# Indexing Probabilistic Nearest-Neighbor Threshold Queries

Yinian Qi[1], Sarvjeet Singh[1], Rahul Shah[2], and Sunil Prabhakar[1]

[1] Department of Computer Science, Purdue University
{yqi, sarvjeet, sunil}@cs.purdue.edu
[2] Department of Computer Science, Louisiana State University
rahul@csc.lsu.edu

**Abstract.** Data uncertainty is inherent in many applications, including sensor networks, scientific data management, data integration, location-based applications, etc. One of common queries for uncertain data is the probabilistic nearest neighbor (PNN) query that returns all uncertain objects with non-zero probabilities to be NN. In this paper we study the PNN query with a probability threshold (PNNT), which returns all objects with the NN probability greater than the threshold. Our PNNT query removes the assumption in all previous papers that the probability of an uncertain object always adds up to 1, i.e., we consider missing probabilities. We propose an augmented R-tree index with additional probabilistic information to facilitate pruning as well as global data structures for maintaining the current pruning status. We present our algorithm for efficiently answering PNNT queries and perform experiments to show that our algorithm significantly reduces the number of objects that need to be further evaluated as NN candidates.

## 1 Introduction

The nearest neighbor (NN) query is one of the most common database queries that finds the nearest object to a query object given some distance function. Many algorithms have been proposed for NN queries [13, 3, 11] where the value of data is certain. However, uncertainty in data is inherent in many applications, such as sensor networks and location-based applications [5], where data can take different values with probabilities due to measurement errors. Take the location-based data for example. Suppose all data objects are in 2-dimensional space. The exact location of an object is unknown. However, each object is associated with a region of its possible locations and the probability density function (pdf) of the object's location within the region is known. In this probabilistic data setting, since each object has a probability (maybe 0) to be NN to a query object, we have to take probabilities into account when answering NN queries: We can either return all objects with a non-zero probability to be NN or return all objects with the NN probability greater than some threshold. We call the former *probabilistic nearest neighbor (PNN) queries* and the latter *probabilistic nearest neighbor threshold (PNNT) queries*. The formal definitions of both

queries are presented in Section 1.1. Several papers have studied the NN problem with uncertain data. For example, [5] proposed an algorithm for answering PNN queries. The algorithm returns all objects along with their non-zero NN probabilities, which requires a large number of expensive computations of the exact NN probabilities. However, most of the time we are only interested in objects with a relatively large probability to be NN, hence a probability threshold can be specified for the query to only return objects with NN probability that meets the threshold (PNNT queries). For such queries, the threshold can be leveraged to prune objects that cannot satisfy the probability requirement. [8] proposed the constrained probabilistic nearest neighbor query (C-PNN) with both threshold ($P$) and tolerance ($\Delta$) constraints, which is equivalent to our PNNT query with the threshold being $P - \Delta$.

We generalize the above NN problems for uncertain data by taking into account missing probabilities, which is not addressed in any of the previous NN papers such as [5, 8]. The missing probabilities of an object result in its absence. For the location-based data, this means that the probability of the object's location in its uncertain region may not add up to 1, indicating that there may be some probability that the object does not exist. In an uncertain database, if we consider each object to be a tuple in a relation, then the uncertain data in our NN problem has both attribute and tuple uncertainty. This is consistent with the recent uncertain model [16] where both the value of a given attribute in a tuple and the presence of the tuple itself may be uncertain. For attribute uncertainty, we assume that the uncertain attribute is associated with a pdf. For tuple uncertainty, the probability of its presence is the sum of probabilities over the pdf of all its attributes, which can be less than 1.

In this paper, we limit our discussion to NN queries with at most two uncertain attributes (i.e., 2-dimensional space) for simplicity, although our approach does not have such restriction and can be extended to higher-dimensional space. We now formally define our NN problem for uncertain data, and show why it is more complicated with missing probabilities.

### 1.1 Problem Definitions

Assuming a database of objects with uncertain attributes as continuous random variables associated with pdfs, we give two formal problem definitions for NN queries of such objects.

**Definition 1.** ***Probabilistic Nearest Neighbor (PNN) Query****: Given a query point $q$ and a set of objects with uncertain attributes and their corresponding pdfs, a PNN query returns the probability $P_{nn}(O)$ that object $O$ is NN to $q$ for each object $O$.*

For PNN queries, the probability for each object to be NN must be computed unless there is evidence that the object cannot be NN (i.e., the NN probability is 0). This implies a huge number of computations if the number of objects is huge. Moreover, the computation of the probability itself is very expensive,

which depends on many other objects whose uncertain regions overlap with its own [5]. The exact probability computation can involve integrations over multiple subregions that may have arbitrary pdfs, resulting in a high computational cost. However, objects having a small probability to be NN are generally less important than those with a high probability. For many applications, it is only necessary to retrieve objects with the NN probability exceeding a given threshold. The formal definition of such queries is given below.

