Knowl Inf Syst
DOI 10.1007/s10115-012-0527-4

REGULAR PAPER

A new approach for maximizing bichromatic reverse nearest neighbor search

Yubao Liu · Raymond Chi-Wing Wong · Ke Wang · Zhijie Li · Cheng Chen · Zhitong Chen

Received: 16 September 2011 / Revised: 30 April 2012 / Accepted: 14 July 2012

Abstract Maximizing bichromatic reverse nearest neighbor (MaxBRNN) is a variant of bichromatic reverse nearest neighbor (BRNN). The purpose of the MaxBRNN problem is to find an optimal region that maximizes the size of BRNNs. This problem has lots of real applications such as location planning and profile-based marketing. The best-known algorithm for the MaxBRNN problem is called *MaxOverlap*. In this paper, we study the MaxBRNN problem and propose a new approach called *MaxSegment* for a two-dimensional space when the L_2-norm is used. Then, we extend our algorithm to other variations of the MaxBRNN problem such as the MaxBRNN problem with other metric spaces, and a three-dimensional space. Finally, we conducted experiments on real and synthetic datasets to compare our proposed algorithm with existing algorithms. The experimental results verify the efficiency of our proposed approach.

Keywords Spatial data search · Reverse nearest neighbor · Bichromatic reverse nearest neighbor

1 Introduction

Nearest neighbor (NN) search [18] finds the data points in the data space that are nearer to a given query point than any other points in the data space. *Reverse nearest neighbor* (RNN) search finds the points that have the query point as their nearest neighbor. RNN

Y. Liu (✉) · Z. Li · C. Chen · Z. Chen
Department of Computer Science, Sun Yat-Sen University, Guangzhou, China
e-mail: liuyubao@mail.sysu.edu.cn

R. C.-W. Wong
Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China

K. Wang
Department of Computer Science, Simon Fraser University, Burnaby, BC, Canada

Published online: 23 August 2012

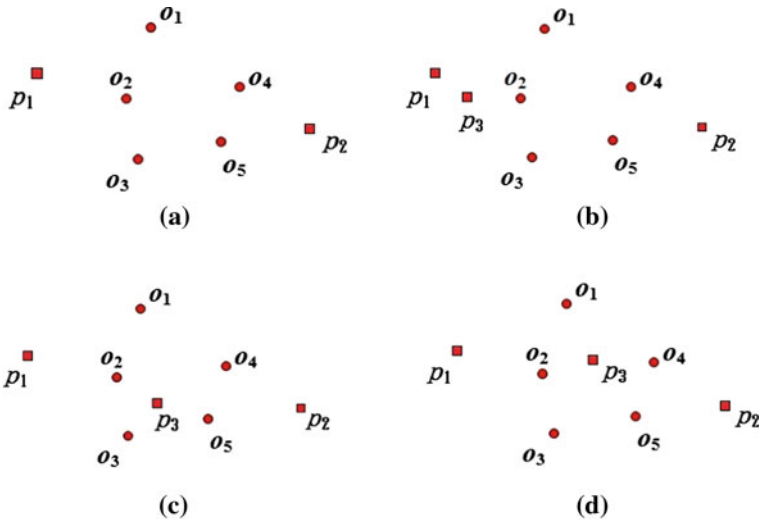


Fig. 1 An example of BRNN

search was presented by Korn et al. [13,14] and has been extensively studied in the database community. There are two kinds of RNN search [13], namely, monochromatic RNN (MRNN) and bichromatic RNN (BRNN). In the case of MRNN, all points are of the same type. A point o is considered as a reverse nearest neighbor for a query point p if there does not exist another data object o' where $|o, o'| < |o, p|$ ($|.|$ denotes the distance). In the case of BRNN, there are two distinct types of point sets $\mathcal{O}$ and $\mathcal{P}$. A point $o \in \mathcal{O}$ is considered as a reverse nearest neighbor for a point $p \in \mathcal{P}$, if there does not exist another point $p' \in \mathcal{P}$, such that $|o, p'| < |o, p|$. The set of all possible points in $\mathcal{O}$ each of which is a reverse nearest neighbor for a point $p \in \mathcal{P}$ is denoted by $BRNN(p, \mathcal{P})$.

Assume that point sets $\mathcal{O}$ and $\mathcal{P}$ correspond to a set of customers and a set of convenience stores, respectively. Assume that the customers would be more interested in visiting a convenience store based on their distances. Figure 1a shows the spatial positions of two stores, p_1 and p_2, and five customers, o_1, o_2, o_3, o_4, and o_5. Then, we have $BRNN(p_1, \mathcal{P}) = \{o_1, o_2, o_3\}$ and $BRNN(p_2, \mathcal{P}) = \{o_4, o_5\}$.

Assume that we want to build a new convenience store p_3. How can we determine the location of convenience store p_3? Intuitively, p_3 can be set up at different positions as shown in Fig. 1b, c, d. In Fig. 1b, we have $BRNN(p_3, \mathcal{P}) = \{o_1, o_2, o_3\}$, in Fig. 1c, we have $BRNN(p_3, \mathcal{P}) = \{o_1, o_2, o_3, o_4, o_5\}$, and in Fig. 1d, we have $BRNN(p_3, \mathcal{P}) = \{o_1, o_2, o_3, o_4, o_5\}$. The largest size of BRNN of p_3 means that we can attract the largest number of customers since we assume the customers would visit a convenience store based on their distances. So the positions of p_3 in Fig. 1c, d are competitive. These two positions are some specific points/positions in the space. In general, we can find a *region* instead of some specific *points*. Finding a region for building a new convenience store can be formulated as a problem called *maximizing bichromatic reverse nearest neighbor* (MaxBRNN) [26]. In this MaxBRNN problem, we assume that all points in both sets $\mathcal{O}$ and $\mathcal{P}$ have a specific location in a Euclidean space. If a new point p is added to $\mathcal{P}$, the MaxBRNN problem [26,27] is to

find the *maximal region* R such that the size of BRNN of p is the largest when p is placed in R.

The MaxBRNN problem is a variant of BRNN search. A large number of applications that exist in BRNN search can also be applied to MaxBRNN search. For example, location planning and profile-based marketing [33,40] are two traditional examples. The example in Fig. 1c can be viewed as location planning application in which a new convenience store can be viewed as a service that needs to attract as many customers as possible. As shown in [26,27], the MaxBRNN problem can also been utilized into other emergency applications, such as natural disasters, sudden big events, and military applications.

There exist two kinds of solutions for the MaxBRNN problem. One solution is presented in [4]. The time complexity of this solution is exponential in terms of $|\mathcal{O}|$. The other solution is *MaxOverlap* and is presented in [26]. To the best of our knowledge, the *MaxOverlap* algorithm [26] is the best solution for the MaxBRNN problem. The key idea of the *MaxOverlap* algorithm is as follows. *MaxOverlap* finds the optimal region using the NLC. The optimal region can be represented by the intersection of multiple NLCs. *MaxOverlap* is the first polynomial-time algorithm for the MaxBRNN problem. The time complexity of *MaxOverlap* is $O(|\mathcal{O}|log|\mathcal{P}| + m^2|\mathcal{O}| + m|\mathcal{O}|log|\mathcal{O}|)$ where m is an integer and denotes the greatest possible number of intersecting NLCs.

We observe that the running time and the storage cost of the *MaxOverlap* algorithm would become large in some cases. For example, in the experiments, when $|\mathcal{O}|$=180K, $|\mathcal{P}|$=360K, and the value of m is about 2,000, the *MaxOverlap* algorithm would take more than 1 h (about 4,500 s). However, in some emergency applications such as the earthquake in China, we often need fast response for the MaxBRNN search to quickly place the supply/service centers for rescue or relief jobs. On the other hand, in many mobile applications, we often only have limited memory in mobile devices such as iPhone and PDA to run the MaxBRNN search. Motivated by such applications, we aim to achieve more efficient MaxBRNN search that would need smaller execution time and storage space. In this paper, we propose a new approach called *MaxSegment* for the MaxBRNN search. Our proposed approach can not only speed up the MaxBRNN search but also reduce the storage cost of the MaxBRNN search.

Specifically, we propose an efficient algorithm called *MaxSegment* whose time complexity is better than that of *MaxOverlap*. In this paper, we show that the running time complexity of the *MaxSegment* algorithm is $O(|\mathcal{O}|log|\mathcal{P}| + m|\mathcal{O}|log\ m + |\mathcal{O}|log|\mathcal{O}|)$.

The major reason why this algorithm is efficient is that we transform the optimal region search problem in a *two-dimensional* space to the optimal interval search problem in a *one-dimensional* space whose search space is significantly smaller than the search space in the two-dimensional space. After the transformation, we can use a plane sweep-like method to find the optimal interval efficiently. Finally, the optimal interval can be used to find the optimal region in the original two-dimensional space.

Besides, the storage of *MaxSegment* is much smaller than that of *MaxOverlap* because *MaxOverlap* requires to store a bulky *overlap table* that occupies $O(|\mathcal{O}|m)$ space but *MaxSegment* does not. In this paper, we show that the storage cost of the *MaxSegment* algorithm is $O(|R_p| + m)$ where R_p denotes the storage cost of R*-tree [1] for point sets $\mathcal{O}$ and $\mathcal{P}$. The main storage cost of the *MaxSegment* algorithm is to store R*-tree for point sets $\mathcal{O}$ and $\mathcal{P}$.

Our contributions can be summarized as follows.

(1) We propose a novel algorithm called *MaxSegment* for the MaxBRNN problem in a two-dimensional space where the L_2-norm is used. The *MaxSegment* algorithm is more efficient than the *MaxOverlap* algorithm in terms of algorithm running time and storage cost.

