K-Nearest Neighbor Search for Fuzzy Objects

Kai Zheng[#] Pui Cheong Fung[#] Xiaofang Zhou[#*]

[#]School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane 4072, Australia

[*]NICTA Queensland Research Laboratory

{kevinz, g.fung, zxf}@itee.uq.edu.au

ABSTRACT

The K-Nearest Neighbor search (kNN) problem has been investigated extensively in the past due to its broad range of applications. In this paper we study this problem in the context of fuzzy objects that have indeterministic boundaries. Fuzzy objects play an important role in many areas, such as biomedical image databases and GIS. Existing research on fuzzy objects mainly focuses on modelling basic fuzzy object types and operations, leaving the processing of more advanced queries such as kNN query untouched. In this paper, we propose two new kinds of kNN queries for fuzzy objects, *Ad-hoc kNN query* (AKNN) and *Range kNN query* (RKNN), to find the k nearest objects qualifying at a probability threshold or within a probability range. For efficient AKNN query processing, we optimize the basic best-first search algorithm by deriving more accurate approximations for the distance function between fuzzy objects and the query object. To improve the performance of RKNN search, effective pruning rules are developed to significantly reduce the search space and further speed up the candidate refinement process. The efficiency of our proposed algorithms as well as the optimization techniques are verified with an extensive set of experiments using both synthetic and real datasets.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms

Algorithms

Keywords

nearest neighbor query, fuzzy database, probabilistic database

1. INTRODUCTION

K-Nearest Neighbor (kNN) search is one of the most important operations in spatial DBMS, due to its broad range of applications including molecular biology [4], medical imaging [13], multimedia databases [26] and so on. After an extensive study on this topic during the last decades, numerous indexing and searching strategies have been proposed [19, 11, 13, 27, 32, 33]. Recently, with the proliferation of mobile computing, kNN queries have also been extended to moving object databases [5, 30] and uncertain databases [8]. These work has collectively made significant advances in improving the efficiency of search algorithms and enriching the queries to support more complex data types. To the best of our knowledge, all these work assume the underlying data to be *crisp objects*, which means their compositions and boundaries are deterministic. However, in some real applications such as biomedical image analysis and geographical information systems, the data may not satisfy this assumption.

Microscope images are typical sources of such kind of non-crisp data. Nowadays, as the high-throughput microscopes are producing images much faster than before, the huge size of the datasets rules out the traditional approach of identifying objects and relationships manually. Instead, we must rely on automatic techniques. However, it is often impossible to interpret the objects in microscope images unequivocally due to the limitation of image resolution and interference of electronic noises. In order to reflect the uncertainty embedded in images and offer subsequent analysis more information to work with, *probabilistic mask* is produced on the extent of identified cells by probabilistic segmentation [14]. For example, Figure 1 shows a typical cell image in biomedical analysis. The boundary of the cell cannot be identified easily, i.e. it is not crisp. Under the model of probabilistic mask, different pixels in the image will be assigned different probabilities to indicate the likelihood that the pixel belongs to the cell. By this means, each object is transformed into a collection of points with probabilities. As such, uncertainty lies in their *compositions*, i.e., a point may or may not belong to the object. Therefore, they are essentially different from the uncertain databases in which the objects are assumed to have *probabilistic locations* at query time.

The concept of *probabilistic mask* essentially represents the cells in images as *fuzzy objects*. Although fuzzy objects have long been studied in GIS community [25, 28, 2, 22], common spatial queries such as range and kNN queries still remain uninvestigated at large. In this paper we will address the problem of searching the k nearest objects in Euclidean space over large fuzzy dataset. This type of query has many applications in the biomedical field such as brain aging study [21], Alzheimer's disease analysis [17] and so on.

Before stepping further to propose our own model, we have to raise the question: does the fuzziness of objects change the nature of traditional kNN problem? Or can we adopt existing solutions

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

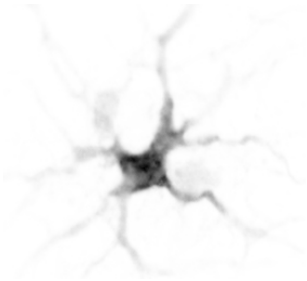


Figure 1: A typical cell image in biomedical analysis. Darker pixels have higher probability of belonging to the cell.

to support this type of data? The most common type of nearest neighbor query is the point-kNN query that finds the k points from a dataset which are closest to a query point. If we want to plug the fuzzy type into existing point-kNN algorithms, an object should be represented by single point. Then a problem arises: how to find *this* point? Usually it is not easy to choose a suitable representative point because the underlying object (e.g. neuron cells) has complex shape, and more importantly, not all parts of the object are equally important (kernel vs. boundary) in terms of their confidence. Hence choosing arbitrary points will cause considerable information loss and even produce misleading results.

To address this problem, we identify the key issue as how to define a meaningful query in which the fuzzy information is taken into account along with spatial proximity. In this work, instead of mixing the probability and distance into some unified similarity function, we offer the users freedom to choose the confidence level on which the kNN set is required. Specifically, we introduce *probability threshold* as a user-defined parameter in the kNN query. Once the threshold is given, we can know which parts of objects should be counted in and then distances between objects and queries are also adjusted accordingly. Users can benefit from this kind of query by tuning the probability threshold to compare outcomes on different fuzzy levels. Considering the biomedical image analysis again as an example, if the nearest cells are needed merely based on the clearest region (e.g. kernel), one can specify a high threshold. On the other hand, if he/she wants to perform the search by considering fuzzier region, a query with low threshold can be issued. The major difference of our new queries with the traditional kNN queries is that the fraction of objects which will be taken into consideration is unknown until the query is given. Consequently, while this novel query offers more flexibility to users, it also brings difficulties to our indexing and search algorithms, which will be investigated and overcome in the rest of the paper.

To sum up, we make the following major contributions:

- We define a distance function, α-distance, which measures the spatial proximity between two fuzzy objects. The key difference from other distance functions like Lp-norms is that it takes a probability value, α, as the parameter so that users can control the confidence level based on which the distance is evaluated.

- Two new kinds of kNN queries are presented, which returns the k nearest fuzzy objects, at single probability threshold and a range of probability thresholds, respectively.

- We propose efficient algorithm to answer AKNN query by utilizing R-tree index. Tightness of lower and upper bounds of distance are further improved by taking advantage of the shrinking property of fuzzy objects and monotonicity of α-distance.

- Effective heuristic rules, including pruning most disqualifying objects and accelerating candidate refinement, are developed to improve the efficiency of RKNN query processing.

- Extensive experiments are conducted on both synthetic and real datasets to verify that the proposed algorithms as well as their optimization mechanisms achieve satisfactory performance.

The remainder of this paper is organized as follows. In Section 2, the fuzzy object model and definitions of kNN queries are given. Proposed algorithms for answering AKNN and RKNN query are presented in Section 3 and 4, followed by a brief analysis to estimate the average number of object access in Section 5. We show the experiment results in Section 6 and discuss the related work in Section 7. Finally we conclude the paper in Section 8.

2. MODELS AND QUERIES

In this section, we firstly introduce a fuzzy object model based on fuzzy set theory. Then we define a distance function to measure spatial closeness between two fuzzy objects in Euclidean space. Finally, two new kNN queries are proposed in order to meet different user purposes. Table 1 summarizes the notations we use through out the paper.

Notation	Definition
$\mathcal{D}$	fuzzy object dataset
α	probability threshold
A_s	the support set of fuzzy object A
A_k	the kernel set of fuzzy object A
A_α	the α-cut of fuzzy object A
$\|\|a-b\|\|$	Euclidean distance between point a and b
$d_\alpha(A,B)$	α-distance between A and B
$d_\alpha^{+(-)}(A,B)$	upper (lower) bound of $d_\alpha(A,B)$
M_A	the MBR of A_s
$M_A(\alpha)$	the MBR of A_α
$M_A(\alpha)^*$	the approximated MBR of A_α
U_A	the distinct membership value set of A
$\Omega(A)$	the critical probability set of A

Table 1: Summarization of notations

2.1 Fuzzy Object Model

Fuzzy objects are usually modeled by fuzzy sets [34], which are characterized by their membership function $\mu : \mathbb{R}^d \rightarrow [0,1]$, mapping any element in the object space to a real value $\mu(x)$ within the interval $[0,1]$. An element mapped to zero means that the member is not included in the fuzzy set, while one describes a fully included member. Values strictly in between characterize the fuzzy members. Fuzzy objects can be defined in continuous space if a continuous membership function can be given. But in real applications, such a membership function is often not explicitly available due to the diversity of fuzzy objects. Besides, fuzzy objects identified from raster images are normally represented by pixels which are actually discrete points. For these reasons, we adopt a very general discrete form to model fuzzy objects.

