The VLDB Journal (2012) 21:729–751
DOI 10.1007/s00778-012-0266-x

REGULAR PAPER

Spatial query processing for fuzzy objects

Kai Zheng · Xiaofang Zhou · Pui Cheong Fung · Kexin Xie

Received: 21 February 2011 / Revised: 14 November 2011 / Accepted: 26 January 2012 / Published online: 14 February 2012

Abstract Range and nearest neighbor queries are the most common types of spatial queries, which have been investigated extensively in the last decades 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 communities. Existing research on fuzzy objects mainly focuses on modeling basic fuzzy object types and operations, leaving the processing of more advanced queries largely untouched. In this paper, we propose two new kinds of spatial queries for fuzzy objects, namely *single threshold query* and *continuous threshold query*, to determine the query results which qualify at a certain probability threshold and within a probability interval, respectively. For efficient single threshold query processing, we optimize the classical R-tree-based search algorithm by deriving more accurate approximations for the distance function between fuzzy objects and the query object. To enhance the performance of continuous threshold queries, effective pruning rules are developed to reduce the search space and speed up the candidate refinement process. The efficiency of our proposed algorithms as well as the optimization techniques is verified with an extensive set of experiments using both synthetic and real datasets.

Keywords Range query · Nearest neighbor query · Fuzzy database · Probabilistic database

K. Zheng (✉) · X. Zhou · P. C. Fung · K. Xie
School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, QLD 4072, Australia
e-mail: kevinz@itee.uq.edu.au

X. Zhou
e-mail: zxf@itee.uq.edu.au

P. C. Fung
e-mail: g.fung@itee.uq.edu.au

K. Xie
e-mail: kexin@itee.uq.edu.au

X. Zhou
NICTA Queensland Research Laboratory, Sydney, NSW, Australia

1 Introduction

The capability of supporting spatial queries is one of the key features in spatial database management system (SDBMS), due to its broad range of applications including molecular biology [5], medical imaging [23], multimedia databases [38], and so on. After extensive study during the last decades, numerous efficient algorithms have been proposed in previous literatures [20,23,30,39,44,45], most of which target traditional SDBMS where the locations of data and queries are precise and static.

Recently, with the advent of mobile and ubiquitous computing, many spatial queries, such as range and kNN queries, have also been extended to moving object databases [7,42] and uncertain databases [11]. This 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, one common assumption of these work is that the data to be processed are all *crisp objects*, i.e., the compositions and boundaries of objects are deterministic. However, in some real applications, such as biomedical image analysis and geographical information systems, this assumption may not be satisfied.

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,

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

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 [25]. For example, Fig. 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 [3,33,36,40], common spatial queries such as range and kNN queries still remain untouched at large. In this paper, we will study efficient algorithms to support the most common types of spatial queries, namely range and kNN queries, over fuzzy objects. These kinds of queries have many applications in the biomedical field such as brain aging study [32], Alzheimer's disease analysis [28], 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 spatial queries? Or can we adopt existing solutions to support this type of data? Based on our study, the answer is negative. For example, 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 query answering set is required. Specifically, we introduce *probability threshold* as a user-defined parameter in the definitions of spatial queries. 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 between our new queries and the traditional spatial queries lies in 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.

A preliminary version of this paper appeared in [47], in which we mainly focused on efficient nearest neighbor query processing for fuzzy objects. Here in this paper, we extend the work in several ways. First, we complement our previous work by introducing two new types of range queries, as the counterparts of the kNN queries for fuzzy objects. Second, we propose efficient processing algorithms for range queries within the same framework as the kNN queries. Third, we present more efficient online algorithms for evaluating α-distance. Fourth, the performance of our proposed approaches for range query processing as well as α-distance computation is demonstrated by more experiment results.

The remainder of this paper is organized as follows. In Sect. 2, the fuzzy object model and two new types of spatial queries are introduced. Proposed algorithms for answering single threshold and continuous threshold query are presented in Sects. 3 and 4. Then, in Sect. 5, we propose algorithms as well as optimizations to evaluate the α-distance more efficiently. It is followed by a brief analysis to estimate the average number of object access in Sect. 6. We show the experiment results in Sect. 7 and review the related work in Sect. 8. Finally, we conclude the paper in Sect. 9.

2 Models and queries

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

2.1 Fuzzy object model

Fuzzy objects are usually modeled by fuzzy sets [46], 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.

Table 1 Summary of notations

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
μ_A	The highest membership probability of 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. Besides, we use μ_A to denote the highest membership probability of all points in A.

Different from our preliminary version of the paper which assumes an non-empty kernel for the fuzzy objects, in this work the kernel of a fuzzy object can be either non-empty or empty. This relaxation of assumption is desired in varies situations like the produced images are of low quality, such that some objects may not exist at all.

Naturally, in order to define spatial queries, we need a scoring function to measure spatial closeness between two fuzzy objects in space. There exists some work [9,10] 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 query result 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 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:

$$d_\alpha(A, B) = \begin{cases} \min_{\langle a,b\rangle \in A_\alpha \times B_\alpha} ||a - b||, & \text{if } \alpha \leq \min(\mu_A, \mu_B) \\ +\infty, & \text{otherwise} \end{cases} \quad (1)$$

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. Given a user-specified probability threshold α, if the α-cuts of both objects are not empty, i.e., $\alpha \leq \min(\mu_A, \mu_B)$, we defuzzy the objects to their α-cuts and adopt the minimum distance as their distance measurement; otherwise, their α-distance is defined to infinite. 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. Evaluating α-distance involves finding the *closest pair* (CP) between two α-cuts, for which we will propose efficient algorithms in Sect. 5.

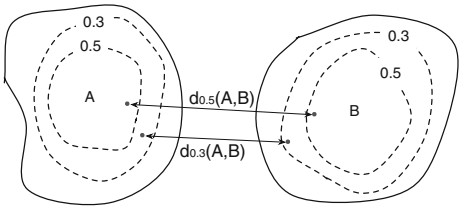


Fig. 2 α-Distance of fuzzy objects

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 exemplifies 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.

2.2 Query definitions

In this subsection, we propose two new types of range and kNN queries for fuzzy objects, namely the *single threshold query* (STQ) and *continuous threshold query* (CTQ).

Definition 4 *(Single threshold range query)* Given a fuzzy object dataset $\mathcal{D}$, a radius r, a probability threshold α, and a query (fuzzy) object Q, a single threshold range query (STR) retrieves the objects from $\mathcal{D}$, whose α-distances with respect to Q are not above r.

Definition 5 *(Single threshold kNN query)* Given a fuzzy object dataset $\mathcal{D}$, natural number k, probability threshold α, and query (fuzzy) object Q, a single threshold kNN query (STN) retrieves k objects from $\mathcal{D}$ which have the smallest α-distances with respect to Q.

The intuition behind this query definition is that we give users the freedom to decide the confidence level of information in the objects based on which the query results will be derived. In some other cases, one may want to try different probability thresholds to examine the differences of the results, to make sure no valuable information is discarded. Motivated by this kind of applications, we generalize the STQ by replacing the single probability threshold with a continuous interval of probability thresholds, resulting in the continuous threshold version of query definitions.

Definition 6 *(Continuous threshold range query)* Given a fuzzy object dataset $\mathcal{D}$, a radius r, a continuous probability interval $I = [\alpha_s, \alpha_e]$, and a query (fuzzy) object Q, a continuous threshold range query (CTR) returns a set of objects $\{\langle A, I_A\rangle | I_A \subseteq [\alpha_s, \alpha_e]\}$, where $\forall\alpha \in I_A$, A belongs to the CTR query results and $\forall\alpha \notin I_A$, A does not belong to the CTR query results. I_A is called the qualifying interval of A.

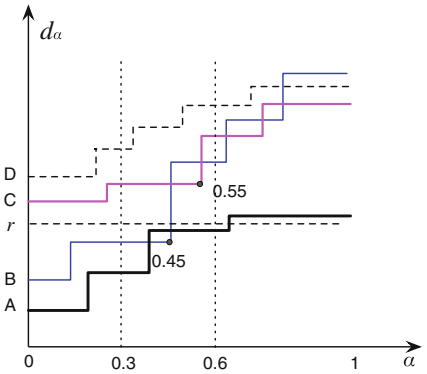


Fig. 3 Illustration of STQ and CTQ

Definition 7 *(Continuous threshold kNN query)* Given a fuzzy object dataset $\mathcal{D}$, a natural number k, a probability range $I = [\alpha_s, \alpha_e]$, and query (fuzzy) object Q, a continuous threshold kNN query (CTN) returns a set of objects $\{\langle A, I_A\rangle | I_A \subseteq [\alpha_s, \alpha_e]\}$, where $\forall\alpha \in I_A$, A belongs to the CTN query results and $\forall\alpha \notin I_A$, A does not belong to the CTN query results. I_A is called the qualifying interval of A.

Continuous threshold 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 results along with their qualifying ranges. Note that in this paper, the order of k nearest neighbors is not important (i.e., order insensitive). But our algorithms can be easily extended to handle the case where the order does matter.

We use Fig. 3 to exemplify the intuition of STQ and CTQ, where the α-distances of four fuzzy objects A, B, C, D with respect to the query object are shown as piecewise function curve. If we set probability threshold to 0.4, the STR query with radius r (indicated as the horizontal line) and the STN query with $k = 2$ will return the set $\{A, B\}$ as the result. But if we raise α to be 0.5, the result of the STR and STN query with the same parameters becomes A and $\{A, C\}$, respectively. Furthermore, if we change the single threshold to a continuous threshold interval $I = [0.3, 0.6]$, the result set of the corresponding CTR and CTN query should be $\{\langle A, [0.3, 0.6]\rangle, \langle B, [0.3, 0.45]\rangle\}$ and $\{\langle A, [0.3, 0.6]\rangle, \langle B, [0.3, 0.45] \cup (0.55, 0.6]\rangle, \langle C, (0.45, 0.55]\rangle\}$.

The new types of spatial queries for fuzzy objects are quite different from the traditional ones such as queries for point objects or continuous queries for moving objects, in the sense that each object may be fully, partially, or not involved at all in the query evaluation process, depending on the probability threshold specified by the users. In other words, objects themselves are not determined until the query is issued, which poses great challenges on the efficiency of query processing algorithms. In the following two sections, we will try to

overcome these difficulties and design efficient solutions to answer these kinds of queries.

3 Single threshold query processing

3.1 Basic algorithms

The most straightforward approach for answering the single threshold query is to linearly scan the whole dataset and calculate the α-distance with the query object for each object encountered. 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 borrow some ideas from traditional query processing, i.e., use R-tree as the index structure to prune the search space. However, we need to make some modifications to suit the characteristic of fuzzy objects.