(2) We also make some extensions for the *MaxSegment* algorithm. The first extension is to extend our *MaxSegment* algorithm to other MaxBRNN problems. The second extension is to extend the *MaxSegment* algorithm to other metric spaces. The third extension is to extend our *MaxSegment* algorithm to a three-dimensional data space. All of the extended algorithms have a similar algorithmic framework with the basic *MaxSegment* algorithm developed for the original MaxBRNN problem.
(3) We conducted experiments to compare the *MaxSegment* algorithm with the best-known *MaxOverlap* algorithm on real and synthetic datasets. The experimental results show the efficiency of our presented methods.

The rest of this paper is organized as follows. Section 2 reviews the related works. Section 3 gives the problem definition including some basic concepts and existing algorithm analysis. Section 4 describes our proposed algorithm *MaxSegment* in a two-dimensional space when the L_2-norm is used. Section 5 proposes our extended algorithms for some variations of the MaxBRNN problem. Section 6 evaluates the proposed algorithms by comparing with the existing best-known algorithm *MaxOverlap* on real and synthetic datasets. Section 7 concludes this paper with future work.

2 Related work

BRNN search was first proposed in [13] and has been extensively studied in spatial databases. Different from existing studies on BRNN search [15,20,22,32], MaxBRNN is to find an optimal region not just a point. Since an optimal region may contain an infinite number of points, how to represent and find such an optimal region become challenging for the MaxBRNN problem. Similarly, the MaxBRNN problem for the L_2-norm space is studied in [4] in which a solution with exponential time complexity is proposed. An extended version of [4] with similar results appears in [3]. Besides, the algorithm in [9] finds an optimal location instead of an optimal region for the L_1-norm space. The best-known solution for the MaxBRNN problem is the *MaxOverlap* algorithm [26] in terms of running time. In some cases, the *MaxOverlap* algorithm is 100,000 times faster than the algorithm in [4]. Some new results, such as the extension of the *MaxOverlap* algorithm in a three-dimensional space, are proposed in [27], an extended version of [26].

In this paper, based on the *MaxOverlap* algorithm, we propose an improved method called *MaxSegment* for the MaxBRNN problem. Different from the *MaxOverlap* algorithm, which transforms the MaxBRNN problem into a point search problem, the *MaxSegment* algorithm transforms the MaxBRNN problem into an *optimal circle arc search* problem. As shown in the experiments, the *MaxSegment* algorithm is more than 60 times faster than the *MaxOverlap* algorithm in some cases. In particular, in a synthetic dataset where $|\mathcal{O}|$ = 180K and $|\mathcal{P}|$ = 360K, the *MaxSegment* algorithm running time is about 70 s while the *MaxOverlap* algorithm is about 4,500 s. The storage cost for the *MaxSegment* algorithm is also distinctly smaller than the *MaxOverlap* algorithm. In particular, in the same synthetic dataset described above, the ratio of the storage of *MaxOverlap* to the storage of *MaxSegment* is about 3.

As shown in Sect. 5.1, the MaxBRkNN problem, which is a variation of the MaxBRNN problem, considers the k nearest neighbors instead of the nearest neighbors of client points. In the MaxBRkNN problem, we assume that each client point (customer) has the same probability to visit the k nearest server points (convenience store). Recently, the authors in [39] studied a generalized MaxBRkNN problem in which a client point may have different

probabilities to visit different server points and at the same time a server point is assumed to have different target sets of client points.

Similar optimal location search problems were also studied in [5] and [35]. Zhang et al. [35] proposes the min-dist optimal location query that finds a location that minimizes the average distance from each client point to its closest server point when a new site is built at this location. Cardinal and Langerman [5] propose to find a location for a new server site and this location can minimize the maximum distance between this new server site and any client point. Different from these problems, our problem is to find an optimal region instead of a location.

There are other related studies. Yiu et al. [34] studies reverse nearest neighbors in large graphs. In [28], spatial matching considers how to efficiently assign each customer (i.e., client point) to her/his nearest server provider (i.e., server point) that has a capacity corresponding to the maximum number of customers it can serve. Lian and Chen [16] proposes some processing techniques for probabilistic reverse nearest neighbor queries over uncertain data. Kang et al. [11] and Stanoi et al. [19] study reverse nearest neighbor queries over dynamic databases. Tao et al. [23,24] study reverse nearest neighbor search in metric spaces and in arbitrary dimensionality. Xia and Zhang [31], Wu et al. [29,30], Cheema et al. [7,8], Emrich et al. [10] study the monitoring problem for continuous reverse nearest neighbor search. These problems focus on spatial search on different scenes such as graph data, uncertain data, and dynamic data. Different from these works, our problem works on static data. In addition, Zhang and Alhajj [36,37] study the similarity search and the reverse nearest neighbor queries in high-dimensional metric space. In [25,38], the concept of k-reverse nearest neighbor is also used to data clustering. The location-based search services [12,21] are also related to our problem.

3 Problem definition

3.1 Basic concepts

We are given two distinct types of point sets $\mathcal{O}$(client point set) and $\mathcal{P}$(server point set). Each point in both $\mathcal{O}$ and $\mathcal{P}$ has a specific location in a Euclidean space $\mathbb{D}$ (e.g., convenience stores in Fig. 1). Each client point $o \in \mathcal{O}$ is associated with a weight, $w(o)$, which denotes the number of clients at location o. A region is defined as an arbitrary shape in the space $\mathbb{D}$ and can also be viewed as a set of points in the space. We say that a region R *covers* another region R' if each point in region R' appears in region R. Similarly, we say that a region R *covers* a curve/line if each point along the curve/line appears in region R.

Definition 3.1 A region R is said to be *consistent* if the following condition holds: $\forall p, p' \in R, p, p' \notin \mathcal{P}, BRNN(p, \mathcal{P} \cup \{p\}) = BRNN(p', \mathcal{P} \cup \{p'\})$.

Definition 3.2 Given a consistent region R, *the influence value* of R is denoted as $I(R)$ and defined as $I(R) = \sum_{o \in BRNN\_R(R)} w(o)$, where $BRNN\_R(R) = BRNN(p, \mathcal{P} \cup \{p\})$ in which p denotes an arbitrary new server point in R.

Definition 3.3 Given a consistent region R, we say that R is a *maximal consistent region*, if there does not exist another consistent region R' satisfying the following conditions: (1) $R \subset R'$, and (2) $BRNN\_R(R) = BRNN\_R(R')$.

Definition 3.4 Given a set $\mathcal{P}$ of server points and a set $\mathcal{O}$ of client points, the *MaxBRNN problem* is to find the maximal consistent region R such that, if a new server point p is set up in R, the influence value of R is maximized.

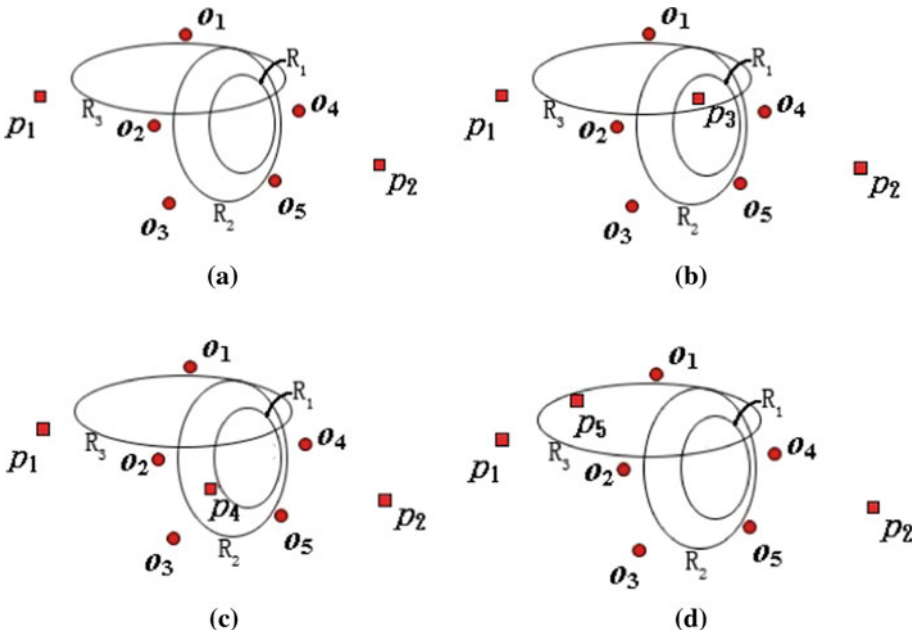


Fig. 2 An example of MaxBRNN problem

In Fig. 2a, R_1, R_2, and R_3 are three different regions. In Fig. 2b, R_1 is a consistent region because any new server point in R_1 such as p_3 has the same BRNN set. Specifically, $BRNN\_R(R_1) = \{o_1, o_2, o_3, o_4, o_5\}$. Similarly, as shown in Fig. 2c, R_2 is a consistent region since any new server point in R_2 such as p_4 has the same BRNN set. Specifically, $BRNN\_R(R_2) = \{o_1, o_2, o_3, o_4, o_5\}$. As shown in Fig. 2b, d, both p_3 and p_5 are in region R_3 and they have different BRNN sets. The BRNN set of p_5 is $\{o_1, o_2, o_3\}$ and the BRNN set of p_3 is $\{o_1, o_2, o_3, o_4, o_5\}$. So R_3 is not a consistent region. In Fig. 2a, since R_1 is inside R_2, R_1 is not a maximal consistent region. If there are no other consistent regions covering R_2, then R_2 would be a maximal consistent region.