DEFINITION 1. *A* **fuzzy object** *in d-dimensional space is represented by a set of probabilistic spatial point*

$$A = \{\langle a, \mu_A(a)\rangle | \mu_A(a) > 0\}$$

where a is a d-dimensional point and $\mu_A(a)$ is the membership value of a which indicates the probability of a belonging to A.

Similar as the fuzzy sets, we can also define the following terms for convenience of use.

DEFINITION 2. *Given a fuzzy object A, the set $A_s = \{a \in A | \mu_A(a) > 0\}$, $A_k = \{a \in A | \mu_A(a) = 1\}$ and $A_\alpha = \{a \in A | \mu_A(a) \geq \alpha\}$ is called the* **support** *set, the* **kernel** *set and the* α**-cut** *of A, respectively.*

In this paper, we assume the kernel of any fuzzy object is not empty. This is reasonable since that once an object can be identified, there must be some certain information belonging to it; otherwise the object itself may not exist.

In order to perform the nearest neighbor search, we need a scoring function to measure spatial closeness between two fuzzy objects in space. There exists some work [6, 7] in mathematics and image processing field which proposed to measure the distance between fuzzy sets. The basic idea is to calculate the distance for each α-cut and then do integration over the entire interval [0,1]. The final score calculated in this way is actually the expected distance by treating the probability as the weight of each distance. Based on this definition, a fuzzy object with low probability region may never be regarded as the nearest neighbor even it is very close to the query object in terms of spatial locations. In other words, users cannot explore the different possibilities of outcomes inherently existing in reality, because the information with low probability can be easily dominated and ignored. In the light of this observation, we in this paper define the distance between fuzzy objects as follows.

DEFINITION 3. *For two fuzzy objects A and B, their* α**-distance** *is given by the function:*

$$d_\alpha(A, B) = \min_{\langle a,b\rangle \in A_\alpha \times B_\alpha} ||a - b||$$

where α is a user-specified probability threshold.

The merit of this distance definition is that we do not mix probabilities into the final score computation. On the contrary, we leave it as a parameter for users to set. Once the probability threshold is given, objects are defuzzied to their α-cuts. Then we adopt the minimum distance as their distance measurement. Minimum distance is commonly used in the cases that an object cannot be abstracted by single point due to its relatively large size. For example, when we say a coffee shop is beside (near to) a football stadium, their minimum distance is what we really refer to. From the computational aspect, calculating minimum distance is equivalent to finding the *closest pair (CP)* between two point sets, which is well studied in both computational geometry [18] and database area [9].

The advantage of this definition is that users can explore the possible distances between two fuzzy objects by setting different probability thresholds. It is easy to verify that the α-distance is a monotonically non-decreasing function of α, i.e., $\forall \alpha_1 < \alpha_2$, $d_{\alpha_1}(A, B) \leq d_{\alpha_2}(A, B)$. Figure 2 examplifies the α-cuts of two fuzzy objects and their corresponding α-distances. We would like to clarify that there is no assumption for the probability distribution of fuzzy objects in this paper. The monotonicity of the α-distance directly comes from the *shrinking property* of α-cut.

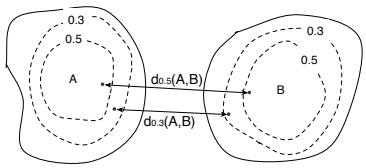


Figure 2: α-distance of fuzzy objects

2.2 KNN Queries

In this subsection, we propose two new types of kNN queries for fuzzy objects, i.e., ad-hoc kNN query and range kNN query.

DEFINITION 4. *Given a fuzzy object dataset $\mathcal{D}$, natural number k, probability threshold α and query (fuzzy) object Q,* **ad-hoc kNN (AKNN) query** *retrieves k objects from $\mathcal{D}$ which have smallest α-distances with respect to Q.*

The intuition behind this query definition is to let users choose the confidence level of information in the objects based on which the nearest neighbors will be derived. Furthermore, one can even try different probability thresholds to examine the differences of the results, to make sure no valuable information is discarded. Motivated by this application, we generalize AKNN query by replacing single probability threshold with a range of probability thresholds, resulting in our second query definition.

DEFINITION 5. *Given a fuzzy object dataset $\mathcal{D}$, a natural number k, a probability range $I = [\alpha_s, \alpha_e]$ and query (fuzzy) object Q,* **range kNN (RKNN) query** *returns a set of objects $\{\langle A, I_A\rangle | I_A \in [\alpha_s, \alpha_e]\}$, where A belongs to the k nearest neighbor set at any probability $\alpha \in I_A$. I_A is called the qualifying range of A.*

RKNN query is more powerful (yet more computationally challenging) since it allows the user to specify a range of probability thresholds and returns all the possible nearest neighbors along with their qualifying ranges. In the above query definitions, the order of k nearest neighbors is not important (order insensitive).

Consider Figure 3 where the α-distances of four objects with respect to the query Q are shown. If we set probability threshold to 0.4, ad-hoc 2-NN query will return the set $\{A, B\}$ as the result; but the set $\{A, C\}$ turns to be result if α is changed to 0.5. Furthermore, if the user issues a RKNN query with $k = 2$ and $I = [0.3, 0.6]$, the result set should be $\{\langle A, [0.3, 0.6]\rangle, \langle B, [0.3, 0.45] \cup (0.55, 0.6]\rangle, \langle C, (0.45, 0.55]\rangle\}$.

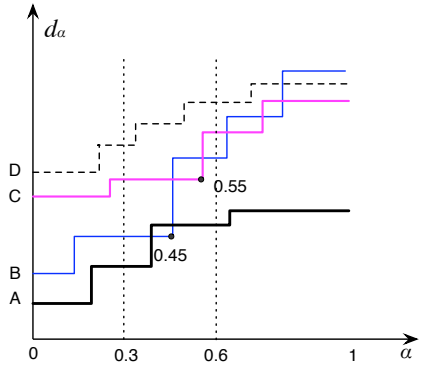


Figure 3: KNN queries

The new kNN queries for fuzzy objects are quite different from the traditional ones such as point-NN queries or continuous kNN queries, in the sense that each object may be fully, partially or not involved at all in the query evaluation, depending on the given probability threshold. In other words, objects themselves are not determined until the query is issued, which poses great challenges on the efficiency of search algorithms. In the following two sections, we will try to overcome these difficulties and design efficient solutions to answer these queries.

3. AKNN QUERY PROCESSING

3.1 Basic AKNN Search

The most straightforward approach for answering AKNN query is to linearly scan the whole dataset. For each fuzzy object, we calculate its α-distance with the query object and keep the top-k results in a heap. However, this method could be both CPU and IO intensive. First, the evaluation of α-distance is quadratic with the number of points in fuzzy objects. In addition, the whole dataset usually cannot fit into the memory due to its high space cost. Therefore, to improve the efficiency, we employ the branch-and-bound technique to prune the search space with the help of R-tree index. But we make some modifications to suit the characteristic of fuzzy objects. In particular, each leaf node of the R-tree corresponds to a fuzzy object. Instead of storing all the points, we just keep the MBR of its fuzzy region in the main memory along with a pointer which refers to the actual location on hard disk. Then the R-tree is traversed in best-first paradigm using the following principles: a) when an intermediate entry is encountered, we visit its subtree only if it may contain any qualifying objects; b) for a leaf node, we retrieve its corresponding fuzzy object from external storage only if it may belong to the k nearest neighbors.

Algorithm 1 sketches the main structure of this process. Similar to the original BFS method, we maintain a priority queue which stores intermediate or leaf index entries. In the traversal process, the algorithm will encounter three kinds of elements popped from the priority queue, which must be treated differently.

- If the popped element is an intermediate node, we insert each child node back to the queue together with its minimum distance to $M_Q(\alpha)$, i.e., the MBR of Q_α.
- The second possibility is that we encounter an object which has been retrieved from hard disk. Because the associated value is already the actual α-distance to the query instead of a lower bound, this object is guaranteed to be closer than any other object left in the queue. So we add it to the result set.
- If a leaf node is encountered, we retrieve the corresponding object from hard disk, evaluate its α-distance with Q and insert it into the queue.

