N|E \cap \bar{F}) \leq \frac{1}{2} \tag{13}$$

Notice that:

$$Pr(E \cap \bar{F}) = Pr(E) - Pr(F) \tag{14}$$

Thus, the final steps are to obtain $Pr(E)$ and $Pr(F)$. To obtain $Pr(E)$, note that the probability that an object X_k's distance is e_j or more is simply $1 - D_k(e_j)$. We then multiply all these probabilities, as $\prod_{R_k \geq e_j \wedge k \neq i}(1 - D_k(e_j))$. This can be simplified to:

$$Pr(E) = \prod_{U_k \cap S_j \neq \emptyset \wedge k \neq i} (1 - D_k(e_j)) \tag{15}$$

since any object whose subregion probability is zero in S_j must have the distance cdf at e_j, i.e.,$D_k(e_j)$ equal to zero.

Similarly, $Pr(F) = \prod_{U_k \cap S_{j+1} \neq \emptyset \wedge k \neq i}(1 - D_k(e_{j+1}))$.

Combining Equations 12, 13, 15, and 14, we can obtain the lower and upper bounds of q_{ij}, i.e., $q_{ij}.l$ and $q_{ij}.u$, as stated in Equations 1 and 5.

4.4 Result-based Verifiers

This class of verifiers makes use of the information deduced by the subregion-based verifiers. The intuition is that if we know the maximum qualification probabilities of any of the $|C| - 1$ objects in C, we can easily derive the lower probability bound of the remaining object in C. Specifically, given an object X_i, its lower probability bound, $p_i.l$, can be computed as:

$$p_i.l = \max(1 - \sum_{X_k \in C \wedge k \neq i} p_k.u, 0) \tag{16}$$

This is because the total probabilities of the $|C| - 1$ objects (apart from X_i) is at most $\sum_{X_k \in C \wedge k \neq i} p_k.u$. Since the sum of qualification probabilities of all objects in C must be equal to one, $p_i.l$ can be obtained by Equation 16.

By using this principle, the Lower Probability Bound (**L-PB**) verifier refreshes the objects' lower probability bounds. It makes use of the new upper bound information yielded by the U-SR verifier (Figure 5). To implement L-PB, note that Equation 16 can be written as

$$p_i.l = \max(1 - p_{total} + p_i.u, 0) \tag{17}$$

where $p_{total} = \sum_{X_k \in C} p_k.u$. Thus, we can compute p_{total} first (in $O(|C|)$ time), and use Equation 17 to derive the lower probability bound of every object in C (also in $O(|C|)$ time). The total complexity for L-PB is thus equal to $O(|C|)$.

The other result-based verifier, namely, the Upper Probability Bound (**U-PB**) verifier, derives an object's upper probability bound by using the lower probability bounds of other objects. It can be used after the RS or L-SR verifier, which yield lower bound information (see Figure 5). The basic principle is the same as that of L-PB, where the upper bound probability of an object is given by:

$$p_i.u = \max(1 - \sum_{X_k \in C \wedge k \neq i} p_k.l, 0) \tag{18}$$

4.5 Incremental Refinement

As discussed in the beginning of Section 4, some objects may still be unclassified after all verifiers have been applied. The exact probabilities of these objects must then be computed or "refined'. This can be expensive, since numerical integration has to be performed over the object's distance pdf [5]. This process can be performed faster by using the information provided by the verifiers. Particularly, the probability bounds of each object in each subregion (i.e., $[q_{ij}.l, q_{ij}.u]$) have already been computed by the verifiers. Therefore, we can decompose the refinement of qualification probability into a series of probability calculations inside subregions. Once we have computed the probability q_{ij} for subregion S_j, we collapse $[q_{ij}.l, q_{ij}.u]$ into a single value q_{ij}, update the probability bound $[p_i.l, p_i.u]$, and test this new bound with classifier. We repeat this process with another subregion until we can classify the object. This "incremental refinement" scheme is usually faster than directly computing qualification probabilities, since checking with a classifier is cheap, and performing numerical integration on a subregion is faster than on the whole uncertainty region, which has a larger integration area than a subregion. The time complexity of incremental refinement has a worst-case complexity of $O(|C|^2 M)$, as detailed in [5].

We store the subregion probabilities (s_{ij}) and the distance cdf values ($D_i(e_j)$) for all objects in the same subregion as a list. These lists are indexed by a hash table, so that the information of each subregion can be accessed easily. The space complexity of this structure is $O(|C|M)$. It can be extended to a disk-based structure by partitioning the lists into disk pages. The complexities of the verifiers are shown in Table 3. The complexity of running all verifiers (including initialization and

Verifier	Probability Bound	Cost
Subregion-based		
RS	Upper	$O(\|C\|)$
L-SR	Lower	$O(\|C\|M)$
U-SR	Upper	$O(\|C\|M)$
Result-based		
L-PB	Lower	$O(\|C\|)$
U-PB	Upper	$O(\|C\|)$

Table 3 Complexity of Verifiers.

sorting of candidate set objects) is $O(|C|(\log|C|+M))$, and is lower than the evaluation of exact probabilities ($O(|C|^2M)$).

We now explain how to arrange the verifiers in the order shown in Figure 5. We place the subregion-based verifiers in the ascending order of running costs (see Table 3). For L-SR and U-SR, which have the same complexity, their relative ordering is dependent on factors like probability threshold P and the density of the dataset used. If P is large and the dataset is dense, then U-SR can be evaluated first, since the number of candidate objects tend to be large. The qualification probabilities of these objects will also be small, so that they can be easily pruned by U-SR. On the other hand, for sparse dataset and small P, L-SR can be used first, since the candidate objects, with potentially high probabilities, can be accepted by L-SR more easily. In Figure 5, the result-based verifier L-PB is executed after U-SR, and U-PB after U-SR. Although L-PB can also be used after RS, we do not show it in the framework because in most of our experiments, the fraction of distance pdfs of objects beyond f_{min} is small. Hence, their upper probability bounds can only be slightly reduced by RS. Consequently, most of the lower probability bounds derived by L-PB are equal to zero. Therefore, we only use L-PB after the execution of U-SR.

4.6 Extension

We conclude this section by discussing how our solution can be extended to handle the following issues: (1) partial query evaluation; (2) computing subregion probabilities in an on-demand manner; and (3) disk-based query processing.

1. Partial Query Evaluation. As mentioned in Section 1, sometimes a query user may not want to wait for the whole query result to be generated. We now discuss how partial query results can be returned to the user before the query finishes. Particularly, the verification framework can be easily modified in the following way: instead of verifying the candidate objects in a verifier and send them to the classifier together, each object is completely verified (and possibly refined). If it is discovered by the classifier as acceptable, then the object will be returned immediately to the user. This process is repeated until all the candidate objects have been examined.