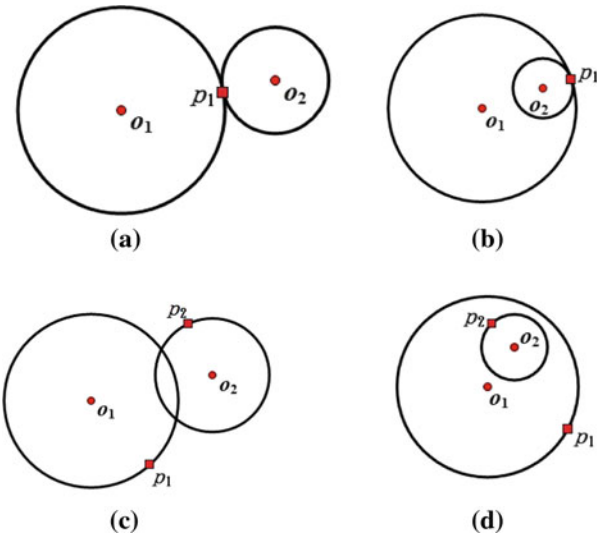
3.2 Existing algorithm analysis

In the literature, there are two solutions for the MaxBRNN problem. One solution has an exponential time complexity in terms of $|\mathcal{O}|$ and was proposed in [4]. The second solution is the best-known solution, the *MaxOverlap* algorithm [26], that is, the first polynomial-time algorithm. Our improved method *MaxSegment* shares some components with the *MaxOverlap* algorithm. So it is necessary to introduce the *MaxOverlap* algorithm together with its basic concepts and important properties. The proofs for these properties are omitted here due to the limit of space.

Definition 3.5 For any client point $o \in \mathcal{O}$, assume p is the nearest neighbor of o in $\mathcal{P}$. The NLC of o is defined as the circle centered at o with radius $|o, p|$.

Property 3.1 [26] *If an NLC covers another NLC, the boundaries of the two NLCs must share at least one point.*

Fig. 3 The relationship of NLCs



In the following, we adopt the convention that a circle centered at o_i is denoted by c_i. The overlapping relationship described in Property 3.1 can be shown in Fig. 3b. Note that according to Property 3.1, it is not possible that the overlapping relationship shown in Fig. 3d appears because the boundaries of the two NLCs in the figure do not share at least one point. The reasoning is described as follows. In this figure, c_1 covers c_2. That is, the area of c_2 is inside the area of c_1, but there are no intersection points between the boundary of c_1 and the boundary of c_2. This relationship of NLCs cannot hold since p_2 on the boundary of c_2 is nearer to the center point o_1 than p_1, which contradicts to the definition of NLC for o_1.

Property 3.2 [26] *For any two overlapping NLCs, say c_1 and c_2, the number of intersection points between the boundary of c_1 and the boundary of c_2 is either one or two.*

This property can be easily illustrated in Fig. 3a, b, c. As we illustrated before, Fig. 3b shows the overlapping relationship that one circle covers another circle. Two other possible overlapping relationships are shown in Figure 3a, c.

Property 3.3 [26] *The optimal region R in the MaxBRNN problem can be represented by an intersection of multiple NLCs.*

According to the above property, we can use NLCs to represent the optimal region. In the following, we focus on describing NLCs.

Consider an example as shown in Fig. 4 containing 6 clients. For the ease of illustration, we remove all servers in the figure and we just show all NLCs. The optimal region (i.e., the shadow part) can be represented by the intersection of three NLCs, namely c_1, c_2, and c_3.

Property 3.4 [26] *Assume that C is a set of NLCs whose intersection corresponds to the optimal region R. If C contains more than one NLC, then there exist two NLCs, say c_1 and c_2, such that region R contains at least one intersection point between the boundaries of c_1 and c_2.*

Property 3.4 tells us that the optimal region must contain at least one *intersection point* between a pair of NLCs. *Suppose* that we know this intersection point and we can find all

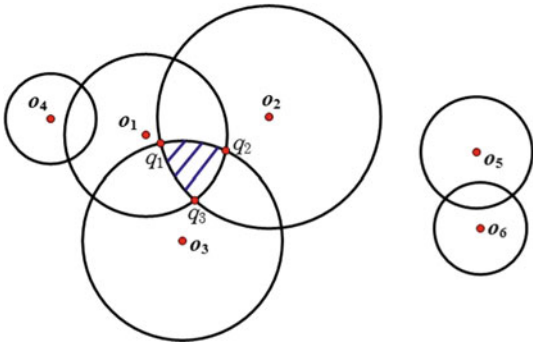


Fig. 4 An example of *MaxOverlap* algorithm

NLCs covering this point. It is easy to verify that the set of all such NLCs corresponds to C. According to this observation, we design the following algorithm because we can regard all possible intersection points between any pair of NLCs as *candidates* for the MaxBRNN search and each candidate can be used for a range query to find all NLCs covering it.

- **Step 1:** find all intersection points between the boundaries of any two overlapping NLCs in the dataset.
- **Step 2:** find a set of NLCs covering each intersection point found in first step.
- **Step 3:** return the set of NLCs with the largest weight value as the final solution.

In the example as shown in Fig. 4 containing 6 clients, the *MaxOverlap* algorithm starts to find a set of intersection points between all pairs of NLCs such as q_1, q_2, and q_3. These intersection points are used to determine the optimal region directly. Suppose that we can find an optimal intersection point q, with the greatest influence value. Let S be a set of NLCs covering point q. The influence value of q is defined to be $\sum_{c \in S} w(c)$. The optimal region of the MaxBRNN problem is equal to region R, which is the intersection of all NLCs in S. In Fig. 4, we can find intersection point q_3 with the largest influence value and the corresponding set of NLCs, $S = \{c_1, c_2, c_3\}$. So, the optimal region corresponds to the intersection of all NLCs in S.

Besides the above three major steps, some pruning techniques are also proposed in the *MaxOverlap* algorithm to reduce the search space of checking all the pairs of overlapping NLCs. In addition, an R*-tree is used to perform a point query, that is, to check whether an intersection point is covered by an NLC.

4 The proposed algorithm

In this section, we propose an efficient algorithm called *MaxSegment* whose time complexity is better than that of *MaxOverlap*. The major reason why this algorithm is efficient is that we transform the optimal region search problem in a *two-dimensional* space to the optimal interval search problem in a *one-dimensional* space whose search space is significantly smaller than the search space in the two-dimensional space. After the transformation, we can use a plane sweep-like method to find the optimal interval efficiently. Finally, the optimal interval can be used to find the optimal region in the original two-dimensional space.

Before introducing the proposed algorithm, we would like to give the preliminaries for our algorithm.

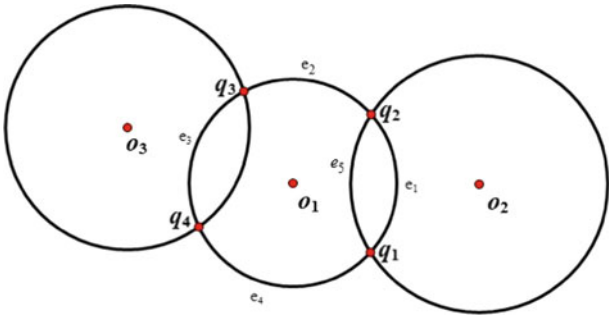


Fig. 5 An example of NLC arc

4.1 Preliminaries

Given two overlapping NLCs, say c_1 and c_2, they form two intersection points, namely q_1 and q_2. Consider the boundary of one NLC, say c_1. These two intersection points divide the boundary of c_1 into two components. The first component corresponds to the boundary of c_1 that is inside c_2 while the second component corresponds to the boundary of c_1 that is not inside c_2. We call each component an *NLC arc* (short for arc). We say that c_1 *owns* the first arc and the second arc. We also say that c_2 *eclipses* the first arc but it does not eclipse the second arc. Note that the two *end points* of an arc correspond to the two intersection points. We give two different names to these two end points according to their positions with respect to NLC c_1. The first one is the *head* of an arc, which is defined to be its end point such that there exists a path from this end point to the other end point in an anticlockwise direction. The second one is the *tail* of an arc, which is defined to be its end point such that there exists a path from this end point to the other end point in a clockwise direction. We say that an arc is *directed* from its head to its tail.

For example, in Fig. 5, we are given 3 clients and their NLCs. In this figure, there are two intersection points between NLCs c_1 and c_2, namely q_1 and q_2, and two intersection points between NLCs c_1 and c_3, namely q_3 and q_4. Consider the boundary of one NLC c_1 that is divided into four arcs according to these four intersection points. Arc e_1 is directed from q_1 to q_2 along the boundary of c_1, denoted by $e_1 = (q_1, q_2)$. q_1 and q_2 are the head and the tail of e_1, respectively. Similarly, arc e_2 is directed from q_2 to q_3 along the boundary of NLC c_1, denoted by $e_2 = (q_2, q_3)$, arc e_3 is directed from q_3 to q_4 along the boundary of c_1, denoted by $e_3 = (q_3, q_4)$, and arc e_4 is directed from q_4 to q_1 along the boundary of c_1, denoted by $e_4 = (q_4, q_1)$.

Consider the Cartesian coordinate system. Assume that NLC c is centered at coordinate (a, b) with radius r. Then, the boundary of c can be expressed as a set of points (x, y) equal to $\{(x, y)|(x-a)^2 + (y-b)^2 = r^2\}$, and the *insider* of c is defined as a set of points (x, y) equal to $\{(x, y)|(x-a)^2 + (y-b)^2 < r^2\}$. We say that a point q is covered by an NLC c if q is on the boundary of NLC c or q is inside NLC c. We say that arc e is covered by NLC c if each point on e is covered by NLC c.

Definition 4.1 Given an arc e, we define the *influence value* of e, denoted by $I(e)$, as the sum of the weights of NLCs that cover arc e.