In the above process, we use the minimum distance between the MBRs of fuzzy objects to serve as a simple lower bound for their α-distance. Compared to the α-distance, the evaluation of minimum distance is much more efficient and does not require the retrieval of the object. In d-dimensional space, we denote a MBR by M = (M^{1+}, M^{1-}, ..., M^{d+}, M^{d-}), where M^{i+} (M^{i-}) is the upper (lower) bound of its i-th dimension. The minimum distance between M_A and M_B can be evaluated by:

$$MinDist(M_A, M_B) = \sqrt{\sum_{i=1}^{d} l_i^2} \quad (1)$$

Algorithm 1: Basic AKNN Search

```
Input: root, Q, α, k
Output: NN: the result set
initialize priority queue H;
enqueue ⟨root, MinDist(M_Q(α), M_root)⟩ into H;
while |NN| < k and H is not empty do
    E ← dequeue H;
    if E is an intermediate entry then
        for each child entry V of E do
            enqueue ⟨V, MinDist(M_Q(α), M_V)⟩;
    else if E is an object then
        add E to NN;
    else
        probe E and enqueue ⟨E, d_α(E, Q)⟩;
return NN;
```

where

$$l_i = \begin{cases} M_A^{i-} - M_B^{i+}, & \text{if } M_A^{i-} > M_B^{i+} \\ M_B^{i-} - M_A^{i+}, & \text{if } M_B^{i-} > M_A^{i+} \\ 0, & \text{otherwise} \end{cases}$$

3.2 Improving the Lower Bound

The AKNN search algorithm adopts $MinDist$ between the MBRs of fuzzy objects as the lower bound of their α-distance. Though computationally simple, this lower bound is relatively loose, especially in the cases that α is set high. This is due to the fact that the α-cut of a fuzzy object *shrinks* as α increases. Accordingly, the size of MBR of its α-cut reduces as well. However, we approximate the object by the MBR of its support all the time, making the lower bound much smaller than the actual distance. In order to improve it, the key is to use more accurate MBR to represent the α-cut whenever α changes. Of course it is not realistic to pre-compute the MBR for every possible α-cut since it is extremely space costly. Is there any way to capture the shrinking trend of fuzzy objects, in the mean time cost little extra space?

Given a fuzzy object A in d-dimensional space, we denote the MBR of its α-cut by

$$M_A(\alpha) = (M_A^{1+}(\alpha), M_A^{1-}(\alpha), \ldots, M_A^{d+}(\alpha), M_A^{d-}(\alpha))$$

where $M_A^{i+}(\alpha)$ ($M_A^{i-}(\alpha)$) represents the upper (lower) bound of the i-th dimension of A_α. Without loss of generality, we just use the $M_A^{i+}(\alpha)$ with arbitrary i to illustrate our idea, since the same techniques apply for other dimensions as well as the lower bounds.

By the definition of fuzzy object, A_α will gradually *shrink* from the support A_s to the kernel A_k as α increases from 0 to 1. In the mean time, $M_A^{i+}(\alpha)$ will also approach $M_A^{i+}(1)$. Let $\delta(\alpha)$ denote the difference between $M_A^{i+}(\alpha)$ and $M_A^{i+}(1)$, i.e., $\delta(\alpha) = |M_A^{i+}(\alpha) - M_A^{i+}(1)|$, then we can obtain the *boundary function (bf)* for M_A^{i+} by calculating all pairs $\langle \alpha, \delta(\alpha) \rangle$ for each $\alpha \in U_A$, i.e.,

$$bf = \{\langle \alpha, \delta(\alpha) \rangle | \alpha \in U_A\}$$

where U_A is the set of all distinct membership values of A, i.e., $U_A = \{r \in (0, 1] | \exists a \in A, \mu_A(a) = r\}$. Naturally, from the shrinking property of α-cut, we have $\forall \alpha_i < \alpha_j$, $\delta(\alpha_i) \geq \delta(\alpha_j)$. If we treat bf as a series of 2d points, it should be plotted as a decreasing curve, as shown by the dashed line in Figure 4. We would like to approximate this bf so that with any give α, we can have a better estimation for the A_α. Ideally, bf can be approximated

by arbitrary function. But computing and storing a linear function need considerably less overhead than a higher order function. For this reason, we focus on using a linear function to approximate bf and leave other functions as future work.

There are many applications where it is necessary to determine a classical regression line without any constraint conditions. A conventional regression line is to find the parameters (m, t) of the linear function $y = m \cdot x + t$ which minimizes the least square error. This line, however, is not a *conservative approximation* of the bf and hence cannot satisfy the lower bounding property for the MBR. In order to guarantee no false dismissals, we need to find a line which minimizes the above condition while meeting the constraint that the estimated y values are more than or equal to the actual δ values, i.e., $(m \cdot \alpha + t) \geq \delta(\alpha)$. We formally define this line as follows.

DEFINITION 6. *The* **optimal conservative approximation** *of the boundary function is a line*

$$L_{opt} : y = m_{opt} \cdot x + t_{opt}$$

with the following constraints:

1. *L_{opt} is a conservative approximation of δ, i.e.,*

$$\forall \alpha \in U_A, \delta(\alpha) \leq m_{opt} \cdot \alpha + t_{opt}$$

2. *L_{opt} minimizes the mean square error, i.e.,*

$$\sum_{\alpha \in U_A} ((m_{opt} \cdot \alpha + t_{opt}) - \delta(\alpha))^2 = min$$

Then the only problem left is how to construct this optimal line when the boundary function is given. We can use the algorithm proposed by Achtert et al. [1] where they try to conservatively approximate the k-nearest neighbor distances for every k by a linear function. We briefly summarize this algorithm below.

- First, this optimal line must interpolate at least one point, called *anchor point*, of the *upper convex null (UCH)* of the bf. The UCH is a sequence extracted from bf,

$$UCH = (\langle \alpha_1, \delta(\alpha_1) \rangle, \ldots, \langle \alpha_u, \delta(\alpha_u) \rangle$$

  where $\alpha_1 = 0, \alpha_u = 1$ and $\forall i < j, \alpha_i < \alpha_j$. The most important property of UCH is that the line segments composed by connecting adjacent points form a "right turn", i.e., the slopes of the line segments are monotonically decreasing. Given the bf, its UCH can be constructed efficiently in linear time [3].

- Next, a bisection search is performed to find the correct anchor point. The algorithm selects the median point p of the UCH as the first anchor point and computes its *anchor optimal line (AOL)*, which is a line interpolating the anchor point while minimizing the objective function. (a) If both the direct predecessor and successor of p are not above AOL, the global optimal line is found. (b) If successor (predecessor) of p is above AOL, the algorithm proceeds recursively with the right (left) half of the UCH.

Figure 4 illustrates the UCH as well as the L_{opt}. Once the L_{opt} has been derived, it can be used to estimate the MBR of α-cut of a fuzzy object A, by the following formula,

$$M_A^{i+}(\alpha)^* = \min \left\{ \begin{array}{l} M_A^{i+}(1) + (m_{opt}^{i+}(A) \cdot \alpha + t_{opt}^{i+}(A)), \\ M_A^{i+}(0) \end{array} \right.$$

$$M_A^{i-}(\alpha)^* = \max \left\{ \begin{array}{l} M_A^{i-}(1) - (m_{opt}^{i-}(A) \cdot \alpha + t_{opt}^{i-}(A)), \\ M_A^{i-}(0) \end{array} \right. \quad (2)$$

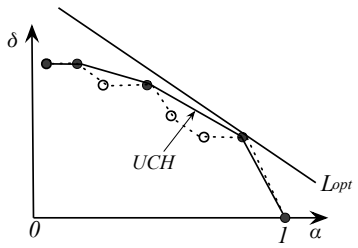


Figure 4: Approximating boundary function

The purpose of using min (max) operator is to assure $M_A(\alpha)^*$ not worse than the MBR of A_s. Moreover, the conservative property of L_{opt} guarantees that $M_A(\alpha)$ is always enclosed by $M_A(\alpha)^*$. Then we can safely use the $MinDist$ between $M_A(\alpha)^*$ and $M_Q(\alpha)$ as a tighter lower bound for their α-distance, i.e.,

$$d_\alpha^-(A, Q) = MinDist(M_A(\alpha)^*, M_Q(\alpha))$$

Figure 5 demonstrates the improvement of the lower bound, where d_α^- is closer to the actual α-distance compared to the original $MinDist$. To optimize the basic AKNN search by d_α^-, we just need to additionally store the L_{opt} (i.e., m_{opt} and t_{opt}) as well as the MBR of the kernel for each fuzzy object in the leaf node. Whenever a leaf node A is about to be inserted into the queue, we compute the tighter MBR, $M_A(\alpha)^*$, using the additional information and evaluate $d_\alpha^-(A, Q)$ as the associated value of this entry.