First, 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. It can be easily proved that the minimum distance $MinDist$ between the MBRs of two fuzzy objects is a lower bound for their α-distance. Compared to the original α-distance, the evaluation of $MinDist$ is much more efficient and does not require the retrieval of the object. After all the objects are transformed to hyper-rectangles and indexed by R-tree, we can adopt the traditional spatial query processing techniques such as [20,30] to find a set of candidates for STR(STN) queries and obtain the final results by calculating the α-distances between the candidates and the query object.

Second, we additionally keep record of the highest membership probability μ_A in the leaf node of each object A, and further propagate the maximum value to their parent node. As such, during the search process, whenever the highest probability of a node (either branch node or leaf node) dequeued from the heap is found to be less than α, we can immediately skip this node since all the objects in this node have infinite distance with the query object according to Definition 3.

3.2 Improving the lower bound distance

The basic 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?

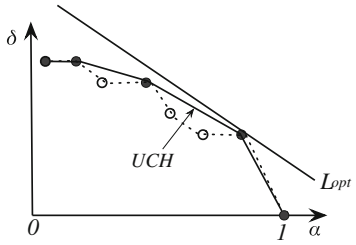


Fig. 4 Approximating boundary function

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 ith 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.,

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

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 Fig. 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 8 The optimal conservative approximation of the boundary function is a line

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

with the following constraints:

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

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

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

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

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 hull* (UCH) of the bf. The UCH is a sequence extracted from bf,

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

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 [4].
– 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,

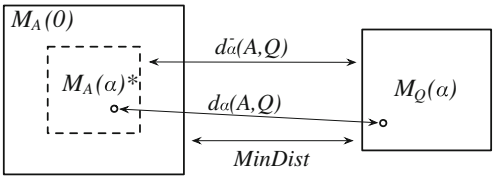


Fig. 5 Improvement of lower bound

$$\begin{aligned} M_A^{i+}(\alpha)^* &= \min \begin{cases} M_A^{i+}(1) + (m_{\text{opt}}^{i+}(A) \cdot \alpha + t_{\text{opt}}^{i+}(A)), \\ M_A^{i+}(0) \end{cases} \\ M_A^{i-}(\alpha)^* &= \max \begin{cases} M_A^{i-}(1) - (m_{\text{opt}}^{i-}(A) \cdot \alpha + t_{\text{opt}}^{i-}(A)), \\ M_A^{i-}(0) \end{cases} \end{aligned} \quad (2)$$

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 improve the basic algorithms with d_α^-, we only need to make slight modifications to the indexing structure. First, we 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 of the R-tree. Whenever a leaf node A is dequeued from the heap, we compute the tighter MBR, $M_A(\alpha)^*$ first using the additional information, and then evaluate $d_\alpha^-(A, Q)$ as the associated value of this entry. By this means, some objects can be filtered without accessing its content in the disk.

Second, in an intermediate entry e of the index, we keep $e.\text{MBR}^-$, $e.\text{MBR}^+$ which are the MBRs of the estimated boundary $M_A(0)*$ and kernel $M_A(1)*$ for all the objects A under e. Then, a linear function $e.L$ can be calculate based on each side of $e.\text{MBR}^-$ and $e.\text{MBR}^+$. When this entry is popped out from the heap at some stage of the search process, we can use $e.L$ to estimate an MBR $e.M(\alpha)^*$, which guarantees to cover the α-cuts of all the objects within the node e. As such, the minimum distance between $e.\text{MBR}(\alpha)^*$ and M_q lower bounds the α-distances between all the query object and all the objects in e.

3.3 Lazy probe

The basic algorithm probes the object in disk and evaluates its actual α-distance with the query object for all the candidates. This method is suitable for the point objects since the distance evaluation is very efficient. But for the fuzzy object, we may want to avoid this costly evaluation

as much as possible. This motivates us to propose the lazy probe optimization to further reduce the distance evaluation cost. The basic idea of this optimization is to include some objects based on an upper bound of the α-distance which is relatively easy to derive, rather than the α-distance. Before detailing the algorithms, we define 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.

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

Lazy probe for STR query. For the STR query, after the candidate set is derived as in the basic method, we do not access the content of each candidate immediately. Instead, their upper bound distances are evaluated and compared with the radius r. If the upper bound distance is smaller than r, the object is certain to be a query result, without its α-distance computed. Only when the upper bound is greater than or equal to r, we need to retrieve the object and figure out the α-distance.

Lazy probe for STN query. For the STN query, we 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. Algorithm 1 illustrates the main structure of this optimization. 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.

Figure 6 shows how this optimization works for STN queries. Suppose a new STN query with $k \geq 2$ is issued from Q.

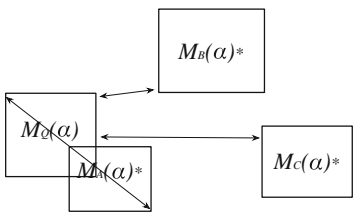


Fig. 6 Lazy probe optimization

Algorithm 1: Lazy Probe for STN Query

```
Input: root, Q, α, k
Output: NN: the result set
initialize priority queue H ← ⟨root, MinDist(M_Q(α), M_root)⟩;
initialize priority queue G ← ∅;
while |NN| < k and H, G is not empty do
    if |G| > k − |NN| then
        E ← dequeue G;
        if E is a leaf node then
            probe E and enqueue ⟨E, d_α(E, Q)⟩ into H;
        else
            add E to NN;
    else
        E ← dequeue H;
        if E is a leaf node or object then
            add U ∈ G that d_α^+(U, Q) < H(E).key into NN and
            remove U from G; // U is true hit without refinement
            enqueue ⟨E, H(E).key⟩ into G;
        else
            for each sub-entry V of E do
                if V is a leaf node then
                    enqueue ⟨V, d_α^-(V, Q)⟩ into H;
                else
                    enqueue ⟨V, MinDist(V, Q)⟩ into H;
return NN;
```

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. Afterward, 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.

Remark Improving the performance by utilizing both lower and upper bounding distances has also been proposed in [24]. The main differences between their method and lazy probe algorithm proposed in this paper are twofold. First, [24] adopts a multi-step paradigm for nearest neighbor search based on [39] while the lazy probe algorithm takes the incremental best-first search style originally proposed in [20]. Second, in [24], the search algorithm fetches the first k candidates from the ranking list based on lower bounding distance and identifies the true hits by using the upper bounding distances. Our algorithm tries to detect the true hits with the upper bounds promptly, whenever the next element is popped out from the priority queue. As such, our method may report parts of the results more quickly. Of course, this benefit comes with costs—the overall processing time may increase due to the prompt detection. We will include [24] for comparison in the experiments.

3.4 Improving the upper bound

In this part, we propose two independent techniques for improving the tightness of the upper bounding distance $MaxDist$, in order to maximize the benefit of the lazy probe algorithm.

MinMax distance-based method. Roussopoulos et al. [30] proposed a metric, $MinMaxDist$, which computes the minimum value of all the maximum distances between a query point and each face of an MBR. It was proved to be an upper bound for the distances between the query point and all the points within the MBR. Following this idea, we can extend this metric for two hyper-rectangles.

Given an d-dimensional rectangle M, we denote its face by $f_i(M), i = 1, 2, \ldots, 2d$. Each face $f_i(M)$ has $2(d-1)$ vertex points, denoted as $v_{ij}(M)$, $j = 1, 2, \ldots, 2(d-1)$. Then, the MinMax distance between two n-dimensional rectangles M_A and M_B can be calculated as:

$$MinMaxDistR(M_A, M_B) = \min_{i\in[1,2d]} \max_{j\in[1,2(d-1)]} MinMaxDist(v_{ij}(M_A), M_B) \quad (4)$$

In the above equation, $MinMaxDist$ is defined the same as in [30]. In analogy to $MinMaxDist$, $MinMaxDistR$ upper bounds the distances between the points in M_A and M_B. Motivated by this, we can precompute the MBR of the kernel for each fuzzy object and store it in the leaf node. In the case that the candidates have no kernel, we replace it with the MBR of the points with the highest probabilities (if there this point is unique, the MBR degenerates to the point). Then, given a fuzzy object A and a query object Q, their α-distance is upper bounded by:

$$d_\alpha^+(A, Q) = \begin{cases} MinMaxDistR(M_A, M_Q), & \text{if } \alpha \le \mu_A \\ +\infty, & \text{otherwise} \end{cases} \quad (5)$$

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

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

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

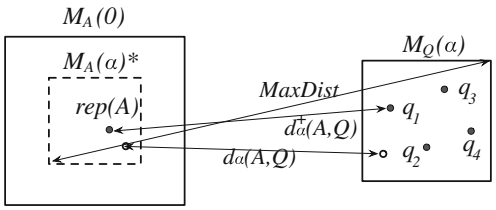


Fig. 7 Sampling-based improvement of upper bound

$$d_\alpha^+(A, Q) = \begin{cases} \min_{q\in Q'_\alpha} ||\text{rep}(A), q||, & \text{if } \alpha \le \mu_A \\ +\infty, & \text{otherwise} \end{cases} \quad (6)$$

The effect of the sampling-based method is shown in Fig. 7, where q_1, q_2, q_3, q_4 are the sampled points from Q_α.

4 Continuous threshold query processing

The single threshold queries restrict the user to examine the query results on a single probability threshold. But sometimes, they may want to see how the query results vary with different thresholds. In such cases, they can issue a continuous threshold query where a probability threshold interval is accepted.

To process a CTQ, the most straightforward approach is to issue a STQ query at every probability within the interval. Despite the interval is continuous, we can always discretize it by enumerating all values in $U_\mathcal{D}$, which is the universe of all the membership values of the whole object set, since the α-distances of all objects with respect to Q do not change at $\alpha \notin U_\mathcal{D}$. However, the cardinality of $U_\mathcal{D}$ is usually huge even in the dataset with moderate size, making this method prohibitive. We call this method *naive approach*.

In the sequel, we will first introduce the basic methods for CTR and CTN queries, which are more efficient than the naive approach. Afterward, we develop several optimization techniques to further improve the performance.

4.1 Basic algorithms

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

Definition 9 Given a fuzzy object A and a query object Q, 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 Fig. 8. The semantics of its each element is that the α-distance is about to change (increase) beyond this probability.

Basic CTR algorithm. First, we have the following lemma.