The order of verifying objects can affect how fast the objects are returned to the user. For example, in Figure 8, object X_1 requires longer time to process (since it spans five subregions), whereas X_3, using three subregions only, can be verified and classified faster. We therefore propose two different policies of partial evaluation:

- **Leftmost Object First (LOF).** This policy considers the candidate objects in the ascending order of $|\frac{n_i+f_i}{2}|$. For example, in Figure 8, the evaluation order is: X_1, X_2, X_3.
- **Rightmost Object First (ROF).** This policy evaluates objects in the descending order of $|\frac{n_i+f_i}{2}|$. In Figure 8, the evaluation order is: X_3, X_2, X_1.

We also compare experimentally the degree of "responsiveness" of the above policies. In particular, we propose two metrics for quantifying responsiveness:

- **Fraction of Classified Objects:** this metric quantifies the portion of candidate objects that have been processed.
- **Sum of Lower-probability-bounds of Classified Objects**: this evaluates the sum of the lower-bounds of the qualification probabilities, i.e., $p_i.l$, for the objects that have been classified. Intuitively, a user may want to see objects with higher qualification probabilities, since these objects may be considered more important than those with smaller probabilities. Hence, if a policy scores high in this metric, it means this policy prioritizes on objects with higher qualification probabilities.

We will explain the details of the experimental results on these policies in Section 6.

2. On-Demand Subregion Evaluation. So far, we have assumed that the distance pdfs and cdfs of all objects are precomputed before the verifiers are used. However, obtaining distance pdfs/cdfs can sometimes be costly. In fact, it is not always necessary to generate these data beforehand – we can just fetch the subregion information when needed. If the query is finished after all objects have been verified by visiting some of the subregions, we save the effort for not generating subregions that are not used, as well as the space for maintaining the subregion information. Consider an extreme case where the RS verifier uses the rightmost subregion to verify objects. If it happens that all objects are successfully tested by RS, then there is no need to use any subregions required by L-SR and U-SR.

In order to use this "on-demand" method, we need to change the solution framework: Instead of completely verifying all the subregions of an object, we sequentially visit each subregion, verify all objects with non-zero distance pdfs inside that subregion, before retrieving the next subregion. This change is feasible, since for our verifiers, the qualification probability bounds within a subregion can be computed without considering other subregions. In Section 6, we study the effectiveness of this on-demand approach experimentally.

3. Disk-based Access. We have assumed that all non-pruned candidate objects are stored in the main memory. If the uncertainty regions of the objects are large, it is possible that they have a lot of overlap with each other. Then, the main memory may not be big enough to store all candidate objects. To handle this problem, we can extend our query processing algorithm in the following way: we store the subregion probability values of all objects within the same subregion S_j, in a bucket B_j of pages. Hence, if we have M subregions, M buckets will be needed. We then visit each bucket sequentially, compute the probability bounds of each object by using verifiers, and update its probability (for details, please see Section 4.5). This process is repeated until all buckets have been visited. Notice that in this process, each bucket is retrieved only once for minimizing the I/O access.

To generate page buckets, we first order the candidate objects according to their maximum distances from q, with the aid of some external sorting algorithm. We then store the subregion information in buckets based on their histogram information. The performance of this procedure can also be improved by using approaches similar to the "on-demand" method discussed earlier. Particularly, we can load the distance pdf/cdf information to the bucket only when its corresponding subregion is visited by verifiers. Another issue is that during subregion construction, we may want to avoid repeatedly reading all the pdf data of an object into the main memory. This is because a pdf can be composed of a large number of histograms, and so loading it to the main memory is expensive. We may use a small index (e.g., a hash table) for the histogram bars for the pdf, and retrieve only the relevant part of the pdf to the main memory.

5 Incremental and Lazy Evaluation of Continuous C-PNN

We now investigate how continuous C-PNN (or CC-PNN in Definition 2) can be efficiently evaluated. As shown in Figure 4, evaluating a C-PNN requires three steps, namely filtering, verification, and refinement. In an environment (e.g., sensor network) where update arrivals are frequent, repeating these processes for a CC-PNN can be quite expensive. We observe that given the previous query results, it is often not necessary for the whole CC-PNN query to be recomputed. We thus develop an incremental update method to reduce the amount of I/O for finding the candidate set in the filtering stage (Section 5.1). We also present a lazy update method where verification and refinement can be effectively delayed without query tolerance requirements, as discussed in Section 5.2.

In subsequent discussions, we assume that the initial answer of the CC-PNN has been obtained at the time it starts running in the system. This can be accomplished by executing the corresponding C-PNN query over the current database. Also, let $\mathbb{X}(t_0) \in X$ be the old query result at time t_0, and $\mathbb{X}(t_u)$ be the newest query result (where $t_u > t_0$), due to the arrival of a set U of object updates at time t_u. We assume that U has been applied to the database before the query is re-evaluated, and U is stored in the main memory. Table 4 shows the symbols used by our approach.

Symbol	Meaning
t_0	time before U is applied to database
t_u	time after U is applied to database
$\mathbb{X}(t)$	Query result at time t
U	Set of update changes, with $U \subseteq X$
C	Candidate set at time t_0
C'	Candidate set at time t_u
D	Set of objects removed from C
A	Set of objects inserted to C
p_i'	Prob. of X_i in $\mathbb{X}(t_u)$

Table 4 Symbols for Continuous C-PNN.

5.1 Incremental Derivation of the Candidate Set

Let C be the candidate set produced by the filtering stage that yields the old query result $\mathbb{X}(t_0)$ (i.e., prior to the arrival of updates U). Let C' be the new candidate set, after applying U to the database. We assume both C and C' can be stored in the main memory. Traditionally, finding C' requires a branch-and-prune processing of the R-tree index [8,30], which can require a lot of I/O overhead. We now present a method that considers C and U, in order to reduce the I/O cost of discovering C'.

To understand how our method works, recall from Section 4.1 that f_{min} is the minimum value of the far points of all the objects in the candidate set C. If $U \cap C = \emptyset$ and none of the objects in U has a near point

smaller than f_{min}, then we can be assured there is at least one object in C completely closer to all objects in U. Therefore, the candidate set C remains unchanged, i.e., $C' = C$. In reality, some object in U may have a near point less than f_{min}, in which case the contents of C may be changed. However, if only a small fraction of the objects in C is affected, we can still find out C without using scanning the index. Let f'_{min} be the minimum of the far points of objects in C'. Also, if an object in C' is also in U, then there is no need to retrieve the object from the database. Otherwise, there are two cases for considering whether an object X_i in C' can be recovered from C, without loading it from the database.