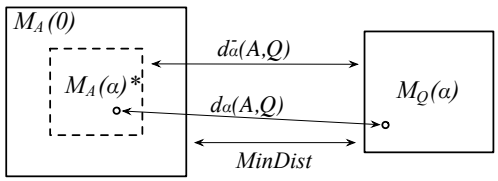


Figure 5: Improvement of lower bound

3.3 Lazy Probe

The basic algorithm probes the object and evaluates its actual distance with query object, whenever it encounters a leaf node. For the point object, it is suitable since the distance evaluation is very fast. But for the fuzzy object, we may want to delay this costly evaluation as long as possible. This motivates us to propose a *lazy probe optimization*.

The main idea is to maintain another priority queue $\mathcal{G}$ which can accommodate at most k elements as a buffer. The search principle is still the same with the basic method. But when a leaf node E is dequeued, we do not immediately probe the object it refers to. Instead, we compare its distance lower bound against the upper bound of every element U already in the buffer $\mathcal{G}$ and re-insert E into $\mathcal{G}$. If there exists some element U in $\mathcal{G}$ satisfying $d_\alpha^+(U, Q) < d_\alpha^-(E, Q)$, we can be sure that U is better than all objects left in $\mathcal{H}$. Even though we cannot decide whether U is better than the objects in $\mathcal{G}$, since there are at most k elements in $\mathcal{G}$, U is guaranteed to be in the top-k.

This algorithm will not retrieve the object until it has to do so, i.e., the buffer $\mathcal{G}$ is overflow. In other words, the lazy probe makes all the object retrieval mandatory. Algorithm 2 illustrates the main

structure of this optimization. We use the maximum distance between $M_A(\alpha)^*$ and $M_Q(\alpha)$ as the upper bound of their α-distance, which can be computed by the following equation given two MBR, M_A and M_B in d-dimensional space.

$$MaxDist(M_A, M_B) = \sqrt{\sum_{i=1}^{d} \max\{|M_A^{i+} - M_B^{i-}|, |M_A^{i-} - M_B^{i+}|\}^2} \quad (3)$$

Figure 6 shows how this optimization works. Suppose a new AKNN query is issued from Q. At some stage of the search, the priority queue $\mathcal{H}$ contains leaf nodes A, B, C and queue $\mathcal{G}$ is empty. Then A is popped from the queue first since it has smallest $d_\alpha^-(A, Q)$, and re-inserted into $\mathcal{G}$. The next element dequeued is B. Since $d_\alpha^-(B, Q) < d_\alpha^+(A, Q)$, we insert B into $\mathcal{G}$ as well. Afterwards, C is popped and found that $d_\alpha^-(C, Q) > d_\alpha^+(A, Q)$, then A can be removed from $\mathcal{G}$ and directly added to the result set without probing on hard disk.

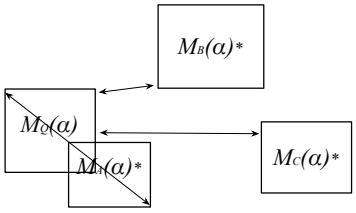


Figure 6: Lazy probe optimization

3.4 Improving the Upper Bound

According to the definition of α-distance, for any point $a \in A, b \in B$, if $\mu_A(a) \geq \alpha, \mu_B(b) \geq \alpha$, then the distance between a and b naturally upper bounds the α-distance of A, B. In the light of this observation, we propose to improve the upper bound of α-distance by the following offline processing.

- We randomly choose a point from the kernel of each fuzzy object A as its *representative point*, denoted by $rep(A)$, and store it in the leaf node of the R-tree.
- For the query object Q, we randomly sample n points from its α-cut and form the sample set Q'_α, where $n \ll |Q_\alpha|$.

LEMMA 1. *Given a fuzzy object A and a query object Q, their α-distance is upper bounded by the minimum distance between $rep(A)$ and Q'_α, i.e.,*

$$d_\alpha^+(A, Q) = \min_{q \in Q'_\alpha} ||rep(A) - q|| \geq d_\alpha(A, Q)$$

The proof is omitted due to its trivialness. Since n is quite small compared to $|Q_\alpha|$, the evaluation of d_α^+ can be very efficient. Besides, it does not require the retrieval of the fuzzy object, hence will not incur extra I/O cost. The effect of the improved upper bound is shown in Figure 7, where q_1, q_2, q_3, q_4 are the sampled points from Q_α.

4. RKNN QUERY PROCESSING

The AKNN query restricts the user to get the kNN set on a single probability threshold. But sometimes, they may want to to see how the kNN set varies on different thresholds. In such a case, they can issue an RKNN query where a probability range instead of a single probability threshold is accepted.

Algorithm 2: Lazy Probe

Input: $root, Q, \alpha, k$
Output: NN: the result set

```
1  initialize priority queue H, G;
2  enqueue ⟨root, MinDist(M_Q(α), M_root)⟩ into H;
3  while |NN| < k and H, G is not empty do
4      if |G| > k − |NN| then
5          E ← dequeue G;
6          if E is a leaf node then
7              probe E and enqueue ⟨E, d_α(E, Q)⟩ into G;
8          else
9              add E to NN;
10     else
11         E ← dequeue H;
12         if E is a leaf node then
13             add the element U of G which satisfies
               d_α^+(U, Q) < d_α^-(E, Q) into NN and remove U
               from G;
14             add E into G;
15         else
16             for each sub-entry V of E do
17                 if V is a leaf node then
18                     enqueue ⟨V, d_α^-(V, Q)⟩ into H;
19                 else
20                     enqueue ⟨V, MinDist(V, Q)⟩ into H;
21 return NN;
```

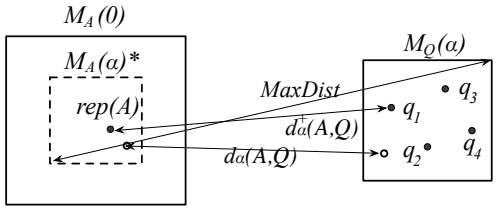


Figure 7: Improvement of upper bound

To process the RKNN query, the most straightforward approach is to issue an AKNN query at every α within the range. Even though the number of probabilities is infinite in the continuous range, we can always get the exact results by enumerating all values in $U_\mathcal{D}$ which is the universe of all membership values of the object set, since the α-distances of all objects with respect to Q do not change at $\alpha \notin U_\mathcal{D}$. However, due to the huge cardinality of $U_\mathcal{D}$, the cost of this method could become prohibitive. We call this method *naive approach*.

In the sequel, we will introduce a basic RKNN search approach which is more efficient than the naive approach. Moreover, we develop several pruning rules to further improve the performance.

4.1 Basic RKNN Search

Before discussing the algorithm, we first introduce the concept of *critical probability set*.

DEFINITION 7. *Given a fuzzy object A and a query object Q,*

Algorithm 3: Basic RKNN Search

Input: $root, Q, k, [\alpha_s, \alpha_e]$
Output: NN: the result set

$\alpha \leftarrow \alpha_s$;
while $\alpha \leq \alpha_e$ **do**
  $NN_\alpha \leftarrow$ AKNN search at threshold α;
  for *each* $A \in NN_\alpha$ **do**
    $\beta_A \leftarrow \min_{\alpha' \in \Omega_A}\{\alpha' \geq \alpha\}$;
  $\alpha^* \leftarrow \min_{A \in NN_\alpha} \beta_A$;
  add $\langle NN_\alpha, [\alpha, \alpha^*]\rangle$ into NN;
  $\alpha \leftarrow \alpha^* + \epsilon$;
return NN

the critical probability set of A with respect to Q, denoted by $\Omega_Q(A)$, is defined as

$$\Omega_Q(A) = \{\alpha \in (0,1] | \nexists \beta > \alpha, d_\beta(A,Q) = d_\alpha(A,Q)\}$$

Intuitively, $\Omega_Q(A)$ refers to all the end points of the horizontal line segments on the curve of $d_\alpha(A,Q)$, as shown in Figure 8. The semantics of its each element is that the α-distance is about to change (increase) beyond this probability.