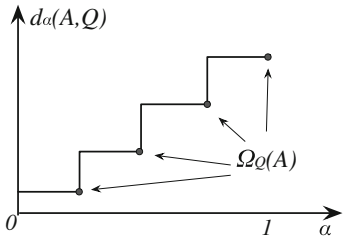


Fig. 8 Critical probability set

Lemma 1 *Suppose object A is a result of a STR query 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 result set within the interval $[\alpha, \alpha']$.*

Proof By definition, the α-distance between A and Q does not change within the interval $[\alpha, \alpha']$. Hence A is guaranteed to remain in the result set within this interval. □

Based on Lemma 1, we propose a basic CTR algorithm as follows. Given an CTR query with $[\alpha_s, \alpha_e]$ as the threshold interval, we issue an STR query at α_s and find the result set, denoted as R_{α_s}. For each object $A \in R_\alpha$, we find the next smallest critical probability after α_s, and choose their minimum value, denoted by α'. According to Lemma 1, R_α is valid within the entire interval since the distance of all objects in R_α do not change. Then, another STR query is issued again to find the new result set at probability threshold of $\alpha' + \epsilon$, 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.

Basic CTN algorithm. Similarly, the following lemma also holds for CTN queries.

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 interval $[\alpha, \alpha']$.*

Proof By definition, the α-distance between A and Q does not change within the interval $[\alpha, \alpha']$. 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 interval. □

Motivated by Lemma 2, we can design a basic CTN algorithm as follows. Given an CTN query with $[\alpha_s, \alpha_e]$ as the threshold interval, we issue an STN 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 choose their minimum value, denoted by α'. According to Lemma 2, the distance of all objects in NN_α will not change within the interval $[\alpha_s, \alpha']$. Then, another STN 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.

Algorithm 2: Basic CTQ Algorithm

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

```
1 α ← α_s;
2 while α ≤ α_e do
3     R_α ← STQ at threshold α;
4     for each A ∈ R_α do
5         β_A ← min_{α'∈Ω_A}{α' ≥ α} ;
6     α* ← min_{A∈R_α} β_A;
7     add ⟨R_α, [α, α*]⟩ into R;
8     α ← α* + ε;
9 return R
```

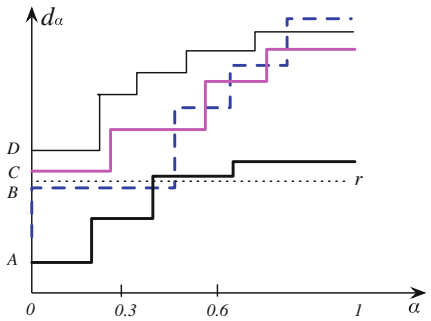


Fig. 9 Reducing search space for CTR query

The structures of CTR and CTN algorithms are very similar, and can be summarized by Algorithm 2. 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 to issue large number of single threshold queries 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.

CTR query

Lemma 3 *Given a CTR query $(Q, r, [\alpha_s, \alpha_e])$, if the α-distance of a fuzzy object A with Q is greater than r when $\alpha = \alpha_s$, then A cannot be a result of this CTR query.*

Proof Due to the monotonicity of α-distance, if $d_{\alpha_s}(A, Q) > r$, then for any $\alpha \in [\alpha_s, \alpha_e]$, we have $d_\alpha(A, Q) > r$. So A cannot be the query result in the entire interval. □

Consider a CTR query with radius r and $I = [0.3, 0.6]$ illustrated in Fig. 9. According to Lemma 3, objects C and

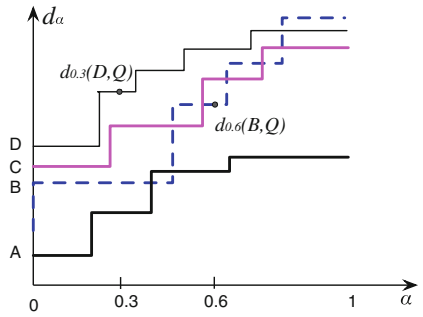


Fig. 10 Reducing search space for CTN query

D cannot be the query results since their distance with Q is greater than r when $\alpha = 0.3$.

Based upon Lemma 3, we first issue a STR query with the same radius and $\alpha = \alpha_s$ against the R-tree to find the candidate set, which includes the objects likely to be the results. Then, we do the refinement by evaluating the α-distances at the critical probabilities of this candidate set instead of the whole dataset. Compared to the whole dataset, the candidate set is usually much smaller and can be loaded into main memory, thus saving considerable amount of I/O costs.

CTN query

Lemma 4 *Given a CTN query* $(Q, k, [\alpha_s, \alpha_e])$, *$B$ is the kth nearest neighbor at α_e, then object A cannot be a result of this CTN 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

$$\begin{aligned} d_\alpha(P, Q) &\le d_{\alpha_e}(P, Q) \le d_{\alpha_e}(B, Q) \\ &< d_{\alpha_s}(A, Q) \le d_\alpha(A, Q) \end{aligned} \tag{7}$$

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 CTN query with $k = 2$ and $I = [0.3, 0.6]$ is issued against the four objects in Fig. 10. According to Lemma 4, object D cannot belong to the result set within the range I, since B is the 2nd nearest neighbor at $\alpha = 0.6$ and $d_{0.3}(D, Q) > d_{0.6}(B, Q)$.

Algorithm 3 shows the new search strategy optimized by Lemma 4. Specifically, given an CTN query with $I = [\alpha_s, \alpha_e]$, we first find the k nearest neighbor set by issuing an STN query with $\alpha = \alpha_e$. Then, we use the kth 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.

Algorithm 3: Reducing Search Space for CTN query

Input: $root$, Q, k, $[\alpha_s, \alpha_e]$
Output: NN: the result set
1 $NN_{\alpha_e} \leftarrow$ STN query at α_e;
2 $r \leftarrow$ the kth nearest neighbor distance at α_e from NN_{α_e};
3 candidate set $\mathcal{C} \leftarrow$ STR query with radius r and $\alpha = \alpha_s$;
4 $NN \leftarrow$ refine candidate set $\mathcal{C}$ using the method of basic STN algorithm;
5 **return** NN

4.3 Improving candidate refinement

With the help of the optimization introduced in the last subsection, 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.

CTR query

Lemma 5 *Given a CTR query* $(Q, r, [\alpha_s, \alpha_e])$, *if the α-distance of a fuzzy object A with Q is greater than r at α', $\alpha' \in [\alpha_s, \alpha_e]$, then A cannot belong to the results of this CTR query within the interval* $[\alpha', \alpha_e]$.

Proof Due to the monotonicity of α-distance, if $d_{\alpha'}(A, Q) > r$, then for any $\alpha^* \in [\alpha', \alpha_e]$, we have $d_{\alpha^*}(A, Q) > r$. So A cannot be the query result in the interval $[\alpha', \alpha_e]$. □

Motivated by Lemma 5, in the refinement process whenever a critical probability threshold α', which makes some candidate A beyond the query range, is encountered, there is no need to refine A further. Instead, we can remove A from candidate set as well as its critical probability thresholds, and report $\langle A, [\alpha_s, \alpha']\rangle$ as a result.

CTN query

Lemma 6 *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* $d_{\alpha'}(A, Q) < d_\alpha(B, Q)$.

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, $d_\alpha(P, Q) \ge d_\alpha(B, Q)$. Then, for any probability $\beta \in [\alpha, \alpha']$, we have

$$\begin{aligned} d_\beta(A, Q) &\le d_{\alpha'}(A, Q) < d_\alpha(B, Q) \\ &\le d_\alpha(P, Q) \le d_\beta(P, Q) \end{aligned} \tag{8}$$

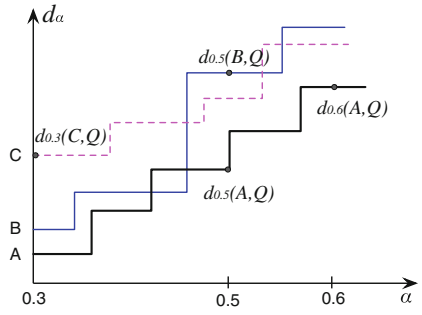


Fig. 11 Improving candidate refinement

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 $k-1$ 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 6, 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 11 illustrates how Lemma 6 can help improve the candidate refinement. Consider an CTN query with $k = 2$ and $I = [0.3, 0.6]$. Since object C is the 3rd nearest neighbor at $\alpha = 0.3$ and $d_{0.5}(A, Q) < d_{0.3}(C, Q)$, by Lemma 6 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 3rd 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 $[0.3, 0.6]$.

Motivated by Lemma 6, 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 CTN algorithm to improve candidate refinement, which is shown in Algorithm 4. 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 the elements $\alpha_1 < \alpha_2 < \cdots < \alpha_n$ 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 CTN algorithm, since it removes lots of critical probability values during the refinement process, which should have been checked by the basic method.

Algorithm 4: Improving candidate refinement for CTN query

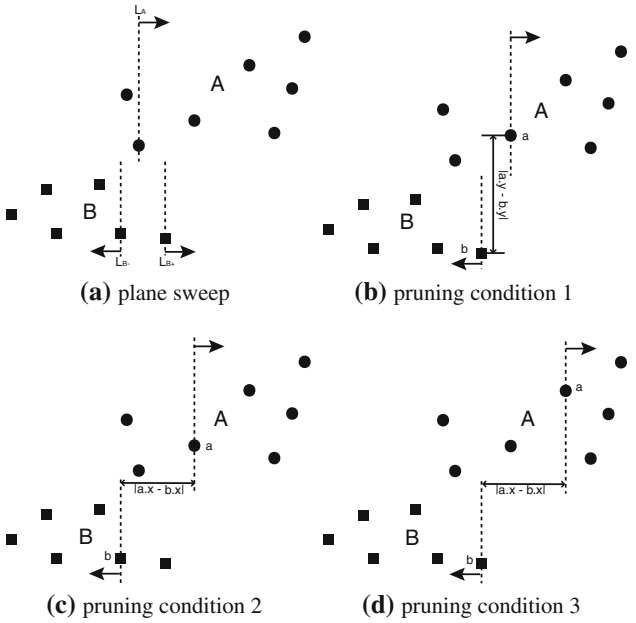
Input: $\mathcal{C}$, Q, k, $[\alpha_s, \alpha_e]$
Output: NN: the result set

```
1  α ← α_s;
2  initialize C' to be empty;
3  while α ≤ α_e do
4      C' ← find the objects in NN which qualify at α;
5      NN_α ← search C\C' for the k − |C'| nearest neighbors at α;
6      d_{k+1} ← the k + 1-th nearest neighbor distance at α;
7      for each A ∈ NN_α do
8          β ← max_{α∈Ω_Q(A)}{d_α(A, Q) < d_{k+1}}
9          add ⟨A, [α, β]⟩ into NN;
10         Ω_Q(A) ← Ω_Q(A)\{α ∈ Ω_Q(A)|d_α(A, Q) < d_{k+1}};
11     α* ← min_{A∈NN_α∪C'} Ω_Q(A);
12     α ← α* + ϵ;
13 return NN
```

5 α-Distance evaluation

Essentially, obtaining the α-Distance is to find the closest pair of qualified points of two fuzzy objects. This actually is to solve the *bichromatic closest pair* (BCP) problem which is the generalization of the *closest pair* (CP) problem. CP problem is well studied in many algorithm textbooks [12,29] and published literature [2,13,14,19,22,37]. One of the most influential work is [13], which utilized two R-tree, one for each set of points, to prune some points which are unlikely to be the closest and speed up the search. According to their experimental study, however, their algorithms are quite sensitive to the degree of overlap of two sets of points. More specifically, the performance gets worse when the portion of overlap increases. This phenomenon can be explained by two extreme cases: (a) when two sets of points are well separated, i.e., their overlap is zero, algorithms will only search the branches of R-tree which are near to each other, and go deep to the leaf nodes quickly since all other branches are pruned by the *MINDIST* metric; (b) if two sets of points fully overlap with each other, almost all branches need to be examined because the *MINDIST* of most pair of MBRs are very small thus cannot be pruned. As discussed before, after the filtering strategy in KNN search, most candidates left for verification are very close to or highly overlap with query object, which makes it not ideal to adopt the algorithm in [13]. Hence, in the context of our problem, an appropriate α-Distance computation algorithm should have better performance when two objects are close to or overlap with each other than when they are far away.