- **Case 1:** $f'_{min} \leq f_{min}$. Any object X_i in C' must have its near point n_i such that $n_i \leq f'_{min}$. This implies $n_i \leq f_{min}$, and so $X_i \in C$. Thus, $C' \subseteq C$, and there is no need to load any object from the index.
- **Case 2:** $f'_{min} > f_{min}$. If the near points of objects are within $[f_{min}, f'_{min}]$, they cannot be found in C (since C only contains objects with near points closer than f_{min}). Thus it is necessary to retrieve them from the database.

Algorithm 1 uses these observations to derive the new candidate set C'. It is first initialized to $C \cup U$ (Step 1) i.e., adhering the update set U to C. The value of f'_{min} is set to be the minimum value of f_i among all objects in C' (Step 2). In Step 3, we initialize N to a null set, the set which contains objects with near points inside $[f_{min}, f'_{min}]$ (for $f'_{min} > f_{min}$). For this case, we need an range query on the R-tree index to retrieve candidates and store them in N (Steps 4 and 5). We then find the minimum f_i of all objects in N, and decide whether f'_{min} should be updated (Steps 6 and 7). Next, we insert the objects in N to C' (Step 8). We then remove objects from C', whose minimum distances from the query point are larger than f'_{min} (Steps 9 to 11).

The algorithm also produces two sets of objects, namely A and D, in Steps 12 and 13. These two sets essentially records the *changes* that yield C'. In particular, D, called the *deletion set*, contains all objects that are in C but not in C'. On the contrary, A, called the *insertion set*, contains objects that are in C' but not in C. Notice that if an object X_i in C has its value (e.g., pdf) updated, then we treat the update of X_i as a deletion followed by an insertion. In other words, we have:

$$C' = (C \setminus D) \cup A \tag{19}$$

As shown in Step 12, D is the union of the set of objects removed from C (i.e., $C \setminus C'$) and the set of objects updated (i.e., $C \cap U$). Step 13 illustrates that A is the union of the objects inserted to C (i.e., $C' \setminus C$) and the objects updated in C that also appears in C' (i.e., $C' \cap C \cap U$).

```
    input  : C, U, f_min;
    output: C', f'_min, D, A;
 1  C' ← C ∪ U;
 2  f'_min ← min(f_i | X_i ∈ C');
 3  N ← ∅;
 4  if f'_min > f_min then
 5      N ← objects where n_i ∈ [f_min, f'_min];
 6      if f'_min > min(f_i | X_i ∈ N) then
 7          f'_min ← min(f_i | X_i ∈ N);
 8  C' ← C' ∪ N;
 9  for each X_i ∈ C' do
10      if n_i > f'_min then
11          C' ← C' \ {X_i};
12  D ← (C \ C') ∪ (C ∩ U) ;
13  A ← (C' \ C) ∪ (C' ∩ C ∩ U);
```

Algorithm 1: Updating the Candidate Set.

The complexity of this algorithm is $O(|X|)$ (mainly due to the execution of range query in Step 5). In practice, the complexity is much lower since $[f_{min}, f'_{min}]$ is a small range, and the sizes of the sets being considered in the algorithm are not large. Next, we show how A and D are extensively used.

5.2 Lazy Evaluation

Once the objects in the candidate set have been constructed for the refreshed data, verification and refinement can be applied to these objects. In this section, we examine how the information of the previous query results can be *reused* to facilitate this process. In particular, for an object previously in the answer set $\mathbb{X}(t_0)$, we develop a new and efficient method to decide whether this object is still in $\mathbb{X}(t_u)$, *without* going through verification and refinement in Section 4. Notice that this new method still maintains the query correctness requirements (i.e., threshold and tolerance). We call our method a *lazy* approach, since the actual probability computation of an object is deferred until the point that the correctness of the query cannot be satisfied.

The main idea of our approach is to develop the probability bounds of the candidate set objects, by using the information of insertion set A and deletion set D. Again, C' is equal to $(C \setminus D) \cup A$. We first compute the qualification probability bounds of objects in A. These are the objects that are new to C, or those in C that are updated. For objects that are neither deleted nor updated, i.e., the set $C \setminus D$, we use the following lemmas to derive their probability bounds. We first con-

sider insertion and deletion separately. We then show how they can be combined together.

Lemma 5 *Suppose $A = \emptyset$. Then, if $d = \sum_{X_j \in D}(p_j)$, p_i and p'_i are the qualification probabilities of X_i at time t_0 and t_u, then*

$$p'_i \geq p_i, \forall X_i \in C' \tag{20}$$

$$p'_i \leq p_i + d, \forall X_i \in C' \tag{21}$$

Proof By assuming $A = \emptyset$, we have $C' = C \setminus D$ from Equation 19, that is, C' is the result of applying removal of objects from C. Also, $C' \subseteq C$. To understand Inequality 20, notice that the qualification probability of object X_i is conceptually computed by considering all the possible worlds (i.e., combinations) of the candidate objects [5,8]. Consider a possible world where X_i is not the nearest neighbor of q before the database changes. If the object $X_j \in D$ is the nearest neighbor and it is deleted, then X_i has more chance to be the nearest neighbor of q. In another possible world, if X_i is the nearest neighbor before deletion, then it remains as the nearest neighbor. Hence, the probability of X_i for being the nearest-neighbor of q increases, and so $p'_i \geq p_i$.

Moreover, with our conclusions of result-based verifiers (Equation 18), we can have

$$p'_i.u = 1 - \sum_{X_j \in C' \wedge j \neq i} p'_j.l \tag{22}$$

$$= 1 - \sum_{X_j \in C' \wedge j \neq i} p_j \tag{23}$$

$$= p_i + \sum_{X_k \in D} p_k \tag{24}$$

$$= p_i + d \tag{25}$$

□

Note that the second deduction step is obtained by using p_j as a lower bound of p'_j (Equation 20).

We may only know the objects' probability bounds (e.g., in C-PNN, we only know $[p_i.l, p_i.u]$ but not p_i). However, we can easily extend Lemma 5 to handle this. Let $d.u = \sum_{X_j \in D}(p_j.u)$, then $d.u \geq d$. Also, since $p_i.u \geq p_i$, then Equation 21 can be rewritten as:

$$p'_i \leq p_i.u + d.u, \forall X_i \in C' \tag{26}$$

Next, we study the case with no deletion.