For example, in Fig. 5, e_1 is covered by c_1 and c_2 only, e_2 is covered by c_1 only, e_3 is covered by c_1 and c_3 only, and e_4 is covered by c_1 only. Assume the weight of each NLC in Fig. 5 is equal to 1. Then, we have $I(e_1) = I(e_3) = 2$, and $I(e_2) = I(e_4) = 1$.

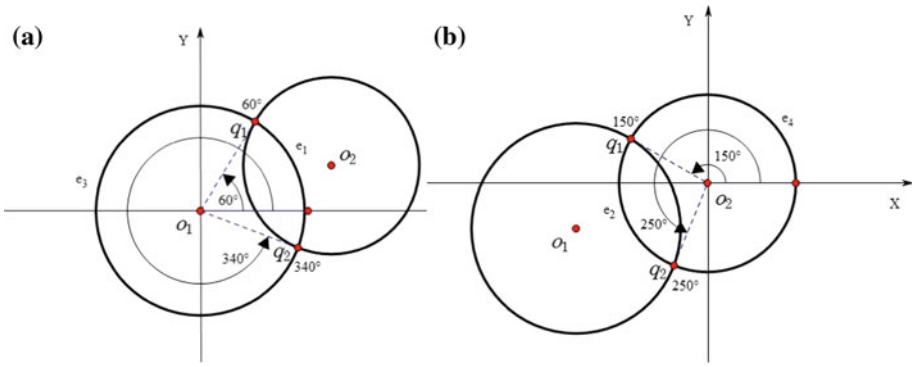


Fig. 6 Representation of arc by angle values

Given two overlapping NLCs, there exists such an arc that is along the boundary of one NLC c_1 and it is also covered by another NLC c_2. We call such an arc as an *intersection NLC arc* (short for an intersection arc). Note that this intersection arc is owned by c_1 and is eclipsed by c_2. For example, in Fig. 5, both arcs e_1 and e_3 are intersection arcs. Arcs e_2 and e_4 are not intersection arcs since both arcs are only along the boundary of c_1.

Given an NLC c and a point x along the boundary of c, we can transform the representation of point x from two real numbers in the Cartesian coordinate system to one real number ranging from 0° to 360° in the polar system whose origin/pole is the center of the NLC. This real number is called the *angle* value of this point. Formally, the angle value of x is defined to be the polar angle of this point when the polar coordinate system is used and the pole is the center of c. Note that the angle value is measured in an anti-clockwise direction from the polar axis defined in the polar coordinate system (i.e., the horizontal axis pointing to the right).

For example, consider the example as shown in Fig. 6a containing two NLCs. Two NLCs intersect and have their intersection points, namely q_1 and q_2. Consider the NLC c_1 and these two points q_1 and q_2 along its boundary. Consider the arc e_1 directed from q_2 to q_1 along the boundary of c_1. The head of intersection arc e_1, q_2, is equal to 340° while the tail of intersection arc e_1, q_1, is equal to 60°.

Note that each arc can also be represented by the mapped angle values of its head and the tail. For example, considering NLC c_1, we represent $e_1 = (340°, 60°)$. For simplicity, a point and its angle value are alternately used below. So 340° and 60° are called the head and tail of e_1, respectively.

In the following, we want to *standardize* the representation of a pair of angle values for each arc. We want to make sure that the angle representing the head of an arc is smaller than or equal to the angle representing the tail of the arc. Given an arc with its angle representation (a_l, a_u), if $a_l \leq a_u$, then we keep this representation. Otherwise, we split this arc into two sub-intersection arcs. In this case, each sub-intersection arc is represented by a pair of angle values. The pair representing a sub-arc is $(a_l, 360°)$, and the pair representing the other sub-arc is $(0°, a_u)$. For example, in Fig. 6a, if we consider NLC c_1, intersection arc e_1 would be split into $e_{11} = (340°, 360°)$ and $e_{12} = (0°, 60°)$.

Similarly, as shown in Fig. 6b, if we consider the other NLC c_2, the head q_1 and the tail q_2 of intersection arc e_2 can be mapped into angle values accordingly. We represent intersection arc $e_2 = (150°, 250°)$ that is along the boundary of NLC c_2. Since $150° \leq 250°$, there is no need to split this arc into two sub-arcs.

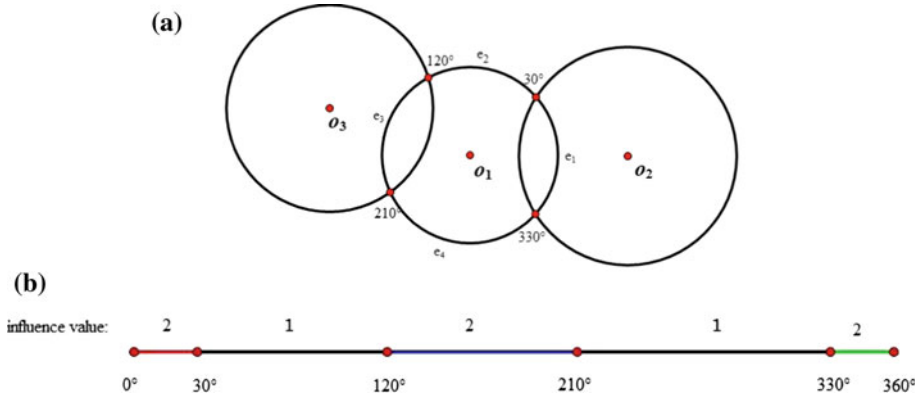


Fig. 7 Computation for influence value

Now, we are ready to describe the major idea of how we use the arc for the MaxBRNN problem. For each NLC c, we *cut* the boundary of c at the position of 0° (or 360°). Then, we stretch the boundary into a *line segment* whose values range from 0° to 360°. The intersection arcs along the boundary of this NLC correspond to different partitions or *intervals* of this line segment.

Note that for a given NLC c, we can transform the boundary of c that is in a two-dimensional space into a line segment that is in a one-dimensional space. In this line segment, we have different partitions or intervals. In the following, we will discuss how we can find the *optimal* interval along this line segment efficiently by a plane sweep-like method. After we find this optimal interval, we can find the corresponding optimal region for the MaxBRNN problem.

We discussed the example in Fig. 5 before where we did not use any angle representation. Figure 7a shows the same example where the angle representation is adopted. On the boundary of NLC c_1, we have intersection arc $e_1 = (330°, 30°)$ and $e_3 = (120°, 210°)$. Since the angle value representing the head of e_1 is larger than that representing the tail of e_1, we would split e_1 into two intersection arcs $e_{11} = (330°, 360°)$ and $e_{12} = (0°, 30°)$. Then, we have three intersection arcs, such as, e_{11}, e_{12}, and e_3, and can construct a line segment as in Fig. 7b. Notice that all intersection points of intersection arcs have been sorted in ascending order of their angle values.

By a *plane sweep*-like method along the line segment, we can compute the influence value of each arc along the boundary of this NLC. In detail, we have the following computation rules for the influence value of each NLC arc.

(1) While scanning, if we meet an intersection point that is a head of an intersection arc, then we increase the influence value with the weight of the NLC eclipsing the arc;
(2) If the intersection point is a tail of an intersection arc, then we shall decrease the influence value with the weight of the NLC eclipsing the arc.

These computation rules are reasonable. If the scanned intersection point is a head of an intersection arc, which means we would enter the region of an intersecting NLC eclipsing the arc, and we need increase the influence value with the weight of this NLC. Otherwise, if the scanned intersection point is a tail of an intersection arc, which means we would leave the region of an NLC eclipsing this arc, and we need decrease the influence value accordingly.

Consider the example as shown in Fig. 7. Assume the weight of each NLC is equal to 1. We scan this line segment owned by c_1 and check each intersection point of all intersection

arcs of NLC c_1. We create a variable called inf for c_1 that will store the influence value of an arc along the boundary of c_1 dynamically. Consider that we are scanning through the line segment for c_1. When we reach an arc or leave an arc, we will update inf accordingly. Details will be described next.

Initially, the influence value, inf, is set as $inf = w(c_1)$. We would first check the intersection point 0° that is the head of e_{12}. The influence value is updated with the weight of the NLC c_2 eclipsing this arc, that is, $inf = inf + w(c_2) = 1 + 1 = 2$, which corresponds to the influence value of NLC arc $e_{12} = (0°, 30°)$. Next, we move to intersection point 30° that is the tail of intersection arc e_{12}. According to the computation rules, we need to reduce the influence value by the weight of NLC c_2, and we would update the influence value of $e_2 = (30°, 120°)$. That is, inf is decreased by $w(c_2)$. That is, $inf = 2 - 1 = 1$. Next, we would meet intersection point 120° that is the head of e_3, and we increase the influence value with the weight of the NLC c_3 eclipsing the arc, and we have inf incremented by $w(c_3)$ and inf becomes 2. Note that $I(e_3) = inf$. Next, we move to intersection point 210° that is the tail of e_3, and we decrease inf by $w(c_3)$. Thus inf becomes 1. Note that $I(e_4) = inf$ where $e_4 = (210°, 330°)$. Finally, we reach intersection point 330° that is the head of e_{11}, and we have $I(e_{11}) = inf + w(c_2) = 2$ where $e_{11} = (330°, 360°)$. The influence value of each arc along the boundary of NLC c_1 is shown in Fig. 7b. The greatest influence value is equal to 2.

As shown in Fig. 7, it is easy to know the following Lemma 4.1 holds.

Lemma 4.1 *Given an NLC and its intersection arcs owned by the NLC, we can find the greatest influence value of an arc by scanning the intersection points of all intersection arcs of this given NLC with the computation rules for influence values of NLC arcs.*

4.2 The algorithm description

The proposed *MaxSegment* algorithm transforms the MaxBRNN problem into an optimal circle arc search problem. Before introducing the algorithm descritption, we would like to introduce why we can transform such problem.