The brute-force way to get the α-Distance of two fuzzy objects is to use a nested loop to check every pair of points and find the minimum distance. The advantage of this method is the simplicity of structure and stability in running time regardless of any variation of fuzzy objects, in terms of dif-



(a) plane sweep **(b)** pruning condition 1

(c) pruning condition 2 **(d)** pruning condition 3

Fig. 12 Illustration of Plane Sweep algorithm

ferent size, probability distribution, relative positions, and so on. However, this method can be very costly when the number of points in fuzzy objects goes large, since it enumerates all pairs of points completely. In the experiment part, this method is used as the baseline approach for comparison. In the sequel, we will illustrated the ideas and techniques in two-dimensional space. However, all the proposed algorithms can be extended to higher dimensionality in straightforward manner.

5.1 Plane sweep algorithm

One of the key techniques in computational geometry is the *plane sweep algorithm* which is a type of algorithm that uses a conceptual sweep line or sweep surface to solve various problems in Euclidean space. [18] applied plane sweep algorithm to find the closest pair in a set of points. A critical observation made in their algorithm is that as the sweep line passing through a point, there are at most constant number of points need to be checked. But this property does not exist in our case which essentially is the BCP problem, because the number of points with monochromatic color cannot be bounded. Hence in this section, we propose a new plane sweep method which utilizes two sweep line to facilitate the search. Figure 12 illustrates the basic idea of our scheme, in which points of two fuzzy objects A, B are represented by circle and square, respectively. When algorithm starts, we first initialize a line L_A which horizontally sweeps A's points from left to right. Whenever L_A encounters a new point a which $A(a) \geq \alpha$, another two lines L_{B-} and L_{B+} originate from L_A and start to scan B's points in the left and right directions. Similar to the nested loop method, if the probability of point b being scanned by L_{B-} (L_{B+}) is higher than α, Euclidean distance will be calculated and update d_{min} when necessary. The major advantage of Plane Sweep is that the speed of search process can be improved by several fast checking of *Pruning Conditions*.

- *Pruning condition 1*. Whenever the distance of a, b along y-dimension is already greater than current d_{min}, b can be safely skipped without knowing the their exact Euclidean distance because $||a - b|| \geq |a.y - b.y|$ (Fig. 12b).
- *Pruning condition 2*. Whenever the distance of a, b along x-dimension is already greater than current d_{min}, sweep of B in this direction can terminate, since any further point b' will have greater distance with a than d_{min} by the in-equation $||a - b'|| \geq |a.x - b'x| \geq |a.x - b.x| > d_{min}$ (Fig. 12c).
- *Pruning condition 3*. If b is the rightmost point being scanned by L_{B-}, and the distance of a, b along x-dimension is greater than current d_{min}, algorithm can terminate since any further point pair (a', b') will have greater distance than d_{min} (Fig. 12d).

The intuition of this method is that it tries to find a relatively close pair as early as possible and use the small distance to prune more other pairs. Note that checking all the above pruning conditions need only minus operations which run much faster than the calculation of Euclidean distance. Besides, even if no condition is satisfied, the intermediate results (difference along some dimensions) can be used for the calculation of Euclidean distance, which causes no extra cost.

Algorithm 5 illustrates this process, for which we just explain two functions:

- *IsPrunable*. There are some cases where no points of A, B can be pruned by the conditions. In this case, we just use nested loop method because it benefits from its brief structure. Function *IsPrunable* checks if it is possible for A, B to meet any pruning condition. We use $||A - B||_{\min}$, $|A - B|^x_{\max}$, $|A - B|^y_{\max}$ to represent the minimum distance, maximum distance along x-dimension and y-dimension between the MBRs of A and B, respectively. All these metrics can be obtained very fast by existing geometry approaches. If the in-equation $||A - B||_{\min} > \max\{|A - B|^x_{\max}, |A - B|^y_{\max}\}$ holds, no points can be pruned by Plane Sweep.
- *CheckRelativePosition*. The relative position of A, B is important to the performance of Plane Sweep algorithm. That our scheme sweeps A from left to right is based on the assumption the leftmost point of A is on the right of that of B. This is because it gets higher chance to find closer

Algorithm 5: Plane Sweep

```
1  initialize d_min ← ∞;
2  if IsPrunable(A,B) then
3      sort the points in A, B according to their x-coordinates;
4      if CheckRelativePosition(A,B) then
5          exchange the roles of A and B;
       /*right sweep of A                                   */
6      for a = a_1 to a_|A| do
7          if A(a) < α then
8              continue;
           /*Pruning Condition 3                            */
9          if a.x − b_|B|.x > d_min then
10             break;
11         else
12             b_m ← arg min_{b.x≥a.x}{b.x − a.x};
               /*right sweep of B                           */
13             for b = b_m to b_|B| do
14                 if B(b) < α then
15                     continue;
                   /*Pruning Condition 2                    */
16                 if |b.x − a.x| > d_min then
17                     Break;
                   /*Pruning Condition 1                    */
18                 else if |b.y − a.y| > d_min then
19                     Continue;
20                 else if ||b − a|| < d_min then
21                     update d_min ← ||b − a||;
               /*left sweep of B                            */
22             for b = b_{m−1} to b_1 do
23                 the same as right sweep process;
24 else
25     use Nested Loop method;
26 return d_min
```

point pair and achieve better pruning effects. Function *CheckRelativePosition* examines this condition by comparing $a_1.x$ and $b_1.x$. If $a_1.x < b_1.x$, the roles of A, B are exchanged before sweep process starts.

5.2 Probability neighbor list

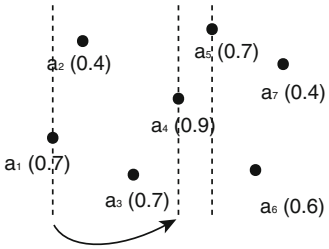
By the definition of α-Distance, only the qualified pairs need to be considered during the procedure. However, Plane Sweep always scans the same set of pairs for the same objects, no matter what α is. This is because all pruning conditions of the algorithm are based only on spatial relationship between objects, but do not take their probability information into account. Motivated by this, we build *Probability Neighbor List (PNL)* for each fuzzy object to further improve the speed of Plane Sweep.

For fuzzy object A, PNL_A maintains a list of entries. The entry corresponding to the point a is a tuple of the form $\langle a.left, a.right\rangle$, where $a.left$ and $a.right$ is called the *left neighbor* and *right neighbor* of point a, which refer to another two points of A:

$$a.left = \arg \min_{a' \in A, a'.x \le a.x} \{a.x - a'.x\}$$

$$a.right = \arg \min_{a' \in A, a'.x \ge a.x} \{a'.x - a.x\}$$

Simply speaking, $a.left$ ($a.right$) is the nearest neighbor on the left (right) side of a among those points whose probability is greater than that of a.



(a)

$a.left$	a	$a.right$
$null$	a_1	a_4
a_1	a_2	a_3
$null$	a_3	a_4
$null$	a_4	$null$
a_4	a_5	$null$
a_5	a_6	$null$
a_6	a_7	$null$

(b)

Fig. 13 Example of PNL

With the help of PNL, whenever the sweeping line encounters a point a with $A(a) < \alpha$, it checks $a.left$ or $a.right$ according to its sweeping direction and "jump" to that point instead of moving to the next point sequentially. By this means, a large number of unqualified points can be pruned without even checking the positions.

Consider the fuzzy object A in Fig. 13a and its PNL shown in Fig. 13b. Suppose $\alpha = 0.8$ and the sweep line is scanning A from left to right. When a_1 is under scanned, because $A(a_1) = 0.7 < 0.8$, we use $a_1.right$ which refers to a_4 as the next point to be scanned, thus skipping a_2 and a_3. Similarly, when a_5 is swept, we check $a_5.right$ which is $null$. That means no probabilities of points on the right are higher than 0.7, so the sweep process can terminate without looking at the following points.

6 Complexity analysis

In this section, we try to estimate the number of object access during the STR and STN query processing 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 10 An ideal fuzzy object A is circle (or sphere), and the radius of its α-cut is characterized by a function $R : \alpha \to radius$.

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 in the basic STQ algorithms.

For the basic STR algorithm, we can apply the formula given in [27] to estimate the leaf node access of the R-tree by a range query:

$$L = \frac{N-1}{C_{\text{avg}}} \cdot \left(\left(\frac{C_{\text{avg}}}{N} \right)^{1/D_0} + 2d \right)^{D_2} \tag{9}$$

$$C_{\text{avg}} = C_{\max} \cdot U_{\text{avg}}$$

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. However, this formula was proposed for the point object set, which means each fuzzy object is represented by its center. In order to make it suitable for fuzzy object set, we need to enlarge the search radius r by twice the radius of the fuzzy object, i.e., $2R(0)$. By substituting d with the query range, we get

$$L = \frac{N-1}{C_{\text{avg}}} \cdot \left(\left(\frac{C_{\text{avg}}}{N} \right)^{1/D_0} + 2r + 4R(0) \right)^{D_2} \tag{10}$$

For the basic STN query, the objects to be accessed are the ones whose $MinDist$ with Q is smaller than $d_{knn}(\alpha)$, where $d_{knn}(\alpha)$ is the α-distance between Q and its kth nearest neighbor. So we need to estimate $d_{knn}(\alpha)$ first. To this end, we can borrow other formulas from [27] to estimate the average number of neighbors $nb(\epsilon, \text{`}shape\text{'})$ of a point Q within distance ϵ from Q, using the concept of correlation fractal dimension of the point set:

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

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

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

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

$$\epsilon = \frac{1}{\sqrt{\pi}} \sqrt{\frac{k}{N-1}} \tag{11}$$

The ϵ derived from Eq. (11) can be treated as distance from the center of query to the center of its kth 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 we can apply Eq. (9) again. Finally, by substituting d with $d_{knn}(\alpha) + R(\alpha)$, we can estimate the average number of object accessed as the function of α:

$$L = \frac{N-1}{C_{\text{avg}}} \cdot \left(\sqrt{\frac{C_{\text{avg}}}{N}} + 2 \left(\frac{1}{\sqrt{\pi}} \sqrt{\frac{k}{N-1}} - R(\alpha) \right) \right)^2 \tag{12}$$

From Eq. (12) we can see that more objects need to be accessed as N, k or α increases independently.