Lemma 6 *Suppose $D = \emptyset$. If $s = \sum_{X_j \in A}(p'_j)$, then*

$$p'_i \leq p_i, \forall X_i \in C \tag{27}$$

$$\sum_{X_i \in C} (p_i - p'_i) = s \tag{28}$$

$$p'_i \geq p_i - s, \forall X_i \in C \tag{29}$$

Proof Lemma 6 can be obtained by using the result of Lemma 5 by considering the update backward in time. Specifically, we consider C as the candidate set obtained by deleting the objects in A from C'. We can then obtain Lemma 6. □

Let $s.u = \sum_{X_j \in A}(p'_j.u)$. Then Equation 29 can be written as

$$p'_i \geq p_i.l - s.u, \forall X_i \in C \tag{30}$$

The next important fact comes from Lemma 6 and Lemma 5, which allows us to handle the case when an update is present, i.e., both A and D are not null.

Lemma 7 *If $d = \sum_{X_j \in D}(p_j)$, and $s = \sum_{X_j \in A}(p'_j)$, then, for object $X_i \in C \setminus D$, we have:*

$$p_i - s \leq p'_i \leq p_i + d \tag{31}$$

Proof Here we only consider the objects in $C \setminus D$, i.e., those that are neither updated nor deleted from C. Since $C' = (C \setminus D) \cup A$ (Equation 19), we can consider the effect of D and A on C in two separate steps, i.e., we first produce an "intermediate" candidate set where $C^- = C \setminus D$, and then use C^- to obtain C, where $C = C^- \cup A$. First, we consider $C^- = C \setminus D$. By using Lemma 5, we know that the qualification probability of object X_i after deletion, denoted by p_i^-, must satisfy the following:

$$p_i \leq p_i^- \leq p_i + d \tag{32}$$

Next, when we apply insertion to C^-, and through Lemma 6, we have:

$$p_i^- - s \leq p'_i \leq p_i^- \tag{33}$$

By combining Inequalities 32 and 33, we obtain Lemma 7. □

We can similarly derive the following result for the probability bounds of p_i:

$$p_i.l - s.u \leq p'_i \leq p_i.u + d.u \tag{34}$$

To utilize these results, we have developed Algorithm 2. Its main goal is to utilize the difference information about C and C', in order to derive the probabilities for candidate objects. It assumes the probability bounds of objects in the insertion set A have already been computed. Steps 1 and 2 generate the maximum deviation of the new probabilities from the old ones. Then, for every object not deleted or updated (i.e., $C \setminus D$), we compute their new lower and upper bound probabilities, using Inequality 34 (Steps 4-5). These new bound information can then be checked against the classifier (Step 6). If X_i cannot be classified, it is passed to the verification phase developed for C-PNN (Step 7). As we can see, the additional cost of this "lazy" way of computing the probability is the computation of the bounds and classification (Steps 4-6), which can be done $O(|C \setminus D|)$ time.

```
input : C, D, A
d.u ← Σ_{X_i ∈ D} p_i.u;
s.u ← Σ_{X_i ∈ A} p'_i.u;
for X_i ∈ C \ D do
    p'_i.l ← p_i.l − s.u;
    p'_i.u ← p_i.u + d.u;
    Pass (X_i, p'_i.l, p'_i.u) to classifier;
    if X_i cannot be classified then
        Verify X_i;
```

Algorithm 2: Lazy computation of probability bounds

6 Experimental Results

We have performed experiments to examine our solution. We describe the simulation setup in Section 6.1. We present the results for C-PNN in Section 6.2, and CC-PNN in Section 6.3.

6.1 Experimental Setup

We use the *Long Beach* dataset[3], where the 53,144 intervals, distributed in the x-dimension of 10K units, are treated as uncertainty regions with uniform pdfs. We also use synthetic datasets, where all items are distributed in the 1D space of size $[0, 10000]$. The number of items varies from 1K to 100K. Each data value is represented by an uncertain interval and a uniform pdf. The widths of the uncertain intervals are uniformly distributed in [10, 100]. For both real and synthetic datasets, the uncertainty pdf is assumed to be uniform. We also consider Gaussian uncertainty pdf in some cases. Unless stated otherwise, the results presented are performed on the real dataset.

For each C-PNN, the default values of threshold (P) and tolerance (Δ) are 0.3 and 0.01 respectively. We assume a user of the C-PNN is not interested in small probabilities, by assuming $P > 0.1$. The query points are randomly generated. Each point in the graph is an average of the results for 100 queries.

We also simulate the evaluation of CC-PNN based on the real data set. At each time point, 1% of data objects are randomly chosen to change their values. For each update, the difference between the center of the new uncertain region and its previous value is uniformly distributed in $[0, 100]$. The difference between the size of the new uncertain region and the previous one is uniformly distributed in $[0, 50]$. Each point in the graph is an average of 20 queries and 20 time points.

The experiments, written in Java, are executed on a PC with an Intel T2400 1.83GHz CPU and 1G of main memory. We implemented the filtering phase by using the R-tree library in [31].

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

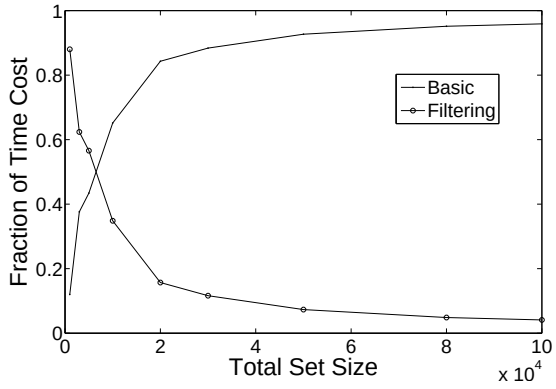

Fig. 10 Basic vs. Filtering.

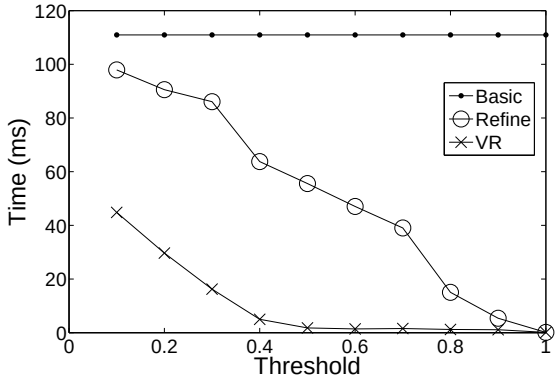

Fig. 11 Probability Computation time vs. P.