**Definition 2.** ***Probabilistic Nearest Neighbor Threshold (PNNT) Query****: Given a query point $q$, a threshold $\tau$ and a set of objects with uncertain attributes and their pdfs, the PNNT query returns every object $O$ with $P_{nn}(O) > \tau$.*

Since we are only concerned about object $O$ with $P_{nn}(O) > \tau$ in PNNT queries, we do not need to compute the exact $P_{nn}$ of the object if we can prove the probability cannot exceed $\tau$. In this case, we can safely prune away those objects, hence reduce the computational cost.

Note that in our PNNT queries, we do not require that the probabilities of an object's region sum up to 1 (in other words, the pdf can be a partial pdf). Suppose the sum is $p$, then $1 - p$ is the missing probability that the object does not exist at all. This is a more general case. We need to consider more when pruning objects: Unless at least one object closer to $q$ is sure to exist, an object that is far from $q$ still has a non-zero probability to be NN, thus cannot be pruned away immediately as in [8].

The main contributions of our paper consist of the following: We proposed an R-tree based index structure that can efficiently answer PNNT queries, which is a normal R-tree index augmented with additional probability information so that we can prune objects away without sacrificing the correctness of the results. Furthermore, we designed global data structures to maintain and update the current pruning status as we retrieve nodes from the R-tree. Our PNNT query processing algorithm overcomes the limitation of previous NN papers with regard to missing probabilities and is proven to be efficient by our experiments.

The rest of the paper is organized as follows: Section 2 introduces probabilistic information that will be added to the R-tree index. Section 3 presents the algorithms used in PNNT query processing, followed by the experimental results in Section 4. Section 5 reviews the related work. Finally, we conclude our paper and point out future work in Section 6.

## 2 Augmented R-tree Index

In this section, we describe our new R-tree index for the PNNT problem defined in the previous section. We propose three types of augmentation to the normal R-tree in order to answer the PNNT queries both effectively and efficiently. The following information is added to the entries in an R-tree to facilitate query processing: Absence probability ($AP$), maximal probability ($MP$) and the absence probability bounds ($AP$-bounds). We first introduce each augmentation separately, then show how to incorporate all of them into our index structure. In the rest of the paper, we use $\tau$ to denote the PNNT query threshold.

### 2.1 Absence Probability ($AP$)

**Definition 3.** ***Pruning Circle****: A circle $C_{q,r}$ centered at query point $q$ with a radius $r$ is called a* pruning circle *if for every object $O$ lying outside $C_{q,r}$ we have $P_{nn}(O) < \tau$.*

The reason why $C_{q,r}$ is called a pruning circle is that given $C_{q,r}$, we can safely prune away all objects lying outside it when processing PNNT queries. Our goal is to shrink the pruning circle as much as possible so that all objects outside it can be pruned away immediately, leaving only a small portion of objects to be further examined as NN candidates. Next we introduce absence probability for our augmented R-tree index:

**Definition 4.** ***Absence Probability*** *AP: Given a Minimum Bounding Rectangle (MBR) $M$ in an R-tree, $AP(M)$ is defined as the probability that none of the objects contained in $M$ is present. Likewise, for a circle $C$, $AP(C)$ is the probability that no object in $C$ is present.*

Moreover, we define *maximum distance* $d_{max}(q, M)$ from query point $q$ to MBR $M$ to be the maximum distance of all distances from $q$ to $M$ and similarly *minimum distance* $d_{min}(q, M)$ is the minimum distance of all distances from $q$ to $M$. We propose the following theorem that leverages $AP(M)$ and $d_{max}(q, M)$ to prune away MBRs whose objects cannot be NN candidates.

**Theorem 1.** *If $AP(M_i) < \tau$ for MBR $M_i$, then a circle $C_{q,r}$ centered at query point $q$ with radius $r = d_{max}(q, M_i)$ is a pruning circle.*

*Proof.* Since there may be objects inside $C_{q,r}$ that are contained in MBRs other than $M_i$ (denoted as $M_j$, as shown in Fig. 1), we can infer that

$$AP(C_{q,r}) \leq AP(M_i) \cdot \left( \prod_{M_j, j \neq i} AP(M_j) \right) \leq AP(M_i) < \tau$$

For any object $O$ in any MBR $M_k$ outside $C_{q,r}$ ($d_{min}(q, M_k) \geq r$) to be NN to $q$, there should be no object inside $C_{q,r}$, i.e., $P_{nn}(O) \leq AP(C_{q,r}) < \tau$. From Definition 3 we conclude that $C_{q,r}$ is a pruning circle.

Fig. 1 illustrates the pruning circle $C_{q,r}$ when $AP(M_i) < \tau$. The MBR $M_k$ outside the circle thus can be pruned away immediately. This pruning strategy with respect to $AP$ will be referenced later as the ***first-level pruning***. We will see in Section 2.3 a variation of it that is finer-grained.