7 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.

7.1 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 containing n points. We place the points within the circle following two kinds of distributions, namely *uniform* and *Zipf*. For the uniform distribution, the points have equal probability to locate anywhere within the circle. For the Zipf distribution, the points are more dense around the center of the circle and get sparse quickly toward outside. Normally, given a Zipf function $f(k, s, N) = \frac{1/k^s}{\sum_{n=1}^{N}(1/n^s)}$, we can use s to control the skewness of the distribution – greater s means more skewed distribution. The membership probabilities of the points follow the Gaussian distribution with the mean at the center of the circle and stand deviation of 0.5. We normalize the probability values across 0 to 1.
- For the real dataset, each object is formed by n points randomly sampled from the horizontal cell identified by probabilistic segmentation [25,26]. 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

Table 2 Parameter settings

Parameter	Default value
Number of points, n	1,000
Number of objects, N	50,000
Query radius, r	5
Number of results, k	20
Probability threshold, α	0.5
Length of probability interval, L	0.2
Number of samples for improving upper bound, w	100

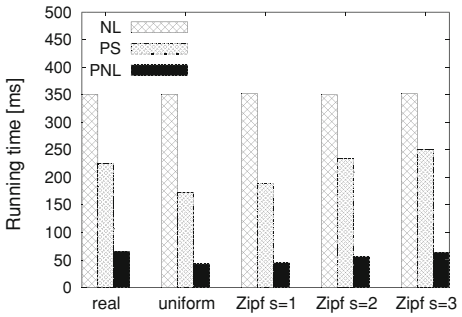


Fig. 14 Effect of point distribution

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.

7.2 Performance of distance computation algorithms

In this subsection, we study the running time of different methods for computing α-distance, namely nested loop (NL), plane sweep (PS), and improved plane sweep (PNL). For each experiment, we randomly choose 1,000 pairs of fuzzy objects from the dataset, evaluate their α-distances, and record the average running time.

7.2.1 Effect of point distribution

First, we study the performance of all the α-distance computation algorithms on different objects. As we can see from Fig. 14 that the NL method has the same runtime cost on all kinds of objects since it only depends on the number of points in each fuzzy object. As expected, the PS method outperforms NL constantly for all the objects. Besides, PS consumes more time on synthetic objects with Zipf distribution than uniform distribution. This is because more points tend to locate around the center of the circle which are assigned higher probabilities in a Zipf distribution with greater s. So more qualifying pairs of points need to be examined by PS. Another observation we made is that PNL can reduce more time cost on synthetic objects. This is due to Gaussian distribution of the probability in synthetic objects that points with lower probabilities always locate further from the kernel. This characteristics greatly benefits the PNL method since a large portion of disqualifying points (with probabilities lower than the probability threshold) can be ruled out directly.

For the rest of experiments, we only show the results on the real objects as the algorithms exhibit the similar performances on other kinds of objects.

7.2.2 Effect of n

Then, we study the scalability of all three α-distance computation algorithms by varying the number of points in each fuzzy objects from 500 to 10,000. According to the results shown in Fig. 15a, though all the algorithms have similar complexity with the number of points, PS and PNL save considerable running time in practice by leveraging several pruning conditions.

7.2.3 Effect of α

Next we investigate the impact of probability threshold by varying α from 0.3 to 0.9. As shown in Fig. 15b, the time cost of all methods decreases as α increases, since there are less qualifying pairs involved in the distance evaluation. We also notice that, when the probability threshold is low, PS

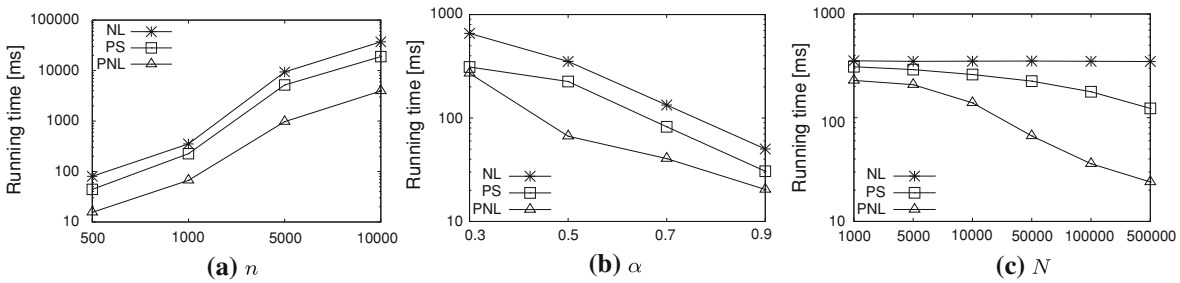


Fig. 15 Performance of α-distance computation algorithms

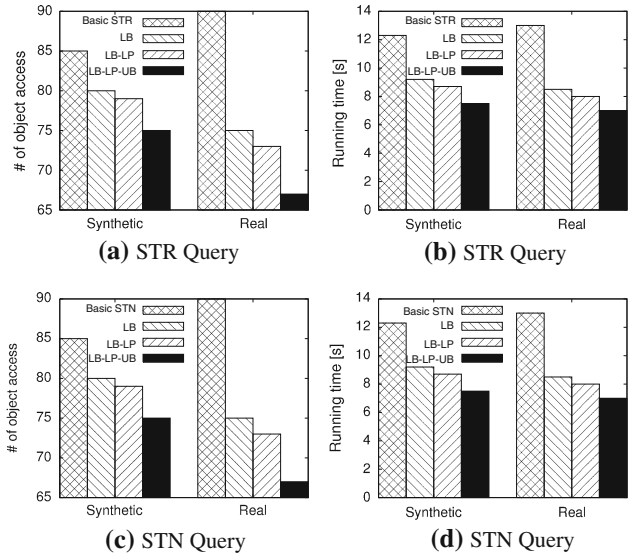


Fig. 16 Effect of dataset

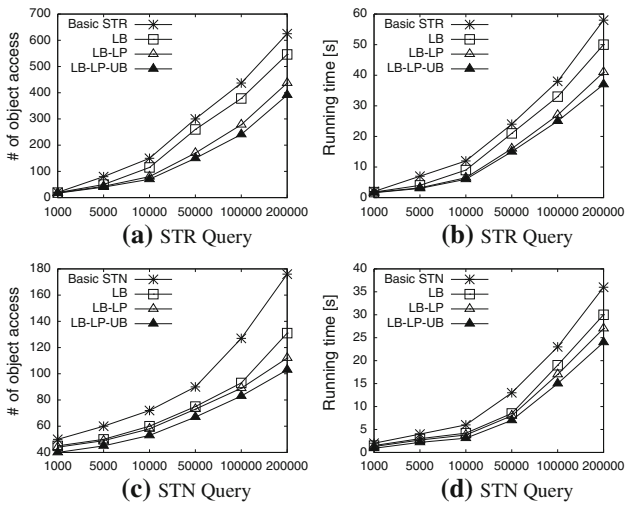


Fig. 17 Effect of N

and PNL have similar performance. The advantage of PNL becomes obvious only when α is beyond 0.5. This is because most points are qualified when α is small and are more difficult to prune by the PNL.

7.2.4 Effect of N

We also present analysis on the relationship between the performance of distance computation algorithms and the size of dataset. As we can see from Fig. 15c, while the basic approach is not affected by this parameter, the other two methods run faster as N increases. Recall that our purpose of designing the plane sweep algorithms as well as its optimization is to improve the efficiency of α-distance evaluation especially when two fuzzy objects heavily overlap with each other. As N increases, the fuzzy object set becomes more dense, making our proposed methods more effective.

7.3 Performance of STQ Algorithms

In this subsection, we compare the performance of the basic STQ algorithm against its competitors, namely STQ algorithm with improved lower bound (LB), LB with lazy probe (LB-LP), and LB-LP with improved upper bound (LB-LP-UB) on an extensive set of parameters. For the improved upper bounding technique (LB-LP-UB), we choose the sampling-based method with 100 samples by default. Then, we investigate the tightness of different upper bounding distances. Lastly, we compare the performance of the proposed lazy probe technique against the multi-step kNN search algorithm proposed in [24].

7.3.1 Effect of dataset

First, we investigate the impact of different kinds of object sets on the performance of STQ processing algorithms. The results are illustrated by Fig. 16. Easy to see that the IO costs of the algorithms on synthetic datasets are lower than that on the real dataset (Fig. 16a, c). This is caused by the fact that synthetic objects usually have more regular and smaller α-cuts (with $\alpha = 0.5$ by default), which makes the approximations of their α-distances more tight. Besides, the LB technique is more effective on Zipf distributions with greater s values than the uniform distribution, since more points are located around the center of the circle making their α-cuts more compact. Another interesting observation is, even though the IO costs on Zipf distribution are lower, their running time is higher than that on uniform distribution (Fig. 16b, d). This is because the evaluation of α-distances for objects with Zipf distribution is more expensive than that of uniform distribution, as we mentioned in the last subsection. For presentation convenience, we only report the results on real dataset for the rest of the experiments, as the trends of algorithms on synthetic dataset are similar.