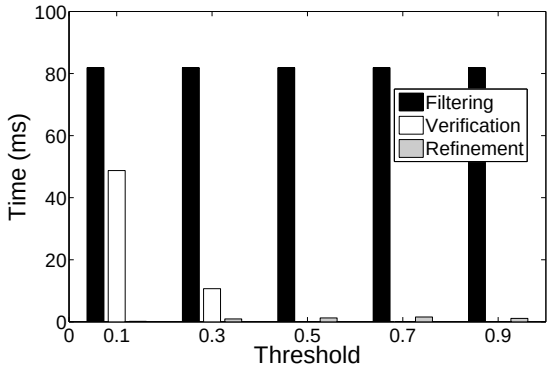

Fig. 12 Analysis of *VR*.

6.2 Results for C-PNN

We compare three strategies of evaluating a C-PNN. The first method, called *Basic*, evaluates the exact qualification probabilities using the formula in [5]. The second one, termed *VR*, uses probabilistic verifiers and incremental refinement. The last method (*Refine*) skips

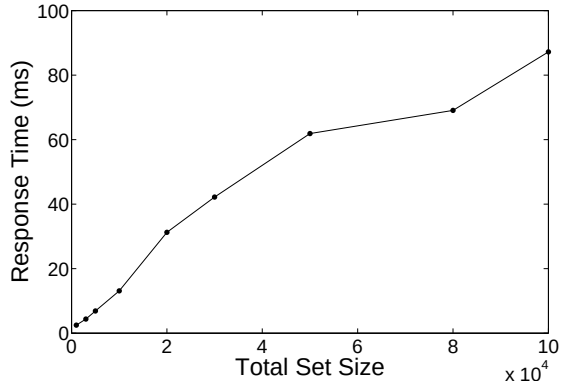

Fig. 13 Scalability.

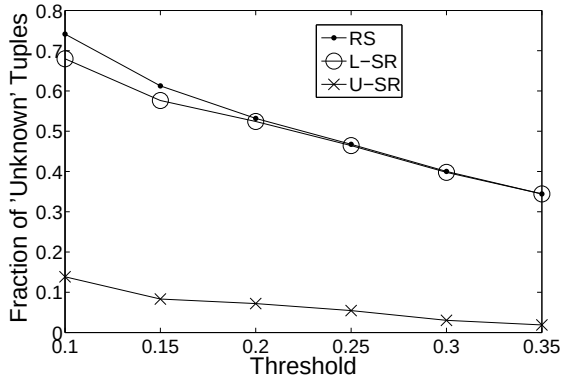

Fig. 15 RS, L-SR, and U-SR.

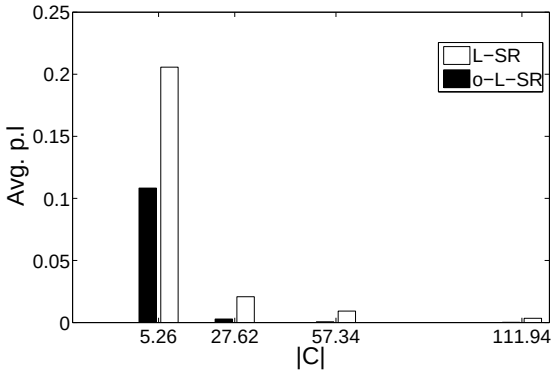

Fig. 14 A tighter L-SR.

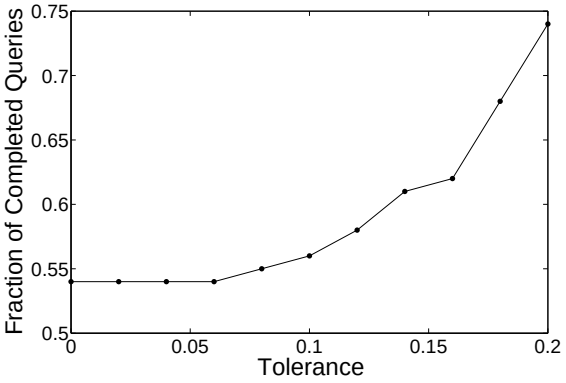

Fig. 16 Effect of Δ.

verification and performs incremental refinement directly. All these strategies assume the candidate set is ready i.e., filtering has already been applied to the original dataset. On average, the candidate set has 96 objects.

1. Cost of the Basic Method. We first compare the time spent on the Basic with filtering. Figure 10 shows that the fraction of total time spent in these two operations on synthetic data sets with different data set sizes. As the total table size increases, the time spent on the Basic solution increases more than filtering, and so its running time starts to dominate the filtering time when the data set size is larger than 5000. As we will show soon, other methods can alleviate this problem.

2. Effectiveness of Verification. In Figure 11, we compare the time required by the three evaluation strategies under a wide range of values of P. Both *Refine* and *VR* perform better than *Basic*. At $P = 0.3$, for instance, the costs for *Refine* and *VR* are 80% and 16% of *Basic* respectively. The reason is that both techniques allow query processing to be finished once all objects have been determined, without waiting for the exact qualification probabilities to be computed. For large values of P, most objects can be classified as *fail* quickly when their upper probability bounds are detected to be lower than P. Moreover, *VR* is consistently better than *Refine*; it is five times faster than *Refine* at $P = 0.3$, and 40 times faster at $P = 0.7$. This can be explained by Figure 12, which shows the execution time of filtering, verification and refinement for *VR*. While the filtering time is fixed, the refinement time decreases with P. The verification takes only 1ms on average, and it significantly reduces the number of objects to be refined. In fact, when $P > 0.3$, no more qualification probabilities need to be computed. Thus, *VR* produces a better performance than *Refine*.

Figure 13 examines the scalability of *VR*. We can see that the overall query response time of *VR* scales well with the dataset size.

3. Comparison of Verifiers. We first compare the performance of the L-SR verifier presented in this paper with its previous version, which were presented in [19] (let us call the old version the *o-L-SR* verifier). Figure 14 compares their performance in terms of the average probability lower bounds over synthetic datasets. As we can see, the L-SR yields higher average lower-bounds than o-L-SR. This is because the current version of L-SR theoretically derives a tighter lower bound than o-L-SR (by having an additional second term in

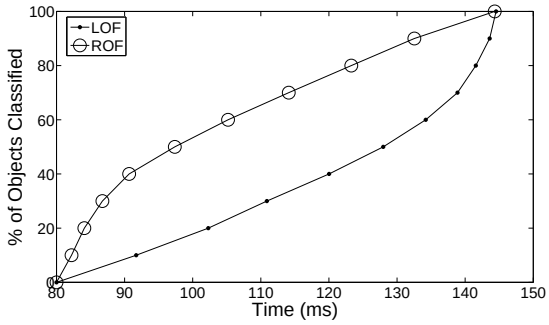

Fig. 17 LOF, ROF vs. Complete (Response).