### 2.2 Maximal Probability ($MP$)

**Definition 5.** ***Maximal Probability*** *$MP(M)$ for MBR $M$ is defined as $\max_{O \in M} p_o$, where $O$ is an object contained in $M$ and $p_o$ is the probability that $O$ is present.*

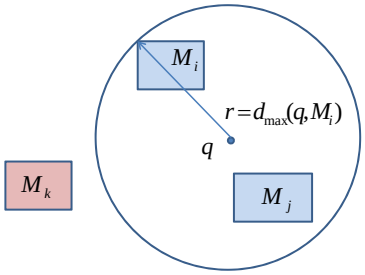


**Fig. 1.** Pruning Circle $C_{q,r}$ $(AP(M_i) < \tau)$

The maximal probability $MP$ is introduced for two purposes. Firstly, it can be used for ***pre-pruning*** to prune away MBRs with $MP < \tau$. Consider an MBR $M$ with $MP(M) < \tau$. By definitions of $MP$ and $p_o$, we know that $p_o \leq MP(M) < \tau$. Since the probability for $O$ to be NN to $q$ is at most the probability of its presence, we have $P_{nn}(O) \leq p_o < \tau$ for any object $O$ in $M$. Hence we can safely prune away the entire $M$. Secondly, $MP(M)$ can also be used for further pruning beyond the capability of the first-level pruning, which we call the ***second-level pruning***, which is supported by the theorem below:

**Theorem 2.** *Let $M_i$ be an MBR within a circle $C$. Let $M_k$ be an MBR outside $C$. If $MP(M_k) \cdot AP(M_i) < \tau$, then for any object $O$ in $M_k$, $P_{nn}(O) < \tau$.*

*Proof.* For any object $O$ in $M_k$, we have $P_{nn}(O) \leq p_o \cdot AP(C) \leq MP(M_k) \cdot AP(M_i) < \tau$, where $p_o$ is the probability that $O$ is present.

We have proved above the probability that any object $O$ in $M_k$ is NN to $q$ is less than $\tau$, so we can safely prune away the entire MBR $M_k$. This is called *second-level pruning*. In Fig. 1, if $AP(M_i) \geq \tau$ instead, we cannot use the first-level pruning to prune away $M_k$. However, if $MP(M_k) \cdot AP(M_i) < \tau$ holds, we can use the second-level pruning to prune $M_k$ away.

### 2.3 *AP*-Bounds

Unlike AP introduced in Section 2.1 that stores the absence probability of an entire MBR, $AP$-bounds store the absence probabilities of regions in the MBR specified by the bounds. The goal of $AP$-bounds is to shrink the size of the pruning circle as much as possible so that more MBRs outside the circle can be pruned away. This is a fine-grained version of the first-level pruning in Section 2.1. Both methods require that we have a pruning circle in which the absence probability is below $\tau$.

The idea of probability bounds (e.g. x-bounds) is first proposed in [6] for range queries with probability thresholds. In our paper, however, we use probability bounds for PNNT queries.

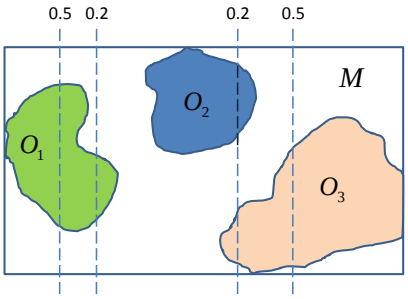


**Fig. 2.** *AP*-bounds in MBR $M$

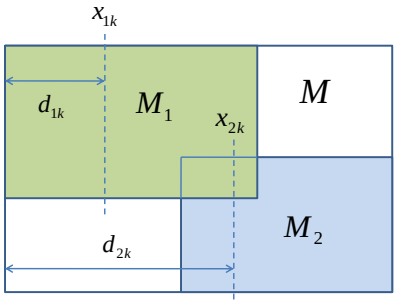


**Fig. 3.** Construct left-*AP*-bounds of $M$

**Definition 6.** ***AP-bounds*** *$AP^l_M(x)$ (left AP-bound) and $AP^r_M(x)$ (right AP-bound) for MBR $M$ are defined as a pair of lines intersecting with $M$ such that the absence probability of the region to the left of $AP^l_M(x)$ and to the right of $AP^r_M(x)$ is no greater than the bounding probability $x$ ($0 \leq x \leq 1$).*

Furthermore, we define *AP-distances* $d^l_M(x)$ and $d^r_M(x)$ to be distances from the left and right edges of MBR $M$ to $AP^l_M(x)$ and $AP^r_M(x)$ respectively. We require that *AP*-bounds be tight — they are pushed towards the left or right edges of the MBR as much as possible while still satisfying Definition 6. This ensures that *AP*-bounds are unique. *AP*-bounds can be represented using *AP*-distances and the bounding probability $x$. For example, $AP^l_M(x)$ is represented using distance $d^l_M(x)$ and $x$ itself. Suppose $O_1$, $O_2$, and $O_3$ are three objects in MBR $M$, as shown in Fig. 2. Let the bounding probability be $x$, then the *AP*-bounds of $M$ ensures that the probability that none of the three objects is present within the *AP*-bounds is no more than $x$. Note that the bounding probability becomes larger as *AP*-bounds are pushed towards the edges, i.e., $0.5 > 0.2$ in Fig. 2.