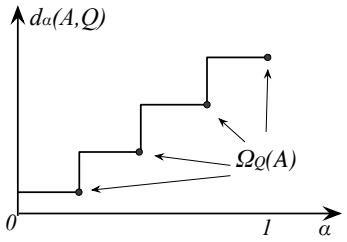


Figure 8: Critical probability set

LEMMA 2. *Suppose object A is one of the k nearest neighbors at some probability α. Let α' denote the minimum critical probability not less than α, i.e., $\alpha' = \min\{\beta \in \Omega_Q(A) | \beta \geq \alpha\}$. Then A remains in the kNN set within the range $[\alpha, \alpha']$.*

PROOF. By definition, the α-distance between A and Q does not change within the range $[\alpha, \alpha']$. On the other hand, though the distances of other objects with Q may vary, they are only possible to increase according to the monotonicity of α-distance. So A is guaranteed to remain in the kNN set within this range. □

Motivated by Lemma 2, we can design a basic RKNN search algorithm as follows.

Given an RKNN query with $[\alpha_s, \alpha_e]$ as the range, we issue an AKNN query at α_s and find the k nearest neighbor set NN_{α_s}. For each object $A \in NN_\alpha$, we find the next smallest critical probability after α_s, and select their minimum value α'. According to Lemma 2, the distance of all objects in NN_α will not change within the range $[\alpha_s, \alpha']$. Then, another AKNN query is issued again to find the new kNN set at probability $\alpha' + \epsilon$, where ϵ is a small enough real value (e.g. the precision of floating number). The above steps will repeat until it reaches α_e. This method is more efficient than the naive method since it only considers the critical probability values which is only a small fraction of the original membership set.

4.2 Reducing Search Space

Although the size of critical probability set is considerably smaller than $U_\mathcal{D}$ in the naive approach, the basic algorithm still requires many AKNN search against the whole dataset, which will cause a number of R-tree traversal and hence great IO overhead. To enhance its efficiency, our first goal is to reduce the search space by pruning most disqualifying objects.

LEMMA 3. *Given a RKNN query $(Q, k, [\alpha_s, \alpha_e])$, B is the k-th nearest neighbor at α_e, then object A cannot be a result of this RKNN query if $d_{\alpha_s}(A,Q) > d_{\alpha_e}(B,Q)$.*

PROOF. Let NN_{α_e} be the k nearest neighbor set at α_e. For any $\alpha \in [\alpha_s, \alpha_e]$ and $P \in NN_{\alpha_e}$, we have

$$\begin{aligned} d_\alpha(P,Q) &\leq d_{\alpha_e}(P,Q) \leq d_{\alpha_e}(B,Q) \\ &< d_{\alpha_s}(A,Q) \leq d_\alpha(A,Q) \end{aligned} \tag{4}$$

So all the objects in NN_{α_e} have smaller α-distance than A to Q. In other words, there are at least k objects closer than A at any $\alpha \in [\alpha_s, \alpha_e]$, which means A cannot be a result in this range. □

Consider an RKNN query with $k = 2$ and $I = [0.3, 0.6]$ is issued against the four objects in Figure 9. According to Lemma 3, object D cannot belong to the result set within the range I, since B is the 2-th nearest neighbor at $\alpha = 0.6$ and $d_{0.3}(D,Q) > d_{0.6}(B,Q)$.

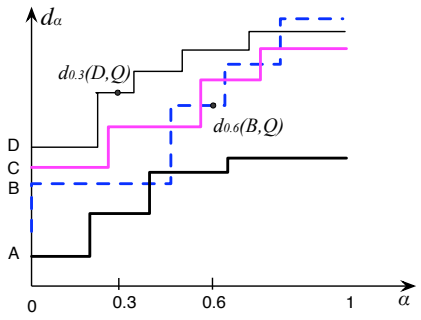


Figure 9: Reducing search space

Algorithm 4 shows the new search strategy optimized by Lemma 3. Specifically, given an RKNN query with $I = [\alpha_s, \alpha_e]$, we first find the k nearest neighbor set by issuing an AKNN query with $\alpha = \alpha_e$. Then we use the k-th nearest neighbor distance as the radius to perform a range search at $\alpha = \alpha_s$. Only the objects included within the radius are the candidates which are possible to be the k nearest neighbors at some probabilities within I. So we only need to search for the final results from candidate set $\mathcal{C}$ instead of the whole dataset at each critical probability. The size of set $\mathcal{C}$ is usually small and can fit into main memory, thus a large amount of IO operations can be avoided.

4.3 Improving Candidate Refinement

With the help of Lemma 3, the search space is significantly reduced. However, high computation cost still exists for the candidate refinement, where we have to check lots of critical probability values within the range. So our next goal is to accelerate the candidate refinement.

LEMMA 4. *Suppose an object A belongs to the kNN set at some probability α, and object B is the $(k+1)$-th nearest neighbor at α. Then A is guaranteed to be in the kNN set within the range $[\alpha, \alpha']$ as long as $d_{\alpha'}(A,Q) < d_\alpha(B,Q)$.*

Algorithm 4: Reducing Search Space

Input: $root, Q, k, [\alpha_s, \alpha_e]$
Output: NN: the result set
1 $NN_{\alpha_e} \leftarrow$ AKNN search at α_e;
2 $r \leftarrow$ the k-th nearest neighbor distance at α_e from NN_{α_e};
3 candidate set $\mathcal{C} \leftarrow$ perform a range search with radius r at probability α_s to get the candidate set;
4 $NN \leftarrow$ refine candidate set $\mathcal{C}$ using the method of basic RKNN search;
5 **return** NN

PROOF. Let P be any object that **does not belong to** the k nearest neighbors of Q at α, i.e., $P \in \mathcal{D} \backslash NN_\alpha$. Naturally, $d_\alpha(P,Q) \geq d_\alpha(B,Q)$. Then, for any probability $\beta \in [\alpha, \alpha']$, we have

$$\begin{aligned} d_\beta(A,Q) &\leq d_{\alpha'}(A,Q) < d_\alpha(B,Q) \\ &\leq d_\alpha(P,Q) \leq d_\beta(P,Q) \end{aligned} \tag{5}$$

That means all the objects in $\mathcal{D} \backslash NN_\alpha$ have larger distance than A at any probability within $[\alpha, \alpha']$. In other words, there are at most k objects which could be closer than A with respect to Q. Therefore A must be one of the k nearest neighbors within $[\alpha, \alpha']$. □

As a special case of Lemma 4, if we set α to α_s and α' to α_e, then it is safe to conclude that object A is a qualifying result across the entire probability range.

Figure 10 illustrates how Lemma 4 can help improve the candidate refinement. Consider an RKNN query with $k = 2$ and $I = [0.3, 0.6]$. Since object C is the 3-th nearest neighbor at $\alpha = 0.3$ and $d_{0.5}(A,Q) < d_{0.3}(C,Q)$, by Lemma 4 object A is guaranteed to be in the 2NNs within $[0.3, 0.5]$. The refinement continues until it reaches $\alpha = 0.5$. Again, as object B is the 3-th nearest neighbor at $\alpha = 0.5$ and $d_{0.6}(A,Q) < d_{0.5}(B,Q)$, A is still a result in $[0.5, 0.6]$. Thus, by checking only two probability thresholds, we can already conclude A is a result within the range $[0.3, 0.6]$.

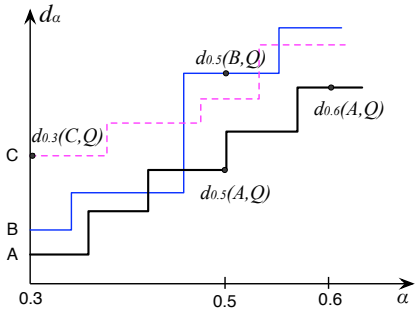


Figure 10: Improving candidate refinement

Motivated by Lemma 4, whenever we have obtained the kNN set at some α, it is often helpful to look for the "safe range" for each object in the kNN set. By this means, a large number of probability checking can be avoided. Based on this, we modify the basic RKNN search to improve candidate refinement, which is shown in Algorithm 5. In each iteration of the algorithm, we search the candidate set $\mathcal{C}$ to get the k nearest neighbors at current threshold α. In addition, we also obtain the $k+1$-th nearest neighbor distance at α, denoted by d_{k+1}. Then for each object $A \in NN_\alpha$, we remove