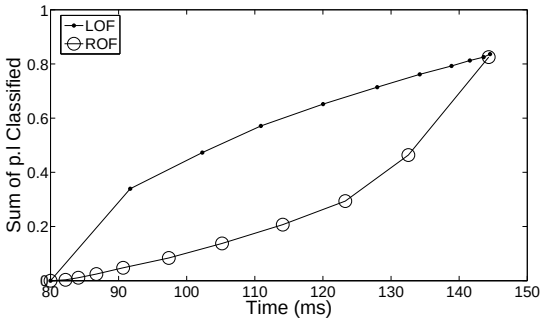

Fig. 18 LOF, ROF vs. Complete (QP).

Equation 1). Thus, the current L-SR verifier performs better, and is used throughout our experiments.

Then, Figure 15 shows the fraction of objects labeled *unknown* after the execution of verifiers in the order: {RS, L-SR, U-SR}. This fraction reflects the amount of work needed to finish the query. At $P = 0.1$, about 75% of *unknown* objects remain after the RS is finished; 7% more objects are removed by L-SR; 15% *unknown* objects are left after the U-SR is executed. When P is large, RS and U-SR perform better, since they reduce upper probability bounds, so that the objects have a higher chance of being labeled as *fail*. L-SR works better for small P (as seen from the gap between the RS and L-SR curves). L-SR increases the lower probability bound, so that an object is easier to be classified as *satisfy* at small P. In this experiment, U-SR performs better than L-SR. This is because the candidate set size is large (about 96 objects), so that the probabilities of the objects are generally quite small. Since U-SR reduces their upper probability bounds, they are relatively easy to be verified as *fail*, compared with L-SR, which attempts to raise their lower probability bounds.

L-PB and U-PB. We also examine the performance of the result-based verifiers, i.e., L-PB and U-PB. We found that they are not very effective when the size of the candidate set is large. Consider the L-PB verifier, which derives an objects X_i' lower probability by subtracting the sum of other objects' upper bounds

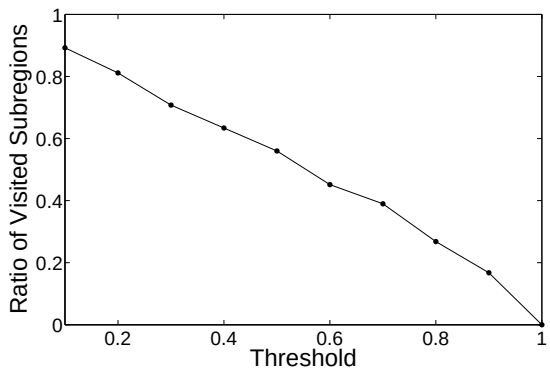

Fig. 19 The On-Demand Method.

from one (Equation 16). If there is a large number of these objects, then their sum can easily reach one, so that a trivial lower bound is yielded for X_i (i.e., equal to zero). A similar problem happens with the U-PB verifier. We note that, however, the costs of these verifiers are low (c.f. Table 3). Moreover, they are more useful in smaller datasets. We have tested the performance of these verifiers over a synthetic dataset of 1000 objects, and we found that the performance is improved by about 5%.

4. Effect of Tolerance. Next, we measure the fraction of queries finished after verification under different tolerance. Figure 16 shows that as Δ increases from 0 to 0.2, more queries are completed. When $\Delta = 0.16$, about 10% more queries will be completed than when $\Delta = 0$. Thus, the use of tolerance can improve query performance.

5. Partial Query Evaluation. We then compare the effectiveness of the LOF and the ROF policies, using the two responsiveness metrics presented in Section 4.6. We first test the performance on the fraction of objects classified. Figure 17 shows the result over the query execution time. During the first 80 ms, the time is spent on filtering, and so no objects are classified during this period. We observe that ROF classifies more fraction of objects than LOF at early stages. This is because ROF examine objects farthest away from the query point first. These objects have fewer subregions (created due to the near points of objects) than those that are close to the query points. Hence, they need less time to process than the LOF policy, which considers objects with average distances closest to the query point first.

On the other hand, LOF performs better than ROF in terms of the sum of the lower-bound probabilities for the classified objects, as shown in Figure 18. This is because LOF examines the objects which are potentially nearest to the query point. Therefore, the objects returned tend to have higher qualification probabilities.

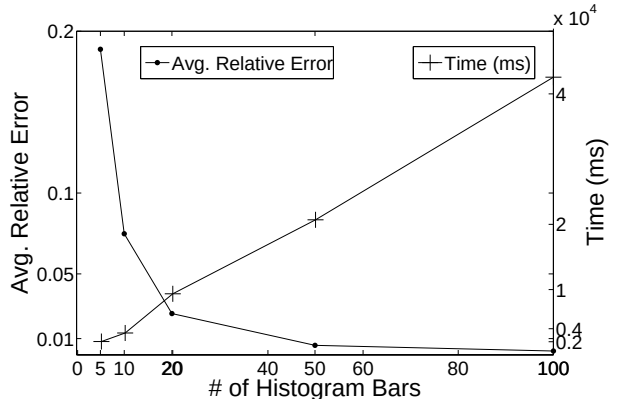

Fig. 20 Relative error vs. pdf precision.

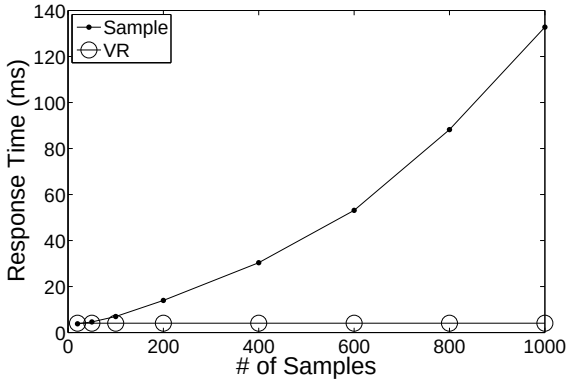

Fig. 22 Effect on the Number of Samples.

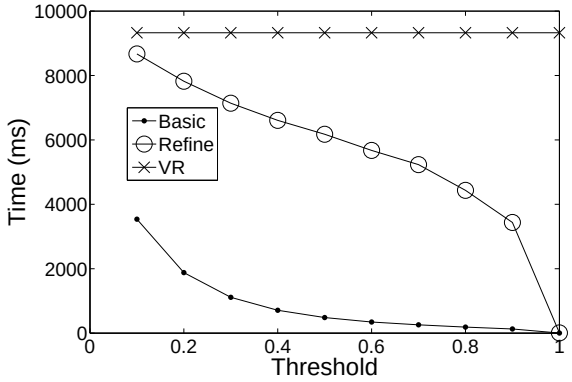

Fig. 21 Gaussian pdf.