**Pruning with *AP*-bounds**: We first find the set of *AP*-bounds with bounding probability $x < \tau$ and as close to the edges of the MBR as possible. We let the new radius of the pruning circle be the minimal distance from the query point $q$ to the *AP*-bound. Then all MBRs outside of this circle can be pruned away. Pruning using *AP*-bounds instead of *AP* of an entire MBR has the advantage that the resulting pruning radius is smaller, indicating that more MBRs are likely to be outside of the pruning circle and thus can be discarded immediately without further evaluation.

### 2.4 The Index Structure

Now that we have introduced all three kinds of information that we want to leverage in PNNT query processing, we redesign the R-tree index structure by adding all the information to the entries of the R-tree internal nodes. The construction of the augmented R-tree index is also discussed in details.

There are multiple entries in an R-tree internal node, each of which has an MBR ($M$) and a pointer ($p$) to a child node that stores information about all smaller MBRs contained in $M$. We augment R-tree by adding the following additional items to each entry of an internal node: i) $AP$ ii) $MP$ iii) left $AP$ bounds and right $AP$ bounds. Note that we keep a set of left and right $AP$ bounds with different bounding probabilities $x$ to suit queries of various thresholds. The list of $x$'s is stored globally, each of which corresponds to a left and right pair of $AP$ bounds in the MBR entry.

When constructing the new index, we propagate the additional information in MBR entries in a bottom-up fashion: The $AP$ of an MBR at a higher level of the R-tree can be obtained by simply multiplying $AP$s of all its child MBRs. Let $M_1, M_2, ..., M_m$ be the child MBRs of $M$. Then $AP(M) = \prod_{k=1}^{m} AP(M_k)$. In contrast, the $MP$ of MBR $M$ is obtained by finding the maximum $MP$ among its child MBRs, i.e., $MP(M) = \max_{k=1}^{m} MP(M_k)$. To compute the left-$AP$-bounds $AP^l_M(x)$ of $M$, we compute the $AP$ distance $d^l_M(x) = \max_{k=1}^{m} d^l_{M_k}(x)$ for each bounding probability $x$. The right-$AP$-bounds are computed in the same way. We call this method "Coarse Estimation Method" (CEM). Alternatively, we have "Fine Estimation Method" (FEM), which leverages the *AP hop function* to obtain a much finer estimation of $AP$-bounds. We compute $AP$ hop functions for all of $M$'s child MBRs and deduce the hop function of $M$ from them.

**Definition 7.** ***AP hop function*** *is a function from AP-distance $d$ to bounding probability $x$, denoted as $x = h(d)$. A hop function is with regard to an MBR $M$ if $d$ is the distance from AP-bounds to $M$'s bounds.*

Note that for both left and right $AP$-bounds, we have a corresponding hop function. Suppose $M_1$ and $M_2$ are two MBRs contained in $M$, as shown in Fig. 3. $x_{11} \cdots x_{1m}$ are the bounding probabilities of left $AP$-bounds ($AP^l_{M_1}$) of $M_1$. Likewise, $x_{21} \cdots x_{2m}$ are the bounding probabilities of left $AP$-bounds ($AP^l_{M_2}$) of $M_2$. Let $h_1$ be the hop function of $M_1$ and $h_2$ for $M_2$. Let $(d_{jk}, x_{jk})$ be the points on $h_j$, where $j \in \{1, 2\}$, $1 \leq k \leq m$, and $d_{jk}$ is the distance from $M$'s left edge to $M_j$'s $AP$-bound $AP^l_{M_j}(x_{jk})$. Moreover, the $AP$-bounds for both MBRs are ordered such that $d_{jk} < d_{jk+1} (d_{jm+1} = +\infty, d_{j0} = 0)$. Then we write function $h_j$ as follows:

$$h_j(d) = x_{jk}, \quad \text{if} \quad d_{jk} \leq d < d_{jk+1} \tag{1}$$

Our goal is to compute $M$'s hop function $h$ from $h_1$ and $h_2$. The absence probability of the region within the $AP$-bound $AP^l_M(x)$ with $AP$-distance at least $\max(d_{1k_1}, d_{2k_2})$ is at most the product of absence probabilities within $AP$-bounds $AP^l_{M_1}(x_{1k_1})$ and $AP^l_{M_2}(x_{2k_2})$, that is, $AP^l_M$'s bounding probability $x \leq x_{1k_1} \cdot x_{2k_2}$.

Having observed this property, we can obtain $h$ from $h_1$ and $h_2$ as follows ($1 \leq k_1, k_2 \leq m$):

$$h(d) = x_{1k_1-1} \cdot x_{2k_2-1} \quad \text{if} \quad d \in [d_{1k_1-1}, d_{1k_1}) \text{ and } d \in [d_{2k_2-1}, d_{2k_2}) \tag{2}$$

Note that more than $m$ AP-bounds for $M$ can be computed from Equation 2. However, to be consistent with $M_1$ and $M_2$, we need to normalize function $h$ so that it has only $m$ $AP$-bounds. This can be done in a number of ways. One naïve solution is to keep the first $m$ bounds and throw the others away. With the help of hop functions, we get tighter $AP$-bounds and thus more MBRs could be pruned away using first-level pruning.