Lemma 4.2 *Let C be a set of NLCs whose intersection corresponds to the optimal region R returned by a MaxBRNN query. Then, R contains at least an optimal arc owned by a certain NLC (i.e., the arc with the greatest influence value).*

Proof There are two cases.

(1) If C contains only one NLC, which means that the optimal solution comes from a single NLC without any overlap or intersection with other NLCs. The optimal region R is the single NLC with the greatest influence value (i.e., the greatest weight) among all NLCs. The boundary of this NLC is the optimal arc contained by the region R. That is, R is this NLC itself.

(2) If C contains more than one NLC. Assume that NLCs are $c_1, c_2, \ldots, c_n$ where $n \geq 2$. It is easy to know that the influence value of R is equal to $\sum_{i=1}^{n} w(c_i)$. Since the boundaries of intersecting NLCs, $c_1, c_2, \ldots, c_n$, form the optimal region R, there exists at least an arc, says arc e, along the boundary of a certain NLC, says c_1. For any point $p \in e$, we have $p \in R$. Then, p is covered by $c_1, c_2, \ldots, c_n$. So, arc e is also covered by $c_1, c_2, \ldots, c_n$. The influence value of arc e is equal to $I(e) = \sum_{i=1}^{n} w(c_i)$. So, arc e is an optimal arc with the greatest influence value and contained in the region R. It is noticed that arc e can be collapsed into a single intersection point when the number of intersection points between two overlapping NLCs is equal to one. □

Similar to Property 3.4, Lemma 4.2 tells that region R contains at least an optimal arc owned by an NLC. We can take such optimal arc as a candidate to perform a MaxBRNN query and transform the MaxBRNN problem into the optimal arc search problem.

Based on Lemma 4.1, we know that we can find the optimal arc with the greatest influence value by checking intersection arcs of all NLCs for a given dataset. The *MaxSegment* algorithm includes three major phases as follows.

Phase 1: Construct NLCs for a given dataset.
Phase 2: Construct all possible intersection arcs for each NLC and find the influence value of each arc. In particular, for each NLC c, we do the following four steps.

Step 1: Find all the other NLCs intersected with c.
Step 2: Compute all intersection points between c and each of the other NLCs intersected with c and construct intersection arcs along the boundary of c.
Step 3: Sort intersection points (i.e., head or tail) of all intersection arcs of c according to the angle values.
Step 4: Scan all intersection points of c to update the influence values of the arcs along the boundary of c accordingly.

Phase 3: Return the arc with the greatest influence value (among all NLCs) and the set of NLCs covering this arc (where the intersection of all NLCs of this set corresponds to the optimal region).

The detailed description of the above three phases can be found in Algorithm 4.1.

Algorithm 4.1 *MaxSegment* algorithm

1: // Phase 1
2: **for** each client point $o \in \mathcal{O}$ **do**
3: search the nearest neighbor of o in $\mathcal{P}$, says p
4: construct an NLC c, centered at o with radius $|o, p|$
5: **end for**
6: // Phase 2
7: choose the NLC c with the largest $w(c)$
8: initialize $MaxInf \leftarrow w(c)$ and $MaxS \leftarrow \{c\}$
9: **for** $i = 1$ to $|\mathcal{O}|$ **do**
10: // Step 1
11: find all NLCs intersected with NLC c_i and store them into list L
12: // Step 2
13: **for** each NLC $c_j \in L$ **do**
14: generate intersection arc $e = (q_1, q_2)$ where q_1 and q_2 are the intersection points between the boundaries of c_i and c_j, and assign both $q_1.NLC$ and $q_2.NLC$ with c_j
15: **if** $q_1 > q_2$ **then**
16: generate two sub-intersection arcs $e_1 = (q_1, 360°)$ and $e_2 = (0°, q_2)$
17: **end if**
18: store intersection points of the generated intersection arcs into Q
19: **end for**
20: // Step 3
21: sort the intersection points in Q according to their angle values
22: initialize $Inf \leftarrow w(c_i)$, $S \leftarrow \{c_i\}$
23: // Step 4
24: **for** each intersection point $t \in Q$ **do**

if t is the head of an arc **then**
  $Inf \leftarrow Inf + w(t.NLC)$, $S \leftarrow S \cup \{t.NLC\}$
else if t is the tail of an arc **then**
  $Inf \leftarrow Inf - w(t.NLC)$, $S \leftarrow S - \{t.NLC\}$
end if
if $Inf > MaxInf$ **then**
  $MaxInf \leftarrow Inf$, $MaxS \leftarrow S$
end if
end for
end for
// Phase 3
return $MaxInf$ and $MaxS$

We use Example 1 to further describe the process of the *MaxSegment* algorithm.

Example 1 Given point sets $\mathcal{O}$ and $\mathcal{P}$ as shown in Fig. 8a, where $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_5, o_6\}$ and $\mathcal{P} = \{p_1, p_2, p_3, p_4, p_5, p_6\}$. Assume that the weight of each NLC is equal to 1.

According to the algorithm description of *MaxSegment*, for datasets in Fig. 8a, we construct six NLCs, such as $c_1, c_2, c_3, c_4, c_5, c_6$, as shown in Fig. 8b. Since each NLC has the same weight and is equal to 1, we randomly choose c_1 as an initialized NLC and set $MaxInf = w(c_1) = 1$ and $MaxS = \{c_1\}$.

Next, the algorithm finds all NLCs intersected with NLC c_1 and store them into L. Then, we have $L = \{c_2, c_3, c_4\}$. Next, the algorithm computes the intersection points and generate the

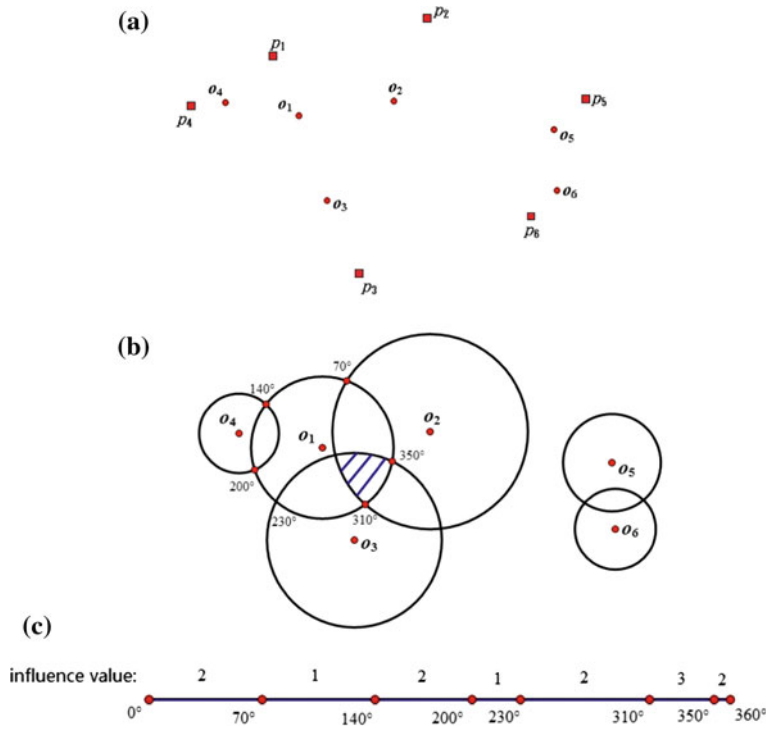


Fig. 8 An example of the *MaxSegment* algorithm

intersection arcs. Therefore, we have intersection arcs $e_1 = (310°, 70°)$, $e_2 = (140°, 200°)$, and $e_3 = (230°, 350°)$. Since the head of e_1 is larger than its tail, we split $e_1 = (310°, 70°)$ into two intersection arcs, namely, $e_{11} = (0°, 70°)$ and $e_{12} = (310°, 360°)$. All intersection points of NLC c_1 would be stored in Q. Next, we sort all intersection points of c_1. That is, $Q = \{0°, 70°, 140°, 200°, 230°, 310°, 350°, 360°\}$. Then we construct a line segment of c_1 as shown in Fig. 8c. According to the algorithm description, we update influence values of the arcs by scanning all intersection points of the intersection arcs along the line segment of c_1. During scanning, we initialize $Inf = w(c_1)$ (i.e., line 22). We first meet the head of intersection arc e_{11}, that is, 0°, and we increase the influence value of e_{11} by the weight of the NLC c_2 eclipsing this arc. Next, we move further and meet the tail of e_{11}, that is, 70°. Then, we decrease the influence value by the weight of the NLC c_2 eclipsing this arc.

Next, we process other intersection arcs similarly. Finally, we find the greatest influence value of an arc and the corresponding NLCs covering this arc, namely, $MaxInf = 3$ and $MaxS = \{c_1, c_2, c_3\}$. The influence values of all arcs along the boundary of NLC c_1 are listed in Fig. 8c.

According to the algorithm description, we take NLC c_2 as another scanned NLC and repeat the above steps. Similarly, we obtain $MaxInf = 3$ and $MaxS = \{c_1, c_2, c_3\}$. After processing all remaining NLCs, we find the optimal influence value of an arc and output the set of NLCs covering this arc. This set is equal to $MaxS = \{c_1, c_2, c_3\}$, which corresponds to the optimal region R (i.e., the shadow part in Fig. 8b) in the MaxBRNN problem.

With this example, we know that the *MaxSegment* algorithm would check the intersection points of all NLCs for a given dataset and can find the optimal region R. So it is easy to verify the following Theorem 4.1.

Theorem 4.1 *Algorithm 4.1 returns the region R with the largest influence value.*

4.3 Algorithm analysis

Time Complexity: We would give an analysis on the running time complexity and the space cost complexity of the *MaxSegment* algorithm.