This metric can be useful if the user values objects that yield high probabilities.

6. On-Demand Subregion Evaluation. Next, we analyze the effect of deriving subregion information in an online manner, as discussed in Section 4.6. Figure 19 shows the average fraction of subregions visited by the time a query is completed. Under a higher probability threshold, fewer subregions are visited before determining that an object need to be pruned. Hence, the on-demand method is able to save resources for deriving the subregion information (i.e., the distance pdf and cdf) at a high probability threshold.

7. Gaussian pdf. We now examine the effect of using a Gaussian distribution as the uncertainty pdf for each object. Each Gaussian pdf, has a mean at the center of its range, and a standard deviation of 1/6 of the width of the uncertainty region. Since we approximate a Gaussian pdf as a histogram, we first conduct a sensitivity test to see how precise a Gaussian pdf should be represented (in terms of the number of histogram bars). We notice that the result qualification probabilities become stable when more than 300 histogram bars are used. Hence we consider this qualification probability as the "true" answer. In Figure 20, we can see that as more histogram bars are used, a smaller relative error (compared with the true probability) is attained. However, it also takes a longer time to obtain the answer. When a pdf is represented as 20 histogram bars, the average relative error is less than 5%, and the time required is about 10 seconds. Hence, we adopt this setting for Gaussian pdf.

Figure 21 shows the probability computation time for Gaussian pdf. Again, *VR* outperforms the other two methods. The saving is more significant than when uniform pdf is used. This is because the relatively more expensive evaluation of Gaussian pdf can be effectively avoided by the verifiers. Hence, our method also works well with Gaussian pdf.

8. Comparison with Sample-based Method. In the final experiment of C-PNN, we compare our method with a sample-based method described in [9]. For each uncertain object, we randomly choose some samples from its pdf. In this experiment, we assume that the pdf is uniform. As discussed in Section 2, the method in [9] first clusters samples into small groups. In our experiment, each cluster contains an average of 10 samples. Figure 22 compares our method and the sample-based method, in terms of the query response time, for a synthetic dataset with $3,000$ objects. We see that the response time of the sample-based method, which does not use the probability threshold and tolerance during computation, increases sharply with the number of samples. Moreover, at least 400 samples are required for achieving a relative error of 10%. Under this setting, the response time of the sample-based method is three times of that of ours. We also performed the same experiment on the real dataset. We found that 2,000 samples are required to guarantee a relative error of 10%, with a response time of $3s$. Our method can perform the query in only $100ms$.

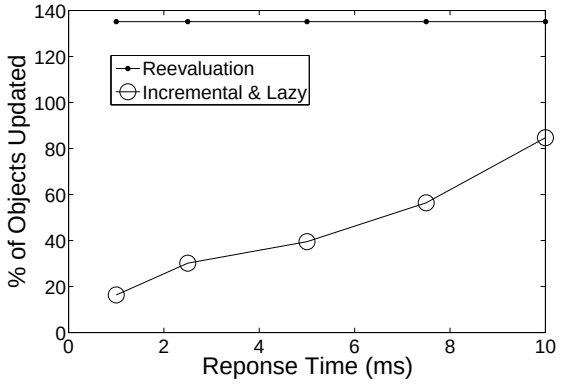

Fig. 23 Continuous C-PNN: Baseline vs. Incremental (Response).

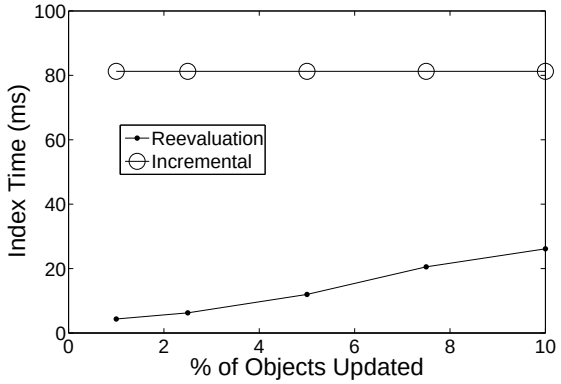

Fig. 24 Continuous C-PNN: Baseline vs. Incremental (Index).

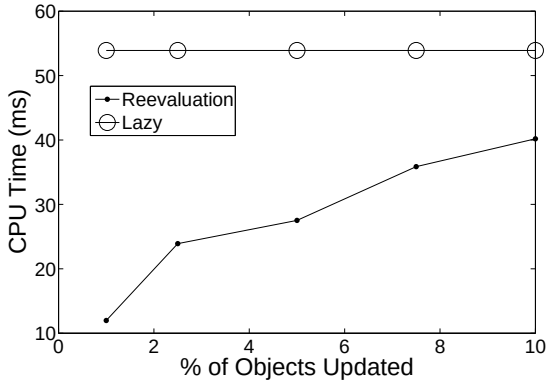

Fig. 25 Continuous C-PNN: Baseline vs. Incremental (cpu).

6.3 Results for CC-PNN

In this section, we investigate the performance of continuous C-PNN queries. For convenience, we call the naive method that re-evaluates the query whenever an update is received as the **Reevaluation** method. We name our incremental candidate set discovery method as **Incremental**, and the lazy update approach as **Lazy**.

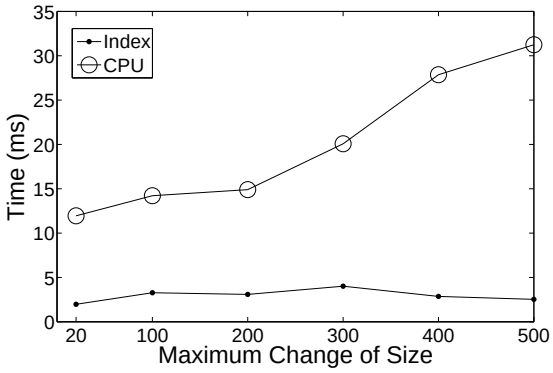

Fig. 26 Impact of Size Change.

Figure 23 compares that the overall processing time of our approach with *Re-evaluation*, over the fraction of objects updated at each time point (denoted as *frac*). Our method clearly performs better than *Re-evaluation*. For example, when *frac*=5%, an improvement of 66% is attained.

To understand why the overall processing time of our approach increases with *frac*, we have further studied the index and CPU time component, as shown respectively in Figures 24 and 25. We see that both components increase with *frac*, which contributes to the overall time increase. This time increase is expected, since the higher the value of *frac*, the more objects are updated, and so the candidate set previously computed becomes less stable, i.e., more objects in the candidate set needs to be updated. Notice that when *frac*=5%, *Incremental* addresses an improvement of 85% in index time over *Reevaluation*, while *Lazy* improves about 50% in terms of CPU time. Thus, our methods improve the performance in both index and CPU time.