7.3.2 Effect of N

We study the impact of dataset size on the performance of STR and STN algorithms by varying the number of objects from 1K to 200K. As shown in Fig. 17a and c, all the methods need to access more objects in order to determine the query results with the growth of the dataset. This is due to the fact that, 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 the time cost aspect, as we can observe from Fig. 17b

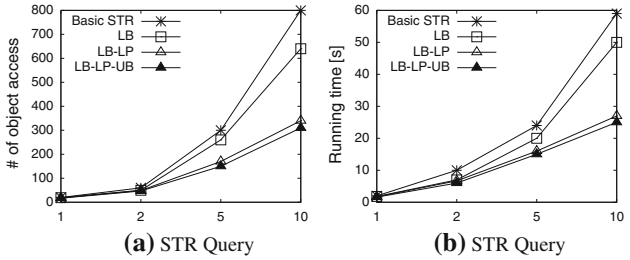


Fig. 18 Effect of r

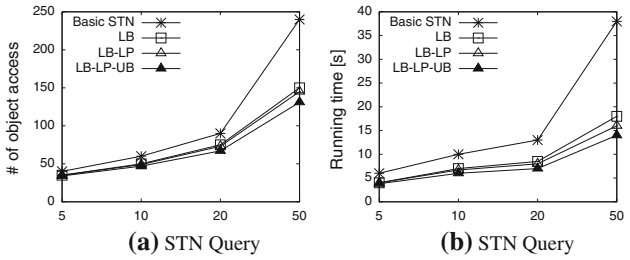


Fig. 19 Effect of k

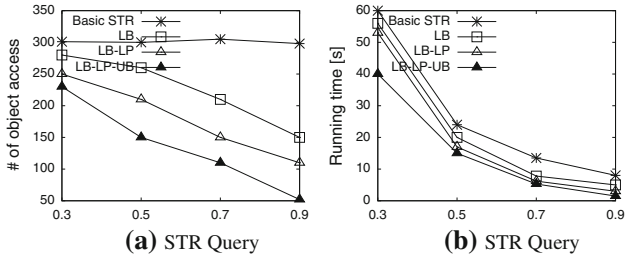
and d, their running time also increases because more IO operation is invoked and more α-distances need to evaluated. We also notice that the performances of all algorithms are still comparable when the dataset is relatively small, since the simple lower bound serves as a tolerable approximation when the data space is sparse. The advantages of the proposed optimizations become more obvious in larger dataset.

7.3.3 Effect of r

We then examine how the performances of STR approaches are affected by the search radius r, the results of which are shown in Fig. 18. As expected, all the algorithms need to access more objects and consume more time when r expands, since more leaf nodes of R-tree are included as candidates and more α-distance needs to be evaluated for refinement. However, compared to the basic approach, the algorithm with all optimizations is much less sensitive to the increases of r.

7.3.4 Effect of k

Next we compare the performance of all STN algorithms by varying k from 5 to 50. According to Fig. 19a and b, the performance of all algorithms deteriorates 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 kth 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.



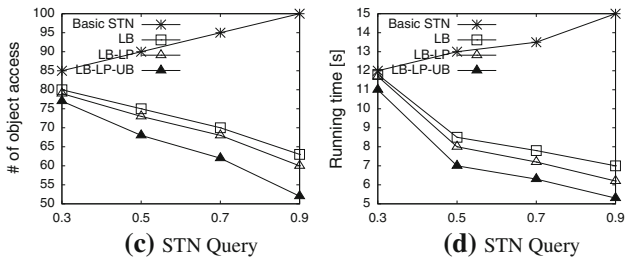


Fig. 20 Effect of α

7.3.5 Effect of α

We also study the impact of the probability threshold by varying α from 0.3 to 0.9. As shown in Fig. 20a, the object access number of the basic STR algorithm is not affected by this parameter since it approximates each fuzzy object by its MBR regardless of the α-cut. As a comparison, the optimized methods perform better as α rises as the distance bounds are tighter. However, the running time of all algorithms decrease with α even for the basic approach, according to Fig. 20b. This is because the α-distance computation algorithm is more efficient at higher probability threshold. On the other hand, the behavior of STN algorithms is slightly different. From Fig. 20c and d, 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 kth nearest neighbor when α is higher. As a consequence, more candidates are retrieved for verification. Although the optimized algorithms face the same situation, their performances 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.

7.3.6 Sensitivity of α

Sometimes it is difficult for a user to choose a desired α when he/she issues the kNN queries. Although we have also proposed the continuous threshold queries to address this issue, it might still be valuable to give some insight on the sensitivity of α, i.e., how significantly the result set will change when α is tuned. To this end, we continuously increase the value of α from 0 to 1 with a step of 0.01, and issue a STQ

for each α. We adopt two similarity measures to evaluate the differences of two result sets on consecutive α. More specifically, we use *Jaccard similarity* for the result sets of STR queries, and *Kendall tau similarity* for the top-k lists of STN queries. Kendall tau distance [16] is a metric that counts the number of pairwise disagreements between two ordered lists. Given two ordered lists τ_1 and τ_2 of size n, their normalized Kendall tau distance is defined as

$$K(\tau_1, \tau_2) = \frac{\sum_{\{i,j\} \in P} \bar{K}_{i,j}(\tau_1, \tau_2)}{n(n-1)/2}$$

where P is the set of unordered pairs of distinct elements in τ_1 and τ_2. $\bar{K} = 0$ or 1 if i and j are in the same or opposite order in the two lists. Kendall tau similarity is defined based on Kendall tau distance as follows:

$$KS(\tau_1, \tau_2) = 1 - K(\tau_1, \tau_2)$$

In this way, the Kendall tau similarity and Jaccard similarity are consistent and both lie in the interval [0, 1].

To ease the interpretation, we show the results on the granularity of 0.1, and each value at some α is the average similarity of the records within $(\alpha - 0.1, \alpha]$. For instance, the result similarity at $\alpha = 0.5$ is calculated by

$$\text{Similarity}(0.5) = \frac{1}{10} \sum_{\alpha=0.4}^{0.49} S(R_\alpha, R_{\alpha+0.01})$$

where R_α is the result set of STQ at α, and S is either Jaccard or Kendall tau similarity measure, depending on the query type. A higher similarity value means the result sets on consecutive α are more consistent. In other words, the result set is not very sensitive to the change of α within this range.

The experimental findings for STR and STN queries are shown in Fig. 21a and b, respectively, in which we also vary the query range r or the required number of results k to study their effects on the sensitivity of α. From Fig. 21a, we can see that the Jaccard similarity is stable when α is very small and undergoes a dramatic decrease when $0.3 \leq \alpha \leq 0.6$. After that the JS value quickly climbs up with the increase of α. The reason for this phenomenon is a real fuzzy object usually consists of a compact set of points with high probabilities and a large and irregular fuzzy area with low probabilities. As such, varying α within ranges of lower values affects the shapes of the α-cuts for fuzzy objects significantly. Consequently, the α-distances with the query object may also alter dramatically, which makes the result set more sensitive to α. However, as α increases to a higher level, the shapes of α-cuts do not change too much since the sets of points with high probabilities are already very compact. Thus, the result set becomes less sensitive to α. The stable JS values at very low α are caused by the fact that the membership probabilities of most points in our dataset are above 0.3. Therefore, altering α within [0, 0.3] has little impact on the result set.

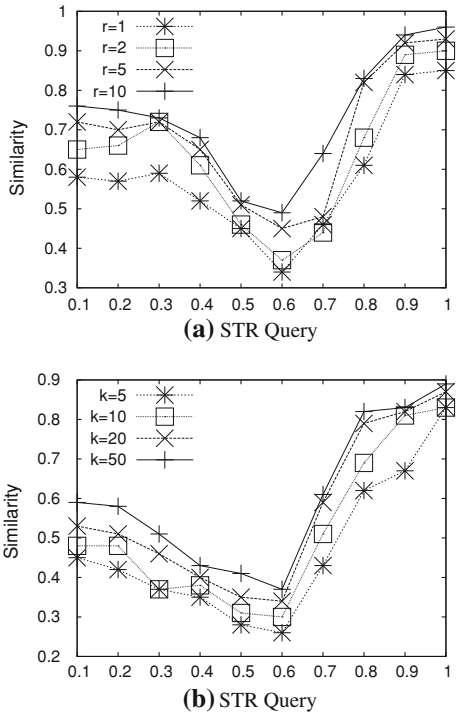


Fig. 21 Sensitivity of α

Besides, with a greater query range r, the overall JS value rises, which means the result sets are less sensitive to α. This is easy to understand, since an object is less unlikely to be excluded from the result set if its α-distance has slight change when the query range is large. Figure 21b shows the similarity results for STN queries by analogous reasons, except that the overall Kendall tau similarity are lower than the STR queries. This implies that the STN queries are more sensitive to α than STR queries.

7.3.7 Memory cost

In this part, we study the memory costs of the proposed indexing structure as well as the STQ processing algorithms. First, the memory cost of index structure (excluding the actual data objects) is shown in Fig. 22a, from which we can see the size of R-tree increases with the cardinality of dataset. The index structure for improved lower bounds (LB) consumes more space than the basic structure since it stores the coefficients of linear functions to approximate the boundaries of α-cuts in each node. The little extra overhead of the index for improved upper bounds is caused by keeping the representative point for each object. We omit the memory cost for LB-LP as it is the same as LB. In addition, we also show the sizes of priority queues for STR and STN query processing in Fig. 22b and c. Consistent with the IO and runtime cost in previous experiments, our proposed optimization schemes with lower

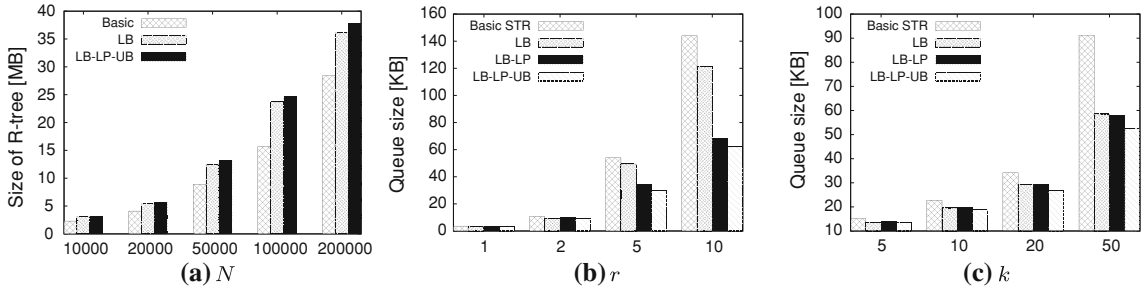


Fig. 22 Memory costs of index and STQ algorithms

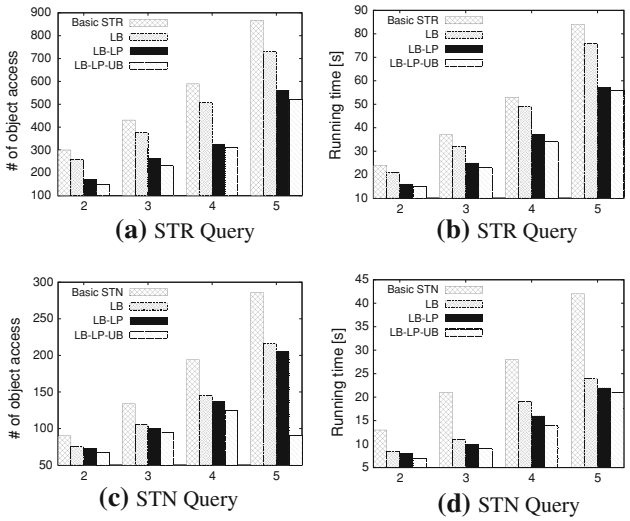


Fig. 23 Effect of dimensionality

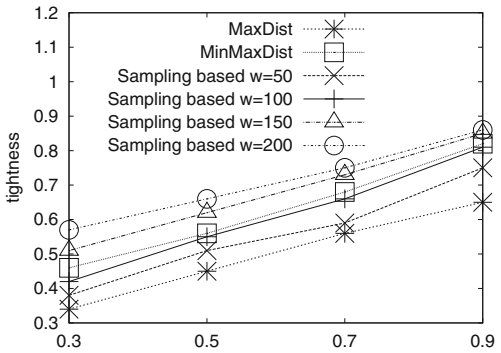


Fig. 24 Tightness of upper bounding distances w.r.t. α

and upper bounding distances can also reduce the queue size by identifying the true drops and true hits in earlier stage.