Algorithm 5: Improving Candidate Refinement

Input: $\mathcal{C}, Q, k, [\alpha_s, \alpha_e]$
Output: NN: the result set
1 $\alpha \leftarrow \alpha_s$;
2 initialize $\mathcal{C}'$ to be empty;
3 **while** $\alpha \leq \alpha_e$ **do**
4 $\quad \mathcal{C}' \leftarrow$ find the objects in NN which qualify at α;
5 $\quad NN_\alpha \leftarrow$ search $\mathcal{C} \backslash \mathcal{C}'$ for the $k - |\mathcal{C}'|$ nearest neighbors at α;
6 $\quad d_{k+1} \leftarrow$ the $k+1$-th nearest neighbor distance at α;
7 $\quad$ **for** *each* $A \in NN_\alpha$ **do**
8 $\quad\quad \beta \leftarrow \max_{\alpha \in \Omega_Q(A)} \{d_\alpha(A,Q) < d_{k+1}\}$
9 $\quad\quad$ add $\langle A, [\alpha, \beta] \rangle$ into NN;
10 $\quad\quad \Omega_Q(A) \leftarrow \Omega_Q(A) \backslash \{\alpha \in \Omega_Q(A) | d_\alpha(A,Q) < d_{k+1}\}$;
11 $\quad \alpha^* \leftarrow \min_{A \in NN_\alpha \cup \mathcal{C}'} \Omega_Q(A)$;
12 $\quad \alpha \leftarrow \alpha^* + \epsilon$;
13 **return** NN

the elements $\alpha_1 < \alpha_2 < \ldots < \alpha_n$ of $\Omega_Q(A)$ at which its distance is less than d_{k+1}. Since A is guaranteed to be a result within $[\alpha, \alpha_n]$, we can immediately add A along with this range into the result set NN. At the next round, we first obtain the objects which are still in the safe range into a set $\mathcal{C}'$. Then we only need to search the set $\mathcal{C} \backslash \mathcal{C}'$ for the top $k - |\mathcal{C}'|$ objects. This algorithm is more efficient than the basic RKNN search, since it removes lots of critical probability values during the refinement process, which should have been checked by the basic method.

5. COMPLEXITY ANALYSIS

In this section, we try to estimate the number of object access during the AKNN search process since retrieving objects from hard disk and computing their α-distances to query are the most costly parts in the whole algorithm. In order to make the analysis feasible, we assume the dataset is formed by *ideal fuzzy objects*.

DEFINITION 8. *An ideal fuzzy object A is circle (or sphere), and the radius of its α-cut is characterized by a function $R : \alpha \rightarrow radius$.*

By assuming the data space is composed of a set of ideal fuzzy objects in $2d$ space, our problem is, given a query object Q and a probability threshold α, to estimate the number of object access during the AKNN search in the data space. For the basic AKNN search, the objects to be accessed is the ones whose $MinDist$ with Q is smaller than $d_{knn}(\alpha)$, where $d_{knn}(\alpha)$ is the α-distance between Q and its k-th nearest neighbor. So we need to estimate $d_{knn}(\alpha)$ first. By representing each fuzzy object with its center, the dataset becomes a point set. Then we can borrow some formulas from existing work [16] to estimate the average number of neighbors $nb(\epsilon, `shape')$ of a point Q within distance ϵ from Q, using the concept of correlation fractal dimension of the point set:

$$nb(\epsilon, `shape') = \left( \frac{vol(\epsilon, `shape')}{vol(\epsilon, `rect')} \right)^{\frac{D_2}{E}} \cdot (N-1) \cdot (2\epsilon)^{D_2}$$

where N is the number of points, D_2 is the correlation fractal dimension and $vol(\epsilon, `shape')$ indicates the volume of a shape of radius $\epsilon$. In a 2-d dimensional space ($E = 2$), we want to estimate the minimum $\epsilon$ that encloses $k$ points. As $vol(\epsilon, `circle') = \pi\epsilon^2$

and $vol(\epsilon, `rect') = (2\epsilon)^2$, the above equation can be simplified to:

$$\begin{aligned} nb(\epsilon, `circle') &= (\frac{\pi\epsilon^2}{4\epsilon^2})^{\frac{D_2}{2}} \cdot (N-1) \cdot (2\epsilon)^{D_2} \\ &= (N-1) \cdot (\epsilon\sqrt{\pi})^{D_2} \end{aligned}$$

By substituting $nb(\epsilon, `circle')$ with k, and $D_2 = 2$ for a uniform dataset, we get:

$$\epsilon = \frac{1}{\sqrt{\pi}}\sqrt{\frac{k}{N-1}} \quad (6)$$

The ϵ derived from Equation (6) can be treated as distance from the center of query to the center of its k-th nearest neighbor. So their α-distance is:

$$d_{knn}(\alpha) = \epsilon - 2 \cdot R(\alpha)$$

Then the problem turns into estimating the number of leaf node accessed by a range query, for which a formula has been given in [16]:

$$\begin{aligned} L &= \frac{N-1}{C_{avg}} \cdot ((\frac{C_{avg}}{N})^{1/D_0} + 2d)^{D_2} \\ C_{avg} &= C_{max} \cdot U_{avg} \end{aligned} \quad (7)$$

where d is the search range, D_0 is the Hausdorff fractal dimension of the dataset (≈ 2 for uniform set), C_{max} is the maximum node capacity and U_{avg} is the average space utilization of the R-tree nodes. By substituting d with $d_{knn}(\alpha) + R(\alpha)$, we can estimate the average number of object accessed as the function of α:

$$L = \frac{N-1}{C_{avg}} \cdot (\sqrt{\frac{C_{avg}}{N}} + 2(\frac{1}{\sqrt{\pi}}\sqrt{\frac{k}{N-1}} - R(\alpha)))^2 \quad (8)$$

From Equation (8) we can see that, more objects need to be accessed as N, k or α increases independently.

6. EXPERIMENTS

In this section, we perform extensive experiments to verify the efficiency of the proposed methods and optimizations on both synthetic and real datasets. All the algorithms are implemented in Java and run on a normal PC with Pentium IV 2.4 GHz CPU and 1GB memory.

6.1 Dataset

The datasets we use for experiments are setup as follows.

- For the synthetic dataset, each object is a circle with radius of 0.5 and containing 1K uniformly distributed points, whose membership values follow the two dimensional Gaussian distribution with mean at the center of circle and $\sigma_x = \sigma_y = 0.5$. In order to assure the kernel set is not empty, we normalize the probability values across 0 to 1.
- For the real dataset, each object is formed by 1K points randomly sampled from the horizontal cell identified by probabilistic segmentation [14],[15]. Then we normalize the positions of all points to restrict them into a 1×1 square. Similar with the synthetic dataset, we also normalize the probability values across 0 to 1.

For both datasets, we generate N objects and randomly distribute them into 100×100 space. All the actual points are stored in files and we index the fuzzy regions by R-tree. In the following experiments, we measure the number of object access from hard disk and running time of the algorithms under different parameter settings. Table 2 summarizes the parameters and their default values.

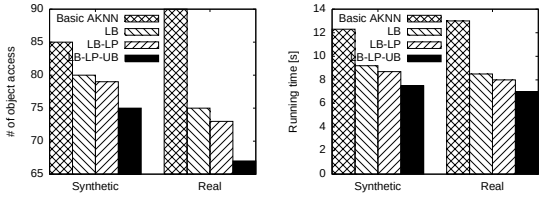
Parameter	Default value
Number of objects N	50000
Number of results k	20
Probability threshold α	0.5
Probability range L	0.2

Table 2: Parameter settings

6.2 AKNN Performance Evaluation

In this subsection, we compare the performance of the basic AKNN search algorithm against its competitors, namely, AKNN search with improved lower bound (LB), LB with lazy probe (LB-LP), and LB-LP with improved upper bound (LB-LP-UB).

6.2.1 *Effect of dataset*



(a) No. of object access (b) Running time

Figure 15: Effect of dataset

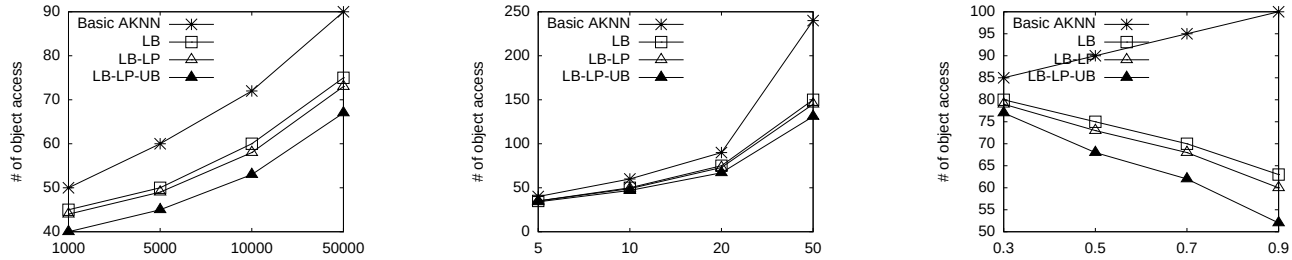
First we run all algorithms on both datasets to see the impact of different fuzzy objects by using the default parameter settings. The results are shown in Figure 15a and 15b. As expected, the basic method exposes the worst performance on both datasets. More specifically, LB can avoid quite a few unnecessary object access and distance calculations by applying a tighter approximation for the α-cut of fuzzy objects. The benefit of lazy probe, however, is not significant since the upper bound it uses is rather loose. By further applying the tighter upper bound, LB-LP-UB method achieves the best performance. As each algorithm shows similar performance on both datasets, we will just show the results on real dataset to keep our presentation concise.