## 3 PNNT Query Processing

Before presenting our PNNT query processing algorithm, we first introduce the Global $AP$ ($GAP$) function that is essential for pruning.

### 3.1 *GAP* Function

$GAP$ function maintains the global $AP$ information for the query point $q$. Let the distance to $q$ be $d$. The definition of $GAP$ function is as follows:

**Definition 8.** *$GAP$ **function** $GAP(d)$ is the probability that no object exists inside the circle $C_{q,d}$.*

$GAP$ is used to find and shrink the pruning circle as much as possible so that all MBRs outside of the circle can be pruned away. The radius $R$ of the current pruning circle is maintained globally and decreases as more MBRs are seen during the query processing. $GAP$ is updated whenever a new MBR is retrieved, whose absence probability contributes to $GAP$ to make it more accurate. The algorithm *updateGAP* has the details. We use $M$ to denote an MBR and $M.AP(q.threshold)$ to denote the $AP$-bound of $M$ with bounding probability no greater than the query threshold. For each point on $GAP$ function, we use $d$ to denote the distance to query $q$ and $ap$ to denote $GAP(d)$, the absence probability of the circle $C_{q,d}$.

### 3.2 PNNT Query Processing Algorithm

Using the augmented R-tree index and the $GAP$ function along with the pruning techniques we have introduced so far, we give the PNNT query processing algorithm here. The algorithm has two stages: Pruning stage and refining stage. In the pruning stage, the algorithm prunes away nodes in the augmented R-tree with the help of the $GAP$ function. The goal of pruning is to dynamically update $GAP$ as we see more MBRs so that we can shrink the pruning circle accordingly. The input of this stage is an augmented R-tree built upon all data items as well as the query point itself. The output of this stage is a list of data items that are NN candidates (obtained by applying pruning techniques) as well as non-candidates that overlap the final pruning circle. The reason such non-candidates are also returned is that they are useful in the refining stage for computing the exact NN probability of the candidates — when computing the NN probability for a data

**Algorithm 1** Update $GAP$

**Require:** The current $GAP$, the query point ($q$), the newly-seen MBR ($M$)
**Ensure:** The updated $GAP$
  **if** $M.ap < q.threshold$ **then**
    set $currentPoint.d$ to be the distance between $q$ and $M.AP(q.threshold)$
    **if** $currentPoint.d == d_{\max}(query, M)$ **then**
      $currentPoint.ap = M.ap$
    **else**
      $currentPoint.ap = q.threshold$
    **end if**
  **end if**  // choosing a $GAP$ point given $M$ ends here
  **if** $GAP$ is empty **then**
    add $currentPoint$ to $GAP$
  **else**
    $savedAP = currentPoint.ap$
    insert $currentPoint$ into $GAP$ according to $d$, let the point before it be $prevPoint$
    **if** $currentPoint$ is not the first point of $GAP$ **then**
      $currentPoint.ap = savedAp * prevPoint.ap$
    **end if**
    **if** there are points after $currentPoint$ in $GAP$ **then**
      set their new $ap$ to be the old $ap$ times $savedAp$
    **end if**
  **end if**
  find the first point ($boundaryPoint$) in $GAP$ with its $ap \leq q.threshold$
  set the pruning radius $R = boundaryPoint.d$
  discard all points in $GAP$ with $d > R$

**Algorithm 2** PNNT Query Processing

**Require:** The augmented R-tree ($tree$) for all data items, the query point ($query$)
**Ensure:** All data items with NN probability greater than $query.threshold$ ($results$)
  $prune(tree.root, query)$
  **for** each $node$ in non-discarded leaf-level nodes after pruning **do**
    **for** each data $item$ in $node$ **do**
      **if** $item$ is marked as '$c$' (candidate) or '$k$' (non-candidate to be kept) **then**
        add $item$ to $remains$ (non-discarded data items)
        **if** $item$ is marked as 'c' **then**
          add $item$ to $candidates$ (NN candidates)
        **end if**
      **end if**
    **end for**
  **end for**  // pruning stage ends here
  **for** each $item$ in $candidates$ **do**
    $P_{nn} = computeNNProbability(item, remains, query)$
    **if** $P_{nn} > query.threshold$ **then**
      add $item$ to $results$
    **end if**
  **end for**
  return $results$  // refining stage ends here

item, all items that are not definitely further to the query must be taken into account [5]. The refining stage then decides whether a NN candidate is indeed a query result by checking whether its exact NN probability is greater than the threshold. The PNNT query processing algorithm is shown in Algorithm 2.