7.3.8 Effect of dimensionality

We also investigate the performances of the proposed algorithms on higher dimensions. As shown in Fig. 23, the IO and runtime costs of all the algorithms deteriorate with the increasing dimensionality. This is as expected since the overlap ratio of the nodes in the R-tree increases with the dimensionality, which consequently reduces the spatial discrimination of the indexing structure. This also explains the phenomenon that the improved lower bounds can save more IO and running time on higher dimensions. However, it is worth pointing out that, though all our proposed techniques can extend to any multidimensional space, considerable performance deterioration will be experienced on high dimensional spaces (e.g., $d > 10$). This is due to the intrinsic difficulties for R-trees to index objects with such high dimensionality. Though this issue can be tackled to some extent by adapting other indexing structures designed for high dimensional spaces (e.g., X-tree [8]), in-depth study about it is beyond the scope of this paper.

7.3.9 Comparison of upper bounding distances

We compare the tightness of different upper bounding distances proposed in this paper, which is measured by the ratio between the exact α-distance and the upper bounds. Higher values of tightness means a better upper bound. As shown in Fig. 24, all the upper bounds get more tightened with the increasing α since the α-distance tends to increase for a greater α. Not surprisingly, both MinMax distance-based and sampling-based upper bounds constantly improve the MaxDist-based method on the entire range of α. The tightness of sampling-based method rises if more points from the query object are sampled, and eventually exceeds the MinMax-based method when the number of samplings $w \geq 150$. However, we also notice that, with the further increase of samplings, the marginal benefit becomes less obvious. This is due to the fact that, as the α-cut of the query object becomes smaller for greater α, a moderate number of samplings already provides a good approximation for the true α-distance.

7.3.10 Comparison of lazy probe algorithms

Lastly, we compare the performance of the lazy probe algorithm proposed in this paper (referred to as incremental method) against the multi-step kNN search based on lower and upper bounds proposed in [24], by varying k from 5 to

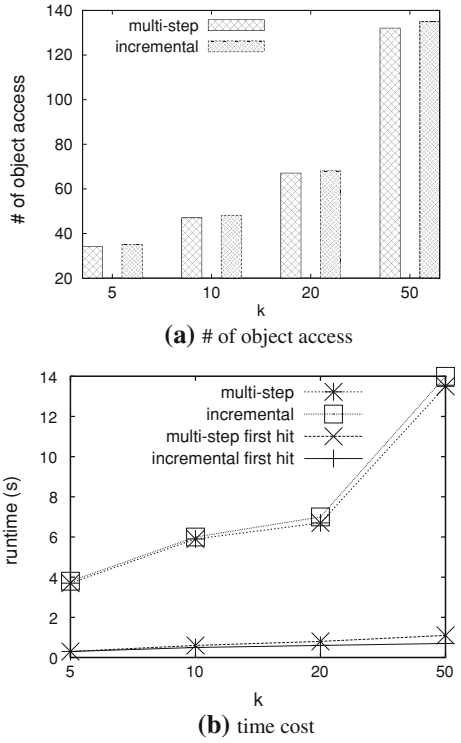


Fig. 25 Comparison of lazy probe algorithms w.r.t. k

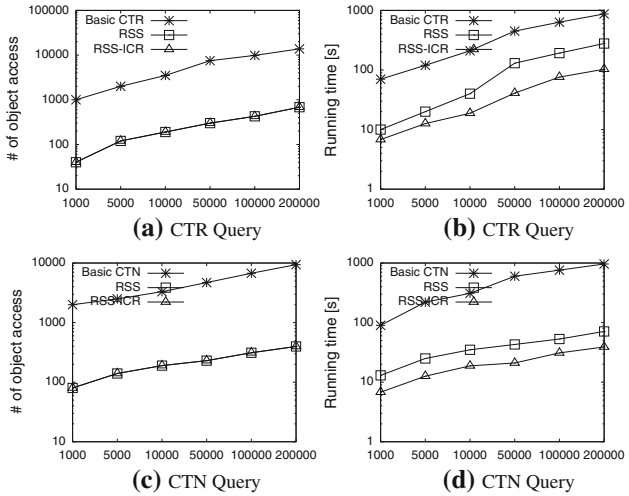


Fig. 26 Effect of N

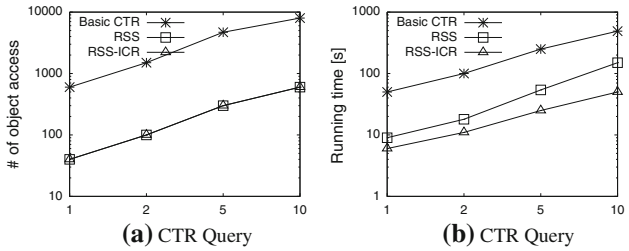


Fig. 27 Effect of r

50. As we can see from Fig. 25, the number of object accesses, which is equivalent to the number of refined objects in [24], are almost the same for both techniques. As we predicate in Sect. 3.3, due to the prompt utilization of the upper bounds of the candidate, the elapsed time when the incremental algorithm confirms the first true hit is less than that of multi-step algorithm, especially for a greater k. This means the incremental algorithms can report parts of the results to the users more quickly. However, the price of this benefit is the slight increase of overall time cost, as proved by Fig. 25b.

7.4 Performance of CTQ Algorithms

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

7.4.1 Effect of N

The first set of experiments tests the scalability of all CTN schemes by varying the dataset size from 1K to 200K. Results are shown in Fig. 26, 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 STQ procedure, it needs to traverse the R-tree many times and incurs high IO and computation cost. With the performance of STQ algorithms getting worse in a larger dataset, the basic CTQ approach deteriorates very faster. On the other hand, RSS only issues an STQ 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 STQ method. 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.

7.4.2 Effect of r

In this part, we investigate the impact of query radius to the efficiency of CTR algorithms. The results are illustrated by Fig. 27, from which we can see that the performance of all approaches deteriorates as r increases since each STR query is less efficient. However, as in the last set of experiments, the optimized schemes constantly outperform the baseline at all radius.

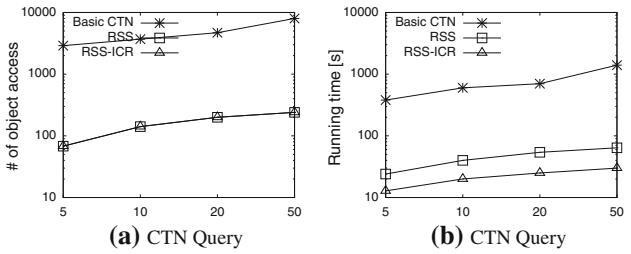


Fig. 28 Effect of k

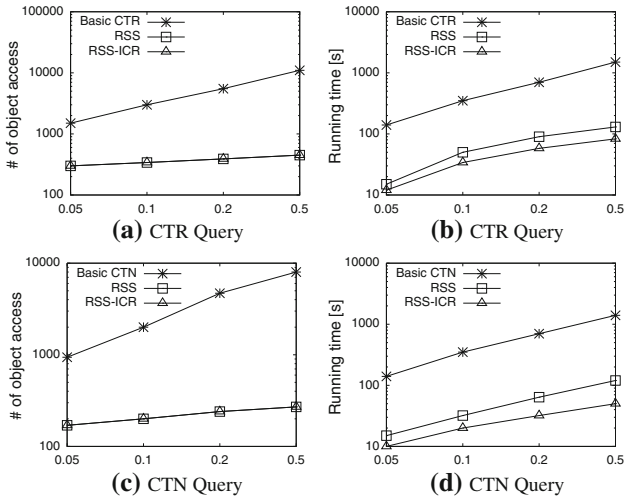


Fig. 29 Effect of L

7.4.3 Effect of k

Then, we study performance trend of all algorithms when different numbers of results are required. Figure 28a and b clearly demonstrates the remarkable advantage of our optimizations, the cause of which is similar with the first set of experiments. For all algorithms, their running time increases are also caused by more critical probability values to be checked as k increases.

7.4.4 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 Fig. 29, with L increasing, the performance of the basic algorithm deteriorates very fast. This is because the number of STQ issued by the basic algorithms 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.

8 Related work

Query processing for traditional objects. The most common and traditional type of object is the static point object, for which spatial query processing algorithms have been long studied. R-trees [6,17] are the most popular indexes for query processing on this kind of objects due to their simplicity and efficiency. The R-tree can be viewed as a multidimensional extension of the B-tree. Points that are close in space are clustered in the same leaf node represented as a minimum bounding rectangle (MBR). Nodes are recursively grouped together following the same principle until the top level, which consists of a single root. R-trees are motivated by the need to efficiently process range query, where the range usually corresponds to a rectangular window or a circular area around the query point. Given a range query, the root is first retrieved and the entries that intersect the range are recursively searched because they may contain qualifying points. A nearest neighbor query retrieves $k (k \geq 1)$ data points closest to the query point q. Numerous algorithms have been proposed based on R-trees. Roussopoulos et al [30] 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 [20] 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 nearest neighbor 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 [23,39,44,45]. But since it is difficult to map a fuzzy object to a single point while keeping the original information, the proposed query processing algorithms for point objects cannot be directly applied to our problem.

Query processing for moving objects. 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 [31] 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 [7] developed query evaluation algorithms for NN and reverse NN search based on TPR-tree. Tao et al [43] optimized the performance of the TPR-tree and extended it to the TPR*-tree. Continuous range and kNN monitoring for moving queries has also been investigated on stationary objects [42] and linearly moving objects [21]. The

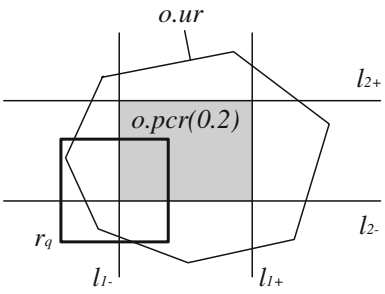


Fig. 30 Illustration of probabilistic constrained region

work in this category shares some features in common with our proposed spatial queries for fuzzy objects. In continuous spatial queries, the result set is only valid within certain time span. Correspondingly, in our work each query result is associated with its qualifying probability interval. 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.