Finally, we study the effect of δ, i.e., the maximum change of the uncertain region size when an object reports its update, on the performance of our approaches. The value of *frac* is set to 1%. The results are shown in Figure 26. While the index time does not change much, the CPU time increases with δ. This is because with a larger δ, the object that reports its update has a higher chance to affect the candidate set (e.g., its new uncertainty region may overlap with the regions of objects in the candidate set). Thus, the probabilities of more objects in the candidate set may be changed by the update reports, necessitating the use of more CPU time to verify or refine their probabilities.

7 Conclusions

The management of uncertain data has become an important research area in recent years. In this paper,

we examined how probabilistic nearest-neighbor queries can be efficiently evaluated on imprecise data. We proposed the C-PNN query, which uses threshold and tolerance constraints, in order to provide users with more flexibility in controlling the confidence and quality of their answers. Moreover, by evaluating C-PNN with the help of probabilistic verifiers, the problem of high costs for computing exact probabilities can be alleviated. These verifiers allow answers to be quickly determined, by using the subregions to compute the probability bounds, or by using the results of other verifiers. We also investigated different techniques of providing partial C-PNN answers to users, and developed metrics for measuring the responsiveness of these techniques. Furthermore, we developed incremental and lazy update methods for continuous C-PNNs, so that the I/O and CPU overhead is significantly reduced. In the future, we will investigate how these methods can be applied to other queries, such as k-NN and reverse-NN queries.

Acknowledgments

Reynold Cheng was supported by the Research Grants Council of Hong Kong (Projects HKU 5138/06E, HKU 513307, HKU 513508), and the Seed Funding Programme of the University of Hong Kong (grant no. 200808159002). Jinchuan Chen was supported by RGC project HKU 5138/06E. Mohamed Mobkel and Chi-Yin Chow are supported in part by the National Science Foundation under Grants IIS0811998, IIS0811935, and CNS0708604. We also thank the reviewers for their insightful comments.

References

1. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J., Hong, W.: Model-driven data acquisition in sensor networks. In: Proc. VLDB (2004)
2. Sistla, P.A., Wolfson, O., Chamberlain, S., Dao, S.: Querying the uncertain position of moving objects. In: Temporal Databases: Research and Practice (1998)
3. D.Pfoser, Jensen, C.: Capturing the uncertainty of moving-objects representations. In: Proc. SSDBM (1999)
4. Böhm, C., Pryakhin, A., Schubert, M.: The gauss-tree: Efficient object identification in databases of probabilistic feature vectors. In: Proc. ICDE (2006)
5. Cheng, R., Kalashnikov, D., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: Proc. ACM SIGMOD (2003)
6. Chen, J., Cheng, R.: Efficient evaluation of imprecise location-dependent queries. In: Proc. ICDE (2007)
7. Mokbel, M., Chow, C., Aref, W.G.: The new casper: Query processing for location services without compromising privacy. In: VLDB (2006)
8. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Querying imprecise data in moving object environments. IEEE TKDE **16**(9) (2004)
9. Kriegel, H., Kunath, P., Renz, M.: Probabilistic nearest-neighbor query on uncertain objects. In: DASFAA (2007)
10. Dyreson, C., Snodgrass, R.: Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. IEEE Trans. On Computers **51**(10) (2002)
11. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: Proc. VLDB (2004)
12. Mar, O., Sarma, A., Halevy, A., Widom, J.: ULDBs: databases with uncertainty and lineage. In: VLDB (2006)
13. Mayfield, C., Singh, S., Cheng, R., Prabhakar, S.: Orion: A database system for managing uncertain data, ver. 0.1 (http://orion.cs.purdue.edu) (2006)
14. Jampani, R., F., Wu, M., Perez, L., Jermaine, C., Haas, P.: Mcdb: a monte carlo approach to managing uncertain data. In: SIGMOD (2008)
15. Ré, C., Dalvi, N., Suciu, D.: Efficient top-k query evaluation on probabilistic data. In: Proc. ICDE (2007)
16. Yi, K., Li, F., Kollios, G., Srivastava, D.: Efficient processing of top-k queries in uncertain databases. In: Proc. ICDE (2008)
17. Soliman, M.A., Ilyas, I.F., Chang, K.C.C.: Top-k query processing in uncertain databases. In: Proc. ICDE (2007)
18. Ljosa, V., Singh, A.K.: APLA: Indexing arbitrary probability distributions. In: Proc. ICDE (2007)
19. Cheng, R., Chen, J., Mokbel, M., Chow, C.Y.: Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In: Proc. ICDE (2008)
20. Prabhakar, S., Xia, Y., Kalashnikov, D., Aref, W., Hambrusch, S.: Supporting valid-time indeterminacy. ACM Trans. Database Syst. **23**(1) (1998)
21. Beskales, G., Soliman, M., Ilyas, I.F.: Efficient search for the top-k probable nearest neighbors in uncertain databases. In: Proc. VLDB (2008)
22. Qi, Y., Singh, S., Shah, R., Prabhakar, S.: Indexing probabilistic nearest-neighbor threshold queries. In: Proc. Workshop on Management of Uncertain Data (2008)
23. Lian, X., Chen, L.: Probabilistic group nearest neighbor queries in uncertain databases. IEEE Trans. On Knowledge and Data Engineering **20**(6) (2008)
24. Cheng, R., Chen, L., Chen, J., Xie, X.: Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In: EDBT (2009)
25. Tao, Y., Cheng, R., Xiao, X., Ngai, W.K., Kao, B., Prabhakar, S.: Indexing multi-dimensional uncertain data with arbitrary probability density functions. In: Proc. VLDB (2005)
26. Xiong, X., Aref, W.: R-trees with update memos. In: Proc. ICDE (2006)
27. Cheng, R., Xia, Y., Prabhakar, S., Shah, R.: Change tolerant indexing on constantly evolving data. In: Proc. ICDE (2005)
28. Xiong, X., Mokbel, M., , Aref, W.: Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In: Proc. ICDE (2005)
29. Kalashnikov, D.V., Prabhakar, S., Hambrusch, S.E.: Main memory evaluation of monitoring queries over moving objects. Distributed and Parallel Databases **15**(2), 117–135 (2004)
30. Cheng, R., Kalashnikov, D., Prabhakar, S.: Evaluation of probabilistic queries over imprecise data in constantly-evolving environments. Information Systems (IS) **32**(1) (2007)
31. M.Hadjieleftheriou: Spatial index library version 0.44.2b URL http://u-foria.org/marioh/spatialindex/index.html