The details of the pruning algorithm (i.e. `prune`) are in Algorithm 3. The input is the query point and the node in the tree where the pruning starts. Note that we update the $GAP$ function whenever we see a new MBR using algorithm `updateGAP` introduced in Section 3.1. `MarkMBRs` (Algorithm 4) marks all MBRs in the node as 'c', 'k' or 'd' according to the latest $GAP$ function, where 'c' means NN candidates, 'k' means non-candidates that we need to keep for the refining stage and 'd' means others to be discarded.

**Algorithm 3** Prune

**Require:** A node in *tree* (*node*) to start pruning, *query*
**Ensure:** All non-discarded nodes with marked MBRs
  **for** each MBR $M$ in *node* **do**
    $updateGAP(M, query)$
  **end for**
  $markMBRs(node, query)$
  **if** *node* is a leaf **then**
    return
  **end if**
  $next = pickMBRtoExplore(node, query)$
  **while** *next* != NULL **do**
    $prune(next, query)$
    $markMBRs(node, query)$
    $next = pickMBRtoExplore(node, query)$
  **end while**

The nodes in the augmented R-tree are visited in a depth-first manner. `PickMBRtoExplore` picks an MBR in the node from all that are marked 'c'. The corresponding child of the node will then be explored. The criteria for picking is to choose the MBR that is furthest from the query point, in the hope that its children will be discarded soon.

## 4 Experimental Results

We performed our experiments on 1-dimensional data represented as intervals. Each interval is the uncertain region of the data and its pdf is represented using histograms. The total probability $p$ over the interval is either in (0, 1] or in (0.5, 1], hence there is 1-$p$ probability that the interval does not exist. The threshold $\tau$ of the PNNT query is at least 0.1, assuming that the user is not interested in small probabilities. All intervals are randomly generated within the range (0, 10000] and the size of the interval is in [1, 10]. For each experiment,

**Algorithm 4** Mark MBRs

**Require:** A node in *tree* (*node*) to mark its MBRs, *query*
**Ensure:** All MBRs in *node* are marked
  **for** each MBR $M$ in *node* **do**
    **if** $M$ is outside of the current pruning circle $C_R$ centered at *query* with radius $R$ **then**
      mark $M$ as '$d$'  // first-level pruning
    **else**
      mark $M$ as '$c$'
      **if** $M.mp < query.threshold$ **then**
        mark $M$ as '$k$'  // pre-pruning
      **else**
        search in $GAP$ for the last point satisfying $GAP.d \leq d_{\min}(query, M)$
        **if** $M.mp * GAP.ap < query.threshold$ **then**
          mark $M$ as '$k$'  //second-level pruning
        **end if**
      **end if**
    **end if**
  **end for**

we average the statistics over 10 randomly generated query points in (0, 10000]. The default values/ranges for data size $n$, threshold $\tau$ and total probability $p$ is 100000 (100K), 0.3 and (0, 1]. Our algorithm can handle arbitrary pdf represented by histograms. In the experiments, we generated data with either uniform pdf (default) or Gaussian pdf. In the following experiments, we always take default values of parameters unless otherwise specified. We ran our experiments (written in C++) on a PC with Intel T2500 2.00GHz CPU and 2.00GB main memory.

Our goal is to evaluate the performance of our PNNT query processing algorithm with different parameters presented below. We compared our algorithm with the naïve algorithm that evaluates the exact NN probability for each data item and then returns all items with NN probabilities greater than $\tau$. As the data size grows, the PNNT algorithm becomes more and more superior compared with the naïve one. For 500 data items, the naïve algorithm takes over 14.5 $s$ while our algorithm takes no more than 3 $ms$, over 500 times faster. We also compared the performances of the three pruning techniques: Pre-pruning (`Prune0`), first-level pruning (`Prune1`) and second-level pruning (`Prune2`) (note that the actual algorithm combines all three together). For each point on the figures, we average its value over ten experiments with randomly generated query locations and draw the confidence interval associated with the value. The experimental results are as follows:

**Effect of Data Size**: We evaluated our algorithm by varying the data set size $n$ from 10000 to 100000. We computed the pruning percentage of our algorithm by dividing the number of NN candidates (`candidateCount`) by $n$. Note that `candidateCount` is obtained after the pruning stage of the algorithm. We compared the pruning percentage when the total probability $p \in (0, 1]$ and

$p \in (0.5, 1]$. In the former case, $p$ is just a random probability while in the latter case, $p$ is above 0.5, indicating that the probability that the data item exists is greater than the probability that it does not exist. Fig. 4 shows the result: As data size grows, the pruning percentage also grows with a relatively sharp increase when the data size turns from 10000 to 20000. Over 99.7% data items are pruned away for both cases while a random $p \in (0, 1]$ generally has a higher pruning percentage than $p \in (0.5, 1]$. This might be due to the effect of pre-pruning that prunes away all data items with $p < 0.3$ ($\tau = 0.3$), which does not work at all for $p \in (0.5, 1]$. We also evaluated the time cost of our algorithm with regard to the pruning stage and the refining stage in Fig. 5. The time cost increases as data size grows. The pruning time cost is always bigger than the refining time due to the large data size (it takes longer to prune in the R-tree) as well as the high pruning percentage of our algorithm (there are less than 200 candidates to be evaluated even when the size reaches 100000). The total time cost (pruning and refining) is also shown in Fig. 5.