It is easy to verify the following Lemma 4.3 by elementary mathematics.

Lemma 4.3 *The computation of the intersection points between two overlapping NLCs takes* $O(1)$ *time.*

Next, we analyze the running time of *MaxSegment*. Before we give the analysis, we define two notations, namely $\alpha(\cdot)$ and $\beta(\cdot)$. Given a dataset D of size $|D|$ and a query point q, we denote the time cost of finding the nearest neighbor from q in D by $\alpha(|D|)$. Besides, given a dataset D of size $|D|$, a query point q and a non-negative real number r, we denote the time cost of finding the answer for a range query from q with radius equal to r in D by $\beta(|D|)$.

Consider Phase 1 (Lines 1–5 of Algorithm 4.1). For each client point o, we need to find the nearest neighbor of o in $\mathcal{P}$, which takes $O(\alpha(|\mathcal{P}|))$ time. Since there are $|\mathcal{O}|$ client points, the total running time of Phase 1 is $O(|\mathcal{O}|\alpha(|\mathcal{P}|))$.

Consider Phase 2 (Lines 6–34 of Algorithm 4.1). Lines 7–8 take $O(|\mathcal{O}|)$ time. There are $|\mathcal{O}|$ iterations in lines 9–34. Consider an iteration (lines 10–33) that involves four steps for one NLC c_i.

– Step 1 (lines 10–11) finds all NLCs intersected with c_i and stores them into list L. This can be done by performing a range query at the center of c_i with the range equal to the radius of c_i on the set of all NLCs. Thus, Step 1 takes $O(\beta(|\mathcal{O}|))$ time.

- Step 2 (lines 12–19) involves a number of iterations. Consider an iteration (lines 14–18) where we are now considering one NLC c_j in L. Line 14 finds the intersection arc between c_i and c_j by finding the intersection points between the boundaries of c_i and c_j, which can be done in $O(1)$ time (by Lemma 4.3). It is easy to see that lines 15–18 can be done in $O(1)$ time if we implement Q with a linked list data structure. Since there are $|L|$ iterations in Step 2, Step 2 takes $O(|L|)$ time. Let m be the greatest number of NLCs overlapping with an NLC. Since $|L| = O(m)$, the time complexity of Step 2 is $O(m)$.
- Step 3 (lines 20–22) sorts the intersection points in Q according to the angle values, which can be done in $O(|Q| \log |Q|)$ time. The variable initialization in line 22 can be done in $O(1)$ time. Thus, the time complexity of Step 3 is $O(|Q| \log |Q|)$ time. Since $|Q| = O(m)$, Step 3 takes $O(m \log m)$ time.
- Step 4 (lines 23–33) involves a number of iterations. Consider an iteration (lines 25–33) where we are now considering one intersection point t in Q. It is easy to verify that lines 25–32 take $O(1)$ time when we implement S with the stack data structure. Since there are $|Q|$ iterations and $|Q| = O(m)$, Step 4 takes $O(m)$ time.

The overall running time of an iteration involving the above four steps is equal to $O(\beta(|\mathcal{O}|) + m + m \log m + m) = O(\beta(|\mathcal{O}|) + m \log m)$. Since there are $|\mathcal{O}|$ iterations, Phase 2 takes $O(|\mathcal{O}| + |\mathcal{O}| \cdot (\beta(|\mathcal{O}|) + m \log m)) = O(|\mathcal{O}| \cdot \beta(|\mathcal{O}|) + |\mathcal{O}| \cdot m \log m)$ time. It is easy to verify that Phase 3 (Lines 35-36 of Algorithm 4.1) takes $O(1)$ time.

The overall time complexity of the *MaxSegment* algorithm is $O(|\mathcal{O}|\alpha(|\mathcal{P}|)+|\mathcal{O}|\cdot\beta(|\mathcal{O}|)+ |\mathcal{O}| \cdot m \log m + 1) = O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}| \cdot \beta(|\mathcal{O}|) + |\mathcal{O}| \cdot m \log m)$ time.

Theorem 4.2 *The running time of the MaxSegment algorithm is* $O(|\mathcal{O}|\alpha(|\mathcal{P}|)+|\mathcal{O}|\cdot\beta(|\mathcal{O}|)+ |\mathcal{O}| \cdot m \log m)$.

Next, we give the theoretical bounds on $\alpha(\cdot)$ and $\beta(\cdot)$.

Given a dataset D of size $|D|$, $\alpha(|D|)$ corresponds to the time cost of a nearest neighbor query. In [2], this query can be done in $O(\log |D|)$ time with an index with the space of $O(|D|)$ size by using some data structures like the trapezoidal map over the Voronoi diagram. Thus, $\alpha(|D|) = O(\log |D|)$.

Given a dataset D of size $|D|$, $\beta(|D|)$ corresponds to the time cost of a range query. In [6], this query can be accomplished in $O(k + \log |D|)$ time where k is the number of points/answers returned for the query. Thus, $\beta(|D|) = O(k + \log |D|)$.

We are interested in analyzing the theoretical bound on the running time of the *MaxSegment* algorithm if the time complexities of the queries can be theoretically bounded. It is easy to verify that with some sophisticated implementations described [2,6], the running time of the *MaxSegment* algorithm can be simplified to $O(|\mathcal{O}| \log |\mathcal{P}| + |\mathcal{O}| \cdot (m + \log |\mathcal{O}|) + |\mathcal{O}| \cdot m \log m) = O(|\mathcal{O}| \log |\mathcal{P}| + m|\mathcal{O}| \log m + |\mathcal{O}| \log |\mathcal{O}|)$ time.

Although $\alpha(\cdot)$ and $\beta(\cdot)$ can be theoretically bounded as discussed above, in our implementation, we adopt the R*-tree data structure for the nearest neighbor query and the range query since it is shown to be efficient in practice and it is commonly used for these queries although this data structure does not have a good worst-case asymptotic performance.

Storage Complexity: From the algorithm description of *MaxSegment*, we know the *MaxSegment* algorithm needs to store three kinds of storage structures, (1) the R*-tree for point sets $\mathcal{O}$ and $\mathcal{P}$, (2) the list L storing NLCs intersected with one NLC, and (3) the set Q to store intersection points of all intersection arcs of one NLC. So, the storage cost of the *MaxSegment* algorithm is $O(R_p + |L| + |Q|)$ where R_p denotes the size of the R*-tree. In general, the size of L and the size of Q are small. Since $|L|$ and $|Q|$ are $O(m)$, the storage complexity can be simplified to $O(R_p + m)$. Thus, the major storage cost of the *MaxSegment* algorithm is the cost of storing the R*-tree.

5 Algorithm extension

We have three kinds of extensions that are described in Sects. 5.1, 5.2 and 5.3.

5.1 Extension to other varied MaxBRNN problems

There are some variations of the MaxBRNN problem, namely MaxBRkNN, Top-t MaxBRNN, and Top-t MaxBRkNN [26]. Our *MaxSegment* algorithm can be extended to these variations.

(1)**MaxBRkNN:** In the basic MaxBRNN problem, we find a server point $p \in \mathcal{P}$ that is the nearest neighbor of a client point $o \in \mathcal{O}$. In practice, we may want to search the k nearest neighbors of a client point instead of the nearest neighbor only. We can give an extension to find the reverse k-nearest neighbors of a server point p, denoted by k-BRNN of p. The purpose of the MaxBRkNN problem is to find the optimal region R, such that a new server p is set up in R, the size of k-BRNN of p is maximized. Our algorithm can be extended to the MaxBRkNN problem. We only need to construct an NLC according to the k-th nearest neighbors of o rather than the nearest neighbor of o for each client point o. In particular, we just need to modify lines 1–4 in Algorithm 4.1 accordingly.
(2)**Top-t MaxBRNN:** The basic MaxBRNN problem is to find one optimal region with the greatest size of BRNN. We can make an extension to find t regions that give the greatest size of BRNN. That is, the top-t MaxBRNN problem is to find t regions with the greatest influence values with respect to BRNN. Our algorithm can be extended to the top-t MaxBRNN problem. We need to maintain t regions with the greatest influence values, rather than maintaining one region with the greatest influence value. In particular, we only need to modify lines 30–32 in Algorithm 4.1.
(3) **Top-tMaxBRkNN:** We can combine the above two extensions and achieve the top-t MaxBRkNN problem. The purpose of top-t MaxBRkNN problem is to find t regions with the greatest values with respect to k-BRNN instead of BRNN. Our algorithm can be extended to the Top-t MaxBRkNN problem. We just need to modify lines 1–4 and lines 30–32 in Algorithm 4.1 together.

5.2 Extension to other metric spaces

The basic algorithm *MaxSegment* is discussed in the L_2-norm space. In this subsection, we would like to extend our algorithm to another metric space, namely, the L_p-norm space. The Minkowski distance is used in the L_p-norm space. Given two n-dimensional space points in metric space $\mathbb{D}$, $q_1 = (x_1, x_2, \ldots, x_n)$ and $q_2 = (y_1, y_2, \ldots, y_n)$, the Minkowski distance of order p between q_1 and q_2 is defined as $\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$. Minkowski distance is typically used with p being 1 or 2. The latter is the Euclidean distance, while the former is sometimes known as the Manhattan distance. In the extreme case when $p = \infty$, we obtain the Chebyshev distance.