6.2.2 *Effect of N*

We then study the impact of dataset size on the performance of AKNN search algorithms by varying the number of objects from 1K to 50K. As shown in Figure 11a, all the algorithms need to access more objects on hard disk in order to determine the k nearest neighbors with a larger dataset. This is because, when the number of objects grows, the density of whole data space becomes higher, which makes it more difficult to prune objects by checking distance lower bound. From Figure 12a, their running time also increases because more IO operation is invoked and more α-distances need to evaluated. We also notice that the performance of all algorithms are still comparable when the dataset is relatively small, since the simple lower bound serves as a tolerable approximation when the data space is sparse. The advantages of the proposed optimizations become more obvious in larger dataset.

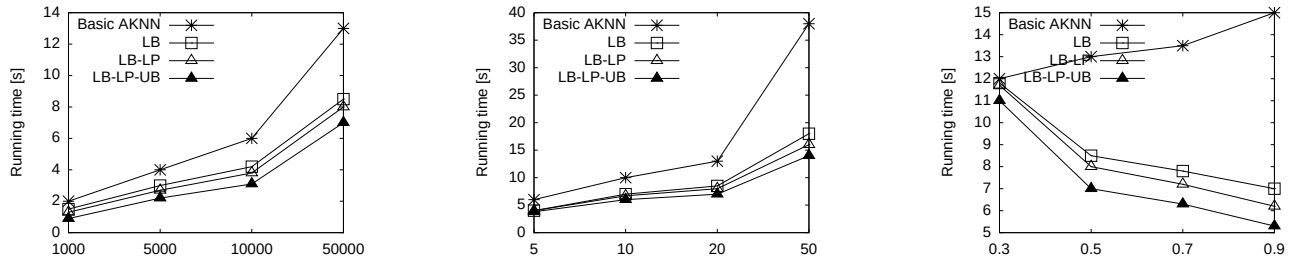
6.2.3 *Effect of k*

Next we compare the performance of all methods by varying k from 5 to 50. According to Figure 11b and 12b, the performance of all algorithms deteriorate as k increases. This is due to the fact that the best-first search strategy has to verify all the objects A satisfying $d_\alpha^-(A, Q) \leq d_{knn}$ where d_{knn} is the α-distance of k-th



(a) Diff. N (b) Diff. k (c) Diff. α

Figure 11: Object access of AKNN search



(a) Diff. N (b) Diff. k (c) Diff. α

Figure 12: Running time of AKNN search

nearest neighbor. As k increases, d_{knn} increases as well, which means more objects need to be retrieved. On the other hand, the optimized algorithms are less sensitive to the variation of k, due to the more accurate estimation of α-distance.

6.2.4 *Effect of* α

We also study the impact of the probability threshold by varying α from 0.3 to 0.9. From Figure 11c and 12c we can observe that the number of object access as well as the running time increases for the basic search method. This is due to the greater α-distance of the k-th nearest neighbor when α is higher. As a consequence, more candidates are retrieved for verification. Although the optimized algorithms face the same situation, their performance are heading for the opposite direction, i.e., fewer objects accessed and less CPU cost. This is because, as α increases, the improved lower and upper bound can reflect the actual α-distance more accurately, making the best-first search more efficient.

6.3 RKNN Performance Evaluation

To verify the efficiency of our proposed methods for answering RKNN query, we compare the performance of the basic algorithm with the optimized algorithms, i.e., basic RKNN method with reduced search space (RSS) and RSS with improved candidate refinement procedure (RSS-ICR).

6.3.1 *Effect of* N

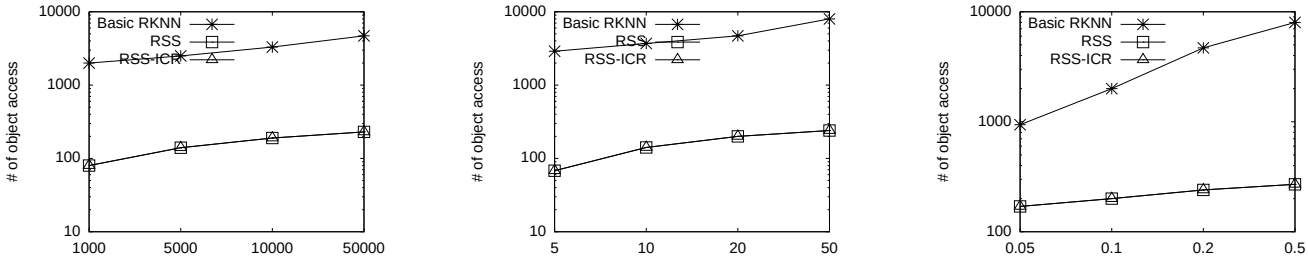
The first set of experiments tests the scalability of all RKNN schemes by varying the dataset size from 1K to 50K. Results are shown in Figure 13a and 14a, from which we can see that performance of all algorithms degrade as the dataset grows, but the optimized algorithms are constantly superior than the basic method by at least one order of magnitude. Since the basic method relies on invoking lots of AKNN search procedure, it needs to traverse the R-tree many times and incurs high IO and computation cost. With the AKNN performance getting worse in larger dataset, the basic RKNN search deteriorates even faster. On the other hand, RSS only issues an AKNN query at the end of the probability range, and then finds all the candidates by a range search at the beginning of the probability range. Other than that, no extra IO operation is invoked. That explains why the number of object access by RSS is within the same magnitude as the corresponding AKNN search. RSS-ICR adopts the same strategy to obtain the candidate set, thus having the same performance in terms of object probe. But its CPU cost is lower than RSS since it avoids a lot of critical probability checking in the candidate refinement process.

6.3.2 *Effect of* k

Then we study performance trend of all algorithms when different numbers of results are required. Figure 13b and 14b clearly demonstrates the remarkable advantage of our optimizations, the cause of which is similar with the last set of experiments. For all algorithms, their running time increases are also caused by more critical probability values to be checked as k increases.

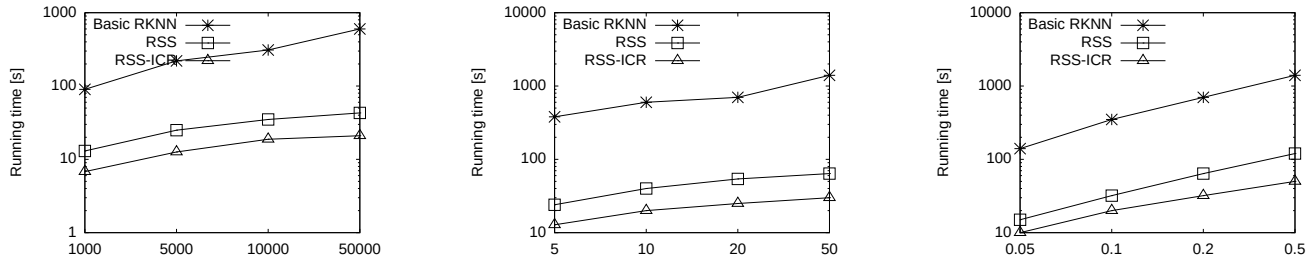
6.3.3 *Effect of* L

Finally we examine the impact of different lengths of probability range on the performance of all algorithms by varying L from 0.05 to 0.5. As shown in Figure 13c and 14c, with L increasing, the performance of the basic algorithm deteriorates very fast. This is because the number of AKNN queries issued by the basic algorithm increases rapidly. On the contrary, the number of object access for the optimized algorithms is not sensitive to the variation of L. Their running time increase is mainly caused by the longer candidate refinement process. By avoiding exhaustively checking all critical probability values, the advantage of RSS-ICR becomes more significant when the probability range is broader.