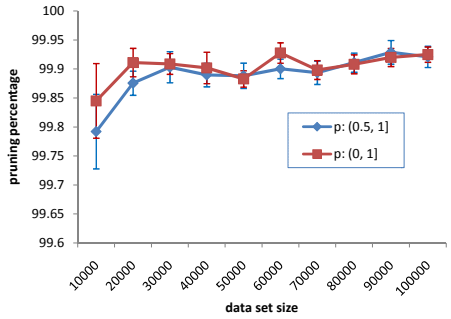


**Fig. 4.** Effect of Data Size on Pruning

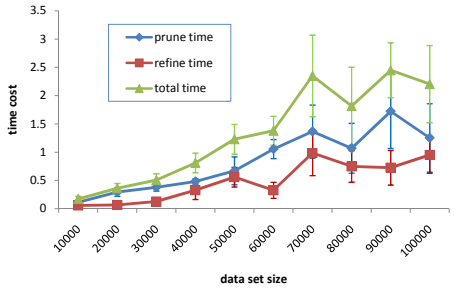


**Fig. 5.** Effect of Data Size on Time Cost

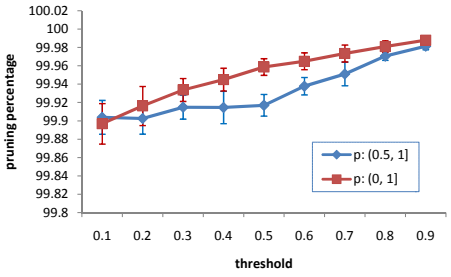


**Fig. 6.** Effect of Threshold on Pruning

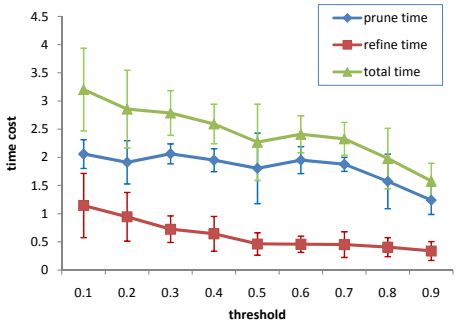


**Fig. 7.** Effect of Threshold on Time Cost

**Effect of Threshold**: We repeated the previous experiments with the data size fixed at 100000 and the threshold varying from 0.1 to 0.9 in Fig. 6 and Fig. 7.

Unlike the time cost experiment when the data size changes, the cost decreases as threshold increases. This is because we can prune much more when the threshold is big, as most of the data items are unlikely to have a NN probability above the big threshold. The pruning time is more than the refining time for the same reason as before. For the pruning percentage experiment, we observed similar results: The percentage increases as threshold becomes bigger and reaches as high as 99.98% when $\tau = 0.9$. The total probability $p \in (0, 1]$ still results in more pruning compared with $p \in (0.5, 1]$ like before. We further compared the three pruning techniques (`Prune0, Prune1, Prune2`) with the varying threshold in Fig. 8. The result shows that `Prune1` contributes the most of all three techniques with a pruning percentage around 99.8%, followed by `Prune0` and `Prune2`. As threshold increases, `Prune0` increases almost linearly when $\tau > 0.3$. `Prune1` stays relatively same with high pruning percentage while `prune2` increases slightly with fluctuations.

**Data with Gaussian PDF**: So far we have experimented with data that has a uniform pdf. We now show that our algorithm performs well for data with Gaussian pdf too. Fig. 9 shows the pruning percentage of the three pruning techniques over different thresholds. Compared with Fig. 8, we observe the similar results: `prune1` prunes most, followed by `prune0` and `prune2`. The pruning percentages of the three techniques are all above 94%.

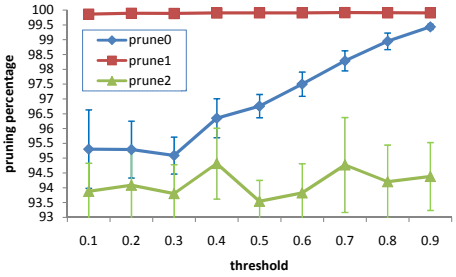


**Fig. 8.** Pruning Techniques

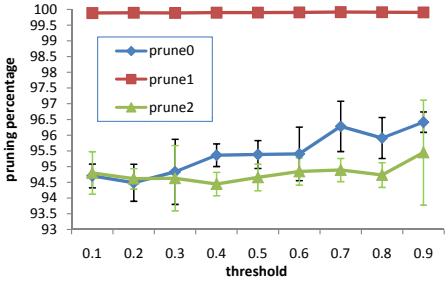


**Fig. 9.** Gaussian PDF

## 5 Related Work

Although the first probabilistic models were proposed as early as in the 1990's [1, 9], it is not until recently when many efforts have been made to design database systems that manage uncertainty [2, 4, 15]. Two major models for uncertainty exist: Tuple uncertainty models and attribute uncertainty models. For tuple uncertainty models [2, 4], each tuple is associated with a probability of its presence. For attribute uncertainty models, a tuple always exists, but there may be one or more uncertain attributes in the tuple with pdfs associated with them. Recently,

a database model is proposed for supporting pdf attributes that can handle both attribute and tuple uncertainty, which considers intra-tuple dependencies (captured by dependency sets) as well as inter-tuple dependencies (captured by history) [16].