The *MaxSegment* algorithm is based on an important concept of NLC in the L_2-norm space. Note that there are *at most two* intersection points between the boundaries of two NLCs in the L_2-norm space and these intersection points are used to find the optimal region. In the L_p-norm space, we can use a similar concept of *nearest location region* (NLR) [27]. The nearest location *region* is a generalized concept of the nearest location *circle* where a circle can be regarded as a region. The major challenge in the L_p-norm space is that there are an *infinite number* of intersection points between the boundaries of two overlapping

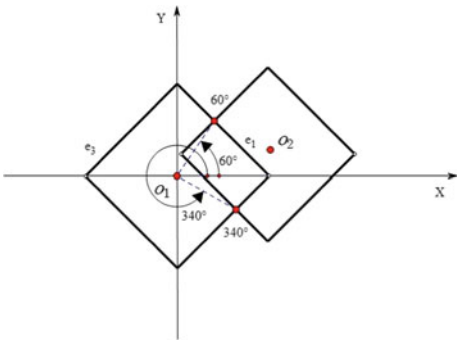


Fig. 9 Representation of NLR arc by angle values

NLRs and thus adopting the *MaxSegment* algorithm *directly* in the L_p-norm space can be computationally expensive. However, interestingly, such an infinite number of intersection points form a *fixed number* of continuous curves/segments where all intersection points lie on these curves and each curve has two *end points* only. In this paper, in the L_p-norm space, we find that it is sufficient to use all endpoints instead of all intersection points to find the optimal region and thus we can derive an efficient algorithm.

Definition 5.1 (*Nearest Location Region*) Given a client point o, the *nearest location region* of o is defined to be a region such that, for each point q along the boundary of the region, $|o, q|$ is equal to $|o, p|$ where p is the nearest neighbor of o in $\mathcal{P}$ with respect to the metric space $\mathbb{D}$.

If the metric space $\mathbb{D}$ is the L_2-norm space, then NLR of a client point o is a circle. If the metric space $\mathbb{D}$ is the L_1-norm space, then NLR is a square rotated 45° clockwise. If the metric space $\mathbb{D}$ is the L_∞ -norm space, then NLR is a square. Similar to the L_2-norm space, there exist the following properties of NLR.

Property 5.1 *[27] The region R returned by the MaxBRNN query in a metric space $\mathbb{D}$ can be represented by an intersection of multiple NLRs.*

Property 5.2 *[27] If an NLR covers another NLR, the boundaries of the two NLRs must share at least one point.*

For the sake of consistency with the L_2-norm space, we re-use the term of arc to denote such overlapping edges in the L_p-norm space. Here, an arc corresponds to the continuous edge that is along the boundary and formed by two endpoints of two overlapping NLRs.

Similar to the L_2-norm space, we can map the endpoints of an edge into the angle values. Then we can represent an arc by the angle values. Similarly, the angle values are also computed in an anti-clockwise direction. For example, intersection arc e_1 in Fig. 9 can be represented by the angle values, such as, $e_1 = (340°, 60°)$, which need to be split into two intersection arcs (340°, 360°) and (0°, 60°).

Thus, the related lemmas on an arc in the L_2-norm space can also be extended to the L_p-norm space. It is easy to see Lemmas 4.1, 4.2, and 4.3 also hold in the L_p-norm space if we consider NLR instead of NLC. For the sake of convenience, we re-state those lemmas in $\mathbb{D}$ as follows. The corresponding Lemma 4.1, Lemma 4.2, and Lemma 4.3 become Lemma 5.1, Lemma 5.2, and Lemma 5.3, respectively.

Lemma 5.1 *Given an NLR and its intersection arcs owned by this NLR, we can find the greatest influence value of an arc by scanning the endpoints of all intersection arcs of a given NLR.*

Lemma 5.2 *Let C be a set of NLRs whose intersection corresponds to the optimal region R returned by a MaxBRNN query. Then, R contains at least an optimal arc along the boundary of a certain NLR (i.e., the arc with the greatest influence value).*

With elementary mathematics, it is easy to verify the following Lemma 5.3.

Lemma 5.3 *The computation of the end points of all arcs owned by a given NLR but eclipsed by another NLR takes* $O(1)$ *time.*

Algorithm 4.1 can also be re-used for the L_p-norm space. Assume that each NLR is intersected with at most m other NLRs. The running time of the *MaxSegment* algorithm for the L_p-norm space is also the same as that in the L_2-norm space.

Theorem 5.1 *The running time of the MaxSegment algorithm for the* L_p*-norm space is* $O(|\mathcal{O}|log|\mathcal{P}| + m|\mathcal{O}|log\ m + |\mathcal{O}|log|\mathcal{O}|)$.

5.3 Extension to three-dimensional space

In this subsection, we would like to extend the *MaxSegment* algorithm in the L_2-norm for the two-dimensional space to a three-dimensional space.

In the two-dimensional space, the correctness of the *MaxSegment* algorithm depends on at least two concepts. The first concept is that the optimal region in the two-dimensional case can be represented by the intersection of multiple NLCs. The second concept is that all the arcs each of which is generated by the boundaries of *two* overlapping NLCs can be used to find the optimal region. In the three-dimensional space, we adapt the above two concepts due to some differences between the two-dimensional case and the three-dimensional case. For the first concept, in the three-dimensional case, we propose a new concept of nearest location spheres (NLSs). Thus, the optimal region can be represented by the intersection of multiple NLSs instead of NLCs. We can regard that an NLS has the same meaning as NLC in the context of the three-dimensional case. The *surface* of an NLS in the three-dimensional case can be regarded as the *boundary* of an NLC in the two-dimensional case. In the following, for the sake of consistency, when we write the boundary of a sphere (or NLS), we mean the surface of this sphere. For the second concept, in the three-dimensional case, consider two overlapping NLSs. In this three-dimensional case, the boundaries of these two overlapping NLSs generate a *circle* instead of an arc that can be generated by two NLCs in the two-dimensional case. We will elaborate how the circle is generated later. In the following, interestingly, we find that *three* overlapping NLSs can generate an *arc* in this three-dimensional case setting. Based on all arcs generated by any three overlapping NLSs, we can find the optimal region accordingly. We will explain this later in this section.

In the following, we give the formal definition.

Definition 5.2 Given a client point o, the NLS of o is defined to be a region such that for each point q along the boundary of the region, $|o, q|$ is equal to $|o, p|$ where p is o's nearest neighbor in $\mathcal{P}$ in a three-dimensional space.

Based on the concept of NLS, we have the following properties.

Property 5.3 *[27] The three-dimensional space region R returned by the MaxBRNN query in a three-dimensional space can be represented by an intersection of multiple NLSs.*

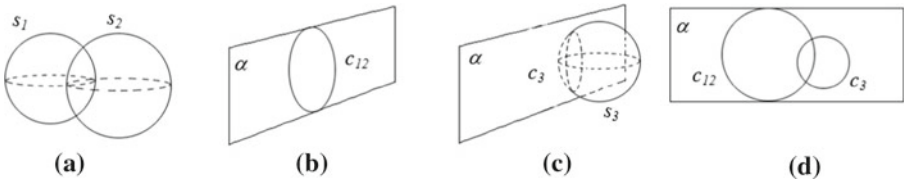


Fig. 10 An example showing intersection arcs in a three-dimensional space

Property 5.4 *[27] If an NLS covers another NLS, the boundaries of the two NLSs must share at least one point.*

Consider the Cartesian coordinate system. Assume that NLS s is centered at coordinate (a, b, c) with radius r. Then, the *boundary* of s can be expressed as a set of point (x, y, z) as $\{(x, y, z)|(x-a)^2+(y-b)^2+(z-c)^2 = r^2\}$, and the *insider* of s can be expressed as a set of point (x, y, z) as $\{(x, y, z)|(x-a)^2+(y-b)^2+(z-c)^2 < r^2\}$. The NLS s itself can be expressed as a set of point (x, y, z) as $\{(x, y, z)|(x-a)^2+(y-b)^2+(z-c)^2 \leq r^2\}$. We say that a point q is covered by NLS s, if q is on the boundary of NLS s or q is inside NLS s. We say that arc e is covered by NLS s if each point on e is covered by NLS s.

In a three-dimensional space, the *intersection arc* is based on *three* NLSs. Suppose that we are given three NLSs, s_1, s_2, and s_3, each of which overlaps with the other two NLSs. The intersection of the boundaries of two three-dimensional NLSs s_1 and s_2 (as shown in Fig. 10a) would generate a circle, c_{12}, which is on a plane denoted by α (see Fig. 10b). Circle c_{12} is called the (s_1, s_2)-*circle* and plane α is called the (s_1, s_2)-*plane*. We say that this plane α is generated from the two NLSs, s_1 and s_2. The plane α intersects with another three-dimensional NLS s_3 and generates a circle c_3 that is also on the plane α (see Fig. 10c). Circle c_3 is called the s_3-*circle on plane* α. Similarly, we say that this circle is *eclipsed* by the NLS s_3 (or the circle c_3 on the plane). Now, we have two circles c_{12} and c_3 (on the plane α), which have a similar scenario as those in the two-dimensional case. Both circles c_{12} and c_3 generate intersection arcs on the plane α (see Fig. 10d). The detailed computation of the (s_1, s_2)-plane, the (s_1, s_2)-circle, and s_3-circle on a plane in a three-dimensional space is given in the "Appendix".

In Fig. 10, given three overlapping NLSs, namely s_1, s_2, and s_3, we know that we can form intersection arcs when we *first* consider the intersection between two NLSs s_1 and s_2 (as shown in Fig. 10a, b) and *then* consider the intersection between the plane generated from the previous intersection and the third NLS s_3 (as shown in Fig. 10c, d). It is noted that there is an ordering of processing the three NLSs. In general, we consider all possible orderings for any three NLSs. For example, one possible ordering of processing is that we can first consider the intersection between s_2 and s_3 and then consider the intersection between the plane generated from the previous intersection and the remaining NLS s_1.

Besides, given three overlapping NLSs and a particular ordering to process these three overlapping NLSs, we can generate the intersection arcs as shown in Fig. 10d. Let $\mathcal{A}$ be a set of all possible intersection arcs generated by the overlapping NLSs with any processing orderings.