Query processing for uncertain objects. As data uncertainty attracts more research interests, spatial query has also been extended to uncertain databases. Cheng et al. [11] proposed the range and nearest neighbor queries 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 [41] developed the concept of *probabilistic constrained region* (PCR), which can assist pruning a non-qualifying object or validate a qualifying object for a probabilistic range query, without computing the accurate appearance probability. Based on PCR, they further propose U-tree to index multi-dimensional uncertain data with arbitrary probability density functions. A 2D example of PCR is illustrated by Fig. 30, where the polygon represents the uncertain region $o.ur$ of the uncertain object o. Its PCR $o.pcr(p)$ is decided by four lines $l_{1+}, l_{i-}, l_{2+}, l_{2-}$. Line l_{1+} splits $o.ur$ into two parts and the appearance probability of o in the right part equals to p. Similarly, the other lines are obtained in the same way. Now suppose the gray area represents $o.pcr(0.2)$, and there is range query r_q with the probability threshold 0.8. Then, o cannot qualify r_q since the appearance probability of o in r_q must be smaller than that of the left side of l_{1+}, i.e., 1-0.2 = 0.8, where 0.2 is the probability of o falling on the right side of l_{1+}.

Although an uncertain object is often represented by an uncertain region, which looks similar with the fuzzy region in our object model for the fuzzy object, we cannot adapt the similar techniques such as PCR to process the queries for fuzzy objects. To understand this, the probabilities within the uncertain region of an uncertain object sum up to one, since each point inside the region represents a possible instance exclusive with other instances. This is the basic assumption for the PCR as well as U-tree to work correctly. Recall the example in Fig. 30, we can safely conclude the probability of o on the left side of l_{1+} is 0.8 once we know the probability on the other side is 0.2. However, the fuzzy object does not possess this property since each point inside the fuzzy region represents a member of this object whose existence probability is independent of other points. For this reason, we cannot adopt a U-tree like structure to index and support the queries on fuzzy objects in this paper.

Fuzzy objects. On the other hand, the notion of fuzzy objects has not been introduced to database field until Altman [3] adopted fuzzy set theoretic approach for handling imprecision in spatial databases. Afterward, fuzzy data types such as fuzzy points, fuzzy lines, and fuzzy regions were defined in [15,33]. 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 [34], in which only unary functions were considered. On relationship between fuzzy objects, different models of fuzzy topological predicates, which characterize the relative position of two fuzzy objects toward each other, were discussed in [35,36,40]. However, it remains largely untouched to answer more advanced spatial queries such as the range and nearest neighbor queries, which have been addressed in this paper.

9 Conclusion

Although range and nearest neighbor queries are the most classical spatial queries and fuzzy objects can often be found in many important applications, considering both have received rather limited attention. In this paper we have studied this problem in depth, defining new types of range and kNN queries for fuzzy objects, namely single threshold queries and continuous threshold queries. 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.

References

1. Achtert, E., Bohm, C., Kroger, P., Kunath, P., Pryakhin, A., Renz, M.: Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In: Proceedings of SIGMOD, pp. 515–526 (2006)
2. Agarwal, P., Edelsbrunner, H., Schwarzkopf, O., Welzl, E.: Euclidean minimum spanning trees and bichromatic closest pairs. Discret. Comput. Geom. **6**(1), 407–422 (1991)

3. Altman, D.: Fuzzy set theoretic approaches for handling imprecision in spatial analysis. Int. J. Geogr. Inf. Sci. **8**(3), 271–289 (1994)
4. Andrew, A.M.: Another efficient algorithm for convex hulls in two dimensions. Inf. Process. Lett. **9**, 216–219 (1979)
5. Badel, A., Mornon, J., Hazout, S.: Searching for geometric molecular shape complementarity using bidimensional surface profiles. J. Mol. Graph. **10**(4), 205–211 (1992)
6. Beckmann, N., Kriegel, H., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. ACM Sigmod Rec. **19**(2), 322–331 (1990)
7. Benetis, R., Jensen, C., Simonas, G.: Nearest neighbor and reverse nearest neighbor queries for moving objects. In: Proceedings of IDEAS, pp. 44–53 (2002)
8. Berchtold, S., Keim, D., Kriegel, H.: The x-tree: An index structure for high-dimensional data. In: Proceedings of VLDB (1996)
9. Bloch, I.: On fuzzy distances and their use in image processing under imprecision. Pattern Recogn. **32**(11), 1873–1895 (1999)
10. Chaudhuri, B., Rosenfeld, A.: On a metric distance between fuzzy sets. Pattern Recogn. Lett. **17**(11), 1157–1160 (1996)
11. Cheng, R., Kalashnikov, D., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: Proceedings of SIGMOD, pp. 551–562 (2003)
12. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to algorithms. The MIT Press (2001)
13. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Closest pair queries in spatial databases. SIGMOD Rec. **29**(2), 189–200 (2000)
14. Dietzfelbinger, M., Hagerup, T., Katajainen, J., Penttonen, M.: A reliable randomized algorithm for the closest-pair problem. J. Algorithms **25**(1), 19–51 (1997)
15. Dilo, A., Rolf, A., Stein, A.: A system of types and operators for handling vague spatial objects. Int. J. Geogr. Inf. Sci. **21**(4), 397–426 (2007)
16. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top k lists. In: ACM-SIAM Symposium on Discrete algorithms, pp. 28–36 (2003)
17. Guttman, A.: R-trees: a dynamic index structure for spatial searching. ACM Sigmod Rec. **14**(2), 47–57 (1984)
18. Hinrichs, K., Nievergelt, J., Schorn, P.: Plane-sweep solves the closest pair problem elegantly. Inf. Process. Lett. **26**(5), 255–261 (1988)
19. Hjaltason, G., Samet, H.: Incremental distance join algorithms for spatial databases. ACM SIGMOD Rec. **27**(2), 237–248 (1998)
20. Hjaltason, G., Samet, H.: Distance browsing in spatial databases. TODS **24**(2), 265–318 (1999)
21. Iwerks, G., Samet, H., Smith, K.: Continuous k-nearest neighbor queries for continuously moving points with updates. In: Proceedings of VLDB, pp. 512–523 (2003)
22. Khuller, S., Matias, Y.: A simple randomized sieve algorithm for the closest-pair problem. Inf. Comput. **118**(1), 34–37 (1995)
23. Korn, F., Sidiropoulos, N., Faloutsos, C., Siegel, E., Protopapas, Z.: Fast nearest neighbor search in medical image databases. In: Proceedings of VLDB, pp. 215–226 (1996)
24. Kriegel, H., Kröger, P., Kunath, P., Renz, M.: Generalizing the optimality of multi-step k-nearest neighbor query processing. In: Advances in Spatial and Temporal Databases, pp. 75–92 (2007)
25. Ljosa, V., Singh, A.: Probabilistic segmentation and analysis of horizontal cells. In: Proceedings of ICDM, pp. 980–985 (2006)
26. Ljosa, V., Singh, A.: Top-k spatial joins of probabilistic objects. In: Proceedings of ICDE, pp. 566–575 (2008)
27. Papadopoulos, A., Manolopoulos, Y.: Performance of nearest neighbor queries in R-trees. In: Proceedings of ICDT, pp. 394–408 (1997)
28. Peng, S., Urbanc, B., Cruz, L., Hyman, B., Stanley, H.: Neuron recognition by parallel potts segmentation. Natl Acad Sci **100**(7), 3847–3852 (2003)
29. Preparata, F., Shamos, M.: Computational Geometry: An Introduction. Springer, Berlin (1985)
30. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: Proceedings of SIGMOD, pp. 71–79 (1995)
31. Saltenis, S., Jensen, C., Leutenegger, S., Lopez, M.: Indexing the positions of continuously moving objects. ACM Sigmod Rec. **29**(2), 331–342 (2000)
32. Schmitz, C., Grolms, N., Hof, P., Boehringer, R., Glaser, J., Korr, H.: A stereological study using a novel three-dimensional analysis method to estimate the nearest neighbor distance distributions of cells in thick sections. Cereb. Cortex **12**(9), 954–960 (2002)
33. Schneider, M.: Uncertainty management for spatial data in databases: fuzzy spatial data types. In: Proceedings of SSD, pp. 330–354 (1999)
34. Schneider, M.: Metric operations on fuzzy spatial objects in databases. In: Proceedings of ACM GIS, pp. 21–26 (2000)
35. Schneider, M.: A design of topological predicates for complex crisp and fuzzy regions. In: Proceedings of ICCM, pp. 103–116 (2001)
36. Schneider, M.: Fuzzy topological predicates, their properties, and their integration into query languages. In: Proceedings of GIS, pp. 9–14 (2001)
37. Schwarz, C., Smid, M., Snoeyink, J.: An optimal algorithm for the on-line closest-pair problem. Algorithmica **12**(1), 18–29 (1994)
38. Seidl, T., Kriegel, H.: Efficient user-adaptable similarity search in large multimedia databases. In: Proceedings of VLDB, pp. 506–515 (1997)
39. Seidl, T., Kriegel, H.: Optimal multi-step k-nearest neighbor search. In: Proceedings of SIGMOD, pp. 154–165 (1998)
40. Tang, X., Kainz, W.: Analysis of topological relations between fuzzy regions in a general fuzzy topological space. In: Symposium on Geospatial Theory, Processing and Applications (2002)
41. Tao, Y., Cheng, R., Xiao, X., Ngai, W., Kao, B., Prabhakar, S.: Indexing multi-dimensional uncertain data with arbitrary probability density functions. In: Proceedings of VLDB, pp. 922–933 (2005)
42. Tao, Y., Papadias, D., Shen, Q.: Continuous nearest neighbor search. In: Proceedings of VLDB, pp. 287–298 (2002)
43. Tao, Y., Papadias, D., Sun, J.: The tpr*-tree: An optimized spatio-temporal access method for predictive queries. In: Proceedings of VLDB, pp. 790–801 (2003)
44. Weber, R., Schek, H., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proceedings of VLDB, pp. 194–205 (1998)
45. Yu, C., Ooi, B., Tan, K., Jagadish, H.: Indexing the distance: an efficient method to kNN processing. In: Proceedings of VLDB, pp. 421–430 (2001)
46. Zadeh, L.: Fuzzy sets. Inf. Control **8**, 338–353 (1965)
47. Zheng, K., Fung, P., Zhou, X.: K-nearest neighbor search for fuzzy objects. In: Proceedings of SIGMOD, pp. 699–710 (2010)