In parallel to uncertainty modeling, much work has been done on indexing and querying uncertain data, including indexing for probabilistic threshold queries [7], indexing uncertain categorical data [14], processing range queries and PNN queries [10, 5], processing ranked queries [17, 12], etc. However, little work has been done on indexing and querying uncertain data with both attribute and tuple uncertainty which are unified in the model of [16]. In our paper, we proposed an efficient algorithm for processing PNNT queries. [5] gives an algorithm for PNN, but only prunes objects when their NN probability is 0. [8] proposed the concept of the "constrained PNN query" with a probability threshold and an error tolerance, which can be leveraged to prune objects away that cannot meet the threshold/tolerance requirement, thus saving many expensive computations of the exact NN probabilities. The authors followed the attribute uncertainty model and assumed that the probability of an object being in a closed region always adds up to 1, i.e., there is no tuple uncertainty – the object always exists. Our approach overcomes this limitation and can efficiently process PNNT queries when objects have both attribute and tuple uncertainty.

## 6 Conclusions

In this paper, we gave an efficient algorithm to process PNNT queries for uncertain data with missing probabilities, which is a problem that has not been addressed by any previous paper. Since data items that are close to the query point may not exist in this case, it is harder to prune away items that are further away (it still has a chance to be the NN). We designed an augmented R-tree index specifically for such queries. Each MBR is now associated with probability information such as $MP$, $AP$, and $AP$ bounds to facilitate pruning. We further introduced a global data structure called $GAP$ function to dynamically capture the decreasing of $AP$ within the pruning circle. With this fine-grained $AP$ information, we can apply our pruning techniques to prune away a large number of nodes in the R-tree. Our experiments proved the efficiency of the algorithm – over 90% data items can be pruned away, leaving only a small portion of data to be further examined as PNNT results.

## References


1. D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. on Knowl. and Data Eng.*, 4(5):487–502, 1992.
2. Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. Uldbs: databases with uncertainty and lineage. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 953–964, 2006.

3. Stefan Berchtold, Bernhard Ertl, Daniel A. Keim, Hans-Peter Kriegel, and Thomas Seidl. Fast nearest neighbor search in high-dimensional space. In *ICDE '98*, pages 209–218, 1998.
4. Jihad Boulos, Nilesh Dalvi, Bhushan Mandhani, Shobhit Mathur, Chris Re, and Dan Suciu. Mystiq: a system for finding more answers by using probabilities. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 891–893, 2005.
5. Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Querying imprecise data in moving object environments. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(9):1112–1127, 2004.
6. Reynold Cheng, Sarvjeet Singh, Sunil Prabhakar, Rahul Shah, Jeffrey Scott Vitter, and Yuni Xia. Efficient join processing over uncertain data. In *Proceedings of the 15th CIKM Conference*, pages 738–747, 2006.
7. Reynold Cheng, Yuni Xia, Sunil Prabhakar, Rahul Shah, and Jeffery Scott Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proceedings of the 30th VLDB Conference*, 2004.
8. Reynold Cheng, Yuni Xia, Sunil Prabhakar, Rahul Shah, and Jeffery Scott Vitter. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *Proceedings of the 24th ICDE Conference*, 2008.
9. Debabrata Dey and Sumit Sarkar. A probabilistic relational model and algebra. *ACM Trans. Database Syst.*, 21(3):339–369, 1996.
10. Dmitri V. Kalashnikov, Sunil Prabhakar, Susanne E. Hambrusch, and Walid G. Aref. Efficient evaluation of continuous range queries on moving objects. In *DEXA '02: Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 731–740, 2002.
11. Jon M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *STOC '97*, pages 599–608, 1997.
12. Xiang Lian and Lei Chen. Probabilistic ranked queries in uncertain databases. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 511–522, 2008.
13. Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD*, 1995.
14. Sarvjeet Singh, Chris Mayfield, Sunil Prabhakar, Rahul Shah, and Susanne Hambrusch. Indexing uncertain categorical data. In *Proceedings of the 23rd ICDE Conference*, 2007.
15. Sarvjeet Singh, Rahul Shah, Sunil Prabhakar, and Chris Mayfield. Unified model for uncertainty management in databases. volume 21, pages 560– 571, 2003.
16. Sarvjeet Singh, Rahul Shah, Sunil Prabhakar, and Chris Mayfield. Database support for pdf attributes. In *Proceedings of the 24th ICDE Conference*, 2008.
17. Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. Urank: formulation and efficient evaluation of top-k queries in uncertain databases. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1082–1084, 2007.