(a) Diff. N (b) Diff. k (c) Diff. L

Figure 13: Object access of RKNN search



(a) Diff. N (b) Diff. k (c) Diff. L

Figure 14: Running time of RKNN search

7. RELATED WORK

There is a large body of research work on nearest neighbor search in spatial databases. The most common type of nearest neighbor search is the point-kNN query that finds the k objects from a dataset that are closest to a query point q. Roussopoulos et al [19] proposed a depth first method that, starting from the root of R-tree, visits the entry with the minimum distance from q. The process is repeated recursively until the leaf level. During the backtracking to the upper level, the algorithm only visits entries whose minimum distance is smaller than the distance of the nearest neighbor already found. Hjaltason and Samet [11] implemented a best-first traversal that follows the entry with the smallest distance among all those visited. In order to achieve this, the algorithm keeps a priority queue with the candidate entries and their minimum distances from the query point. Furthermore, conventional NN search and its variations in low and high dimensional spaces have also received considerable attention due to their applicability in domains such as content based retrieval and multimedia database [13, 27, 32, 33]. But since it is difficult to map a fuzzy object to a single point while keeping the original information, the proposed algorithms for point-NN search cannot be directly applied to our problem.

With the proliferation of location-based e-commerce and mobile computing, a lot of attention has been received for moving-object databases, where the locations of data objects or queries (or both) are changing. Assuming object movement trajectories are known a priori, Saltenis et al [20] proposed the Time-Parameterized R-tree (TPR-tree) for indexing moving objects, in which the location of a moving object is represented by a linear function. Benetis et al [5] developed query evaluation algorithms for NN and reverse NN search based on TPR-tree. Tao et al [31] optimized the performance of the TPR-tree and extended it to the TPR*-tree. Continuous kNN monitoring for moving queries has also been investigated on stationary objects [30] and linearly moving objects [12]. The work in this category shares some features in common with our proposed kNN queries for fuzzy objects. In continuous kNN queries, the result set is only valid within certain time span. Correspondingly, in our work each k nearest neighbor is associated with its qualifying probability range. Yet essential differences still exist. In moving object databases, what varies is the location of an object; while in fuzzy object databases, the dynamic part is the composition of an object, which alters with the probability threshold.

As data uncertainty attracts more research interests, KNN query has also been extended to uncertain databases. Cheng et al. [8] proposed the *probabilistic nearest-neighbor query (PNNQ)* in uncertain environments, which finds a set of data objects that have non-zero probability to be the nearest neighbor of the query point. Tao et al [29] developed U-tree to index multi-dimensional uncertain data with arbitrary probability density functions. Although an uncertain object is often represented by a pdf region, which looks similar with the fuzzy object model in this paper, essential difference exists. The accumulation of probabilities within a pdf region is equal to one, which is the basic assumption for most of their techniques to work correctly. Obviously, the fuzzy object model disobeys this assumption. That is why we cannot adopt their index method to support the kNN queries in this paper.

On the other hand, the notion of fuzzy objects has not been introduced to database field until Altman [2] adopted fuzzy set theoretic approach for handling imprecision in spatial databases. Afterwards, fuzzy data types such as fuzzy points, fuzzy lines, and fuzzy regions were defined in [22, 10]. Based upon that, simple metric operations such as the area of a fuzzy region and the length of a fuzzy line were further developed in [23], in which only unary functions were considered. On relationship between fuzzy objects, different models of fuzzy topological predicates, which character-

ize the relative position of two fuzzy objects toward each other were discussed in [25, 28, 24]. However, it remains largely untouched to answer more advanced spatial queries such as the nearest neighbor query, which is addressed in this paper.

8. CONCLUSION

Although kNN query is one of the most useful spatial queries and fuzzy objects can often be found in many important applications, considering both has received rather limited attention. In this paper we have studied this problem in depth, defining two new types of kNN queries for fuzzy objects. We have developed efficient algorithms to answer these queries by extending the R-tree indexing structure and deriving several highly effective heuristic rules. Extensive experiments on both synthetic and real datasets have shown that our optimized algorithms achieve superior performance than the baseline approaches constantly. Given the relevance of fuzzy objects to a wide range of applications, we expect this research to trigger further work in this area, opening a way for other advanced queries such as spatial join queries, reverse nearest neighbor queries and skyline queries.

Acknowledgement: The research reported in this paper has been partially supported by ARC Centre of Excellence in Bioinformatics.

9. REFERENCES

[1] E. Achtert, C. Bohm, P. Kroger, P. Kunath, A. Pryakhin, and M. Renz. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *SIGMOD*, pages 515 – 526, 2006.

[2] D. Altman. Fuzzy set theoretic approaches for handling imprecision in spatial analysis. *International Journal of Geographical Information Science*, 8(3):271–289, 1994.

[3] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9, 1979.

[4] A. Badel, J. Mornon, and S. Hazout. Searching for geometric molecular shape complementarity using bidimensional surface profiles. *Journal of Molecular Graphics*, 10(4):205–211, 1992.

[5] R. Benetis, C. Jensen, and G. Simonas. Nearest neighbor and reverse nearest neighbor queries for moving objects. In *IDEAS*, pages 44–53, 2002.

[6] I. Bloch. On fuzzy distances and their use in image processing under imprecision. *Pattern Recognition*, 32(11):1873–1895, 1999.

[7] B. Chaudhuri and A. Rosenfeld. On a metric distance between fuzzy sets. *Pattern Recognition Letters*, 17(11):1157–1160, 1996.

[8] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, pages 551–562, 2003.

[9] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest pair queries in spatial databases. *SIGMOD Record*, 29(2):189–200, 2000.

[10] A. Dilo, A. Rolf, and A. Stein. A system of types and operators for handling vague spatial objects. *International Journal of Geographical Information Science*, 21(4):397–426, 2007.

[11] G. Hjaltason and H. Samet. Distance browsing in spatial databases. *TODS*, 24(2):265–318, 1999.

[12] G. Iwerks, H. Samet, and K. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. In *VLDB*, pages 512–523, 2003.

[13] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *VLDB*, pages 215–226, 1996.

[14] V. Ljosa and A. Singh. Probabilistic segmentation and analysis of horizontal cells. In *ICDM*, pages 980–985, 2006.

[15] V. Ljosa and A. Singh. Top-k spatial joins of probabilistic objects. In *ICDE*, pages 566–575, 2008.

[16] A. Papadopoulos and Y. Manolopoulos. Performance of nearest neighbor queries in r-trees. In *ICDT*, pages 394–408, 1997.

[17] S. Peng, B. Urbanc, L. Cruz, B. Hyman, and H. Stanley. Neuron recognition by parallel potts segmentation. *National Academy of Sciences*, 100(7):3847–3852, 2003.

[18] F. Preparata and M. Shamos. *Computational geometry: an introduction*. Springer, 1985.

[19] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.

[20] S. Saltenis, C. Jensen, S. Leutenegger, and M. Lopez. Indexing the positions of continuously moving objects. *ACM Sigmod Record*, 29(2):331–342, 2000.

[21] C. Schmitz, N. Grolms, P. Hof, R. Boehringer, J. Glaser, and H. Korr. A stereological study using a novel three-dimensional analysis method to estimate the nearest neighbor distance distributions of cells in thick sections. *Cerebral Cortex*, 12(9):954–960, 2002.

[22] M. Schneider. Uncertainty management for spatial data in databases: fuzzy spatial data types. In *SSD*, pages 330–354, 1999.

[23] M. Schneider. Metric operations on fuzzy spatial objects in databases. In *ACMGIS*, pages 21–26, 2000.

[24] M. Schneider. A design of topological predicates for complex crisp and fuzzy regions. In *ICCM*, pages 103–116, 2001.

[25] M. Schneider. Fuzzy topological predicates, their properties, and their integration into query languages. In *ACMGIS*, pages 9–14, 2001.

[26] T. Seidl and H. Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *VLDB*, pages 506–515, 1997.

[27] T. Seidl and H. Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD*, pages 154–165, 1998.

[28] X. Tang and W. Kainz. Analysis of topological relations between fuzzy regions in a general fuzzy topological space. In *Symposium on Geospatial Theory, Processing and Applications*, 2002.

[29] Y. Tao, R. Cheng, X. Xiao, W. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, pages 922–933, 2005.

[30] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298, 2002.

[31] Y. Tao, D. Papadias, and J. Sun. The tpr*-tree: An optimized spatio-temporal access method for predictive queries. In *VLDB*, pages 790–801, 2003.

[32] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.

[33] C. Yu, B. Ooi, K. Tan, and H. Jagadish. Indexing the distance: An efficient method to knn processing. In *VLDB*, pages 421–430, 2001.

[34] L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.