In the following, we will first prove that the optimal region R returned by the MaxBRNN query in a three-dimensional space contains at least the arc in $\mathcal{A}$ that is covered by the greatest number of NLSs. Next, we will introduce how to find the optimal region R formed by the arcs.

Lemma 5.4 *Let C be a set of NLSs whose intersection corresponds to the optimal region R returned by a MaxBRNN query. Then, R contains at least an arc $a \in \mathcal{A}$ that is covered by the greatest number of NLSs.*

Proof There are two cases.

(1) Suppose that C contains only one NLS, which means that the optimal solution comes from a single NLS without any overlap or intersection with other NLSs. The optimal region R is the single NLS with the greatest influence value (i.e., the greatest weight) among all NLSs. Any arc along the boundary of this NLS is the optimal arc contained by region R. This lemma holds.
(2) Suppose that C contains more than one NLS. Assume that NLSs are $s_1, s_2, \ldots s_n$ where $n \geq 2$. It is easy to know the influence value of R is equal to $\sum_{i=1}^{n} w(s_i)$. According to Property 5.3, we can assume that $R = s_1 \cap s_2 \cap \cdots \cap s_n$. Then, we have a set of points $R' = \{q | q \in B(s_i) \cap B(s_j)$, and $q \in s_1 \cap s_2 \cap \cdots \cap s_{i-1} \cap s_{i+1} \cap \cdots \cap s_{j-1} \cap s_{j+1} \cap \cdots \cap s_n\}$ where $i < j$, $B(s_i)$ and $B(s_j)$ denote the set of points along the boundaries of s_i and s_j, respectively. From the expression of points of an NLS, it is easy to know $B(s_i) \subseteq s_i$ and $B(s_j) \subseteq s_j$. Thus, we have $R' \subseteq R$. From Property 5.4, we can know that there is no such case that the area of one NLS is inside the area of another NLS (i.e., covering relationship) but the boundaries of both NLSs do not share any points. So we have $R' \neq \emptyset$. In other words, if there is an optimal three-dimensional region R, then we can always find the subset of R, R'. It is obvious that the influence value of R' is equal to that of R since the optimal region R is consistent. Intuitively, the intersection of boundaries of s_i and s_j, namely, $B(s_i) \cap B(s_j)$, would generate a (s_i, s_j)-circle on a two-dimensional (s_i, s_j)-plane. So R' is constructed by the intersection of the boundary of the (s_i, s_j)-circle with other three-dimensional NLSs. The intersection of the boundary of a (s_i, s_j)-circle and a three-dimensional NLS would generate an intersection arc $a \in \mathcal{A}$, as shown in Fig. 10. It is easy to know that the influence value of intersection arc a is also equal to $\sum_{i=1}^{n} w(s_i)$. So the optimal region R covers an optimal arc. This lemma holds. □

Lemma 5.4 tells us that we can transform the MaxBRNN problem in a three-dimensional space into a two-dimensional arc search. That is, we can map the intersection points of intersection arcs into different angle values. Then, we can scan all the intersection points to find the optimal arc with the greatest influence value. The *MaxSegment* algorithm in a three-dimensional space includes the following three major phases.

Phase 1: Construct NLSs for a given dataset.
Phase 2: Construct all possible intersection arcs for each NLS and find the influence value of each arc. In particular, for each NLS s, we do the following two steps.

Step 1: Find all the other NLSs intersected with s.
Step 2: For each NLS s' intersected with s, we do the following steps.
  Step 2a: Construct the (s, s')-circle, says c, and the (s, s')-plane, says α.
  Step 2b: For each NLS s'' such that the s''-circle on plane α, says c'', intersects with c, compute all intersection points between c and c'' and construct intersection arcs along the boundary of c
  Step 2c: Sort intersection points (i.e., head or tail) of all intersection arcs of c according to the angle values
  Step 2d: Scan all intersection points of c to update the influence values of arcs along the boundary of c accordingly.

Phase 3: Return the arc with the greatest influence value (among all arcs generated) and the set of NLSs covering this arc (where the intersection of all NLSs of this set corresponds to the optimal region).

The detailed description of the above three phases can be found in Algorithm 5.1. Similar to the two-dimensional case, we denote the NLS centered at client point $o_i \in \mathcal{O}$ by s_i for $i \in [1, |\mathcal{O}|]$.

Algorithm 5.1 The *MaxSegment* algorithm in a three-dimensional space

// Phase 1
for each client point $o \in \mathcal{O}$ **do**
  search the nearest neighbor of o in $\mathcal{P}$, says p
  construct an NLS s, centered at o with radius $|o, p|$
end for
// Phase 2
choose the NLS s with the largest $w(s)$
initialize $MaxInf \leftarrow w(s)$ and $MaxS \leftarrow \{s\}$
for $i = 1$ to $|\mathcal{O}|$ **do**
  // Step 1
  find all NLSs intersected with NLS s_i and store them into list L
  // Step 2
  for each NLS $s_j \in L$ **do**
    // Step 2a
    compute the (s_i, s_j)-circle, says c_{ij}, and the (s_i, s_j)-plane, says α,
    // Step 2b
    find all NLSs such that each of these NLSs, says s_k, has its s_k-circle on plane α, which intersects with c_{ij} and store them into list M
    for each NLS $s_k \in M$ **do**
      generate intersection arc $e = (q_1, q_2)$ where q_1 and q_2 are the intersection points between the boundaries of c_k and c_{ij}, and assign both $q_1.NLS$ and $q_2.NLS$ with s_k
      if $q_1 > q_2$ **then**
        generate two sub-intersection arcs $e_1 = (q_1, 360°)$ and $e_2 = (0°, q_2)$
      end if
      store intersection points of the generated intersection arcs into Q
    end for
    // Step 2c
    sort the intersection points in Q according to their angle values
    initialize $Inf \leftarrow w(s_i) + w(s_j)$ and $S \leftarrow \{s_i, s_j\}$
    // Step 2d
    for each intersection point $t \in Q$ **do**
      if t is the head of an arc **then**
        $Inf \leftarrow Inf + w(t.NLS)$, $S \leftarrow S \cup \{t.NLS\}$
      else if t is the tail of an arc **then**
        $Inf \leftarrow Inf - w(t.NLS)$, $S \leftarrow S - \{t.NLS\}$
      end if
      if $Inf > MaxInf$ **then**
        $MaxInf \leftarrow Inf$, $MaxS \leftarrow S$
      end if
    end for
  end for
end for
// Phase 3
return $MaxInf$ and $MaxS$

Next, we give the theoretical analysis on the time complexity and the space complexity of the *MaxSegment* algorithm.

Time Complexity: Given two NLSs s and s', the computation of the (s, s')-circle and the computation of the (s, s')-plane are given in the "Appendix". From this computation, it is easy to verify that Lemma 5.5 holds.

Lemma 5.5 *Given two NLSs s and s', the computation of the computation of the (s, s')-circle and the computation of the (s, s')-plane in a three-dimensional space takes $O(1)$ time.*

We analyze the running time of *MaxSegment* in a three-dimensional case. In the following analysis, we also use the two notations, namely $\alpha(\cdot)$ and $\beta(\cdot)$. But, the context is based on the three-dimensional case instead of the two-dimensional case.

Consider Phase 1 (Lines 1–5 of Algorithm 5.1). Similar to the two-dimensional case, Phase 1 takes $O(|\mathcal{O}|\alpha(|\mathcal{P}|))$ time.

Consider Phase 2 (Lines 6–40 of Algorithm 5.1). Lines 7–8 take $O(|\mathcal{O}|)$ time. There are $|\mathcal{O}|$ iterations in lines 9–40. Consider an iteration (lines 10–39) which involves two steps for one NLS s_i.

- Similar to the two-dimensional case, Step 1 (lines 10–11) takes $O(\beta(|\mathcal{O}|))$ time.
- Step 2 (lines 12–39) involves a number of iterations. Consider an iteration (lines 14–38) where we are now considering one NLS s_j in L.

- Step 2a (lines 14–15) finds the (s_i, s_j)-circle, says c_{ij}, and the (s_i, s_j)-plane, says α, which takes $O(1)$ time by Lemma 5.5.
- Note that the number of NLSs each of which is denoted by s_k and has its s_k-circle on plane α, which intersects with c_{ij} is $O(m)$. Similar to the two-dimensional case, Step 2b (lines 16–24), Step 2c (lines 25–27), and Step 2d (lines 28–38) take $O(m)$ time, $O(m \log m)$ time and $O(m)$ time, respectively.

There are $O(m)$ iterations in Step 2 and thus Step 2 takes $O(m \cdot (1 + m + m \log m + m)) = O(m^2 \log m)$ time.

The time complexity of executing Step 1 and Step 2 for an iteration in Phase 2 is $O(\beta(|\mathcal{O}|) + m^2 \log m)$ time. Note that there are $|\mathcal{O}|$ iterations in Phase 2. Thus, Phase 2 takes $O(|\mathcal{O}| + |\mathcal{O}| \cdot (\beta(|\mathcal{O}|) + m^2 \log m)) = O(|\mathcal{O}|\beta(|\mathcal{O}|) + |\mathcal{O}|m^2 \log m)$ time. It is easy to verify that Phase 3 (Lines 41–42 of Algorithm 5.1) takes $O(1)$ time.

The overall time complexity of the *MaxSegment* algorithm is $O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + |\mathcal{O}|m^2 \log m + 1) = O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + |\mathcal{O}|m^2 \log m)$ time.

Theorem 5.2 *The running time of the MaxSegment algorithm is $O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + |\mathcal{O}|m^2 \log m)$.*

It is easy to verify that with some sophisticated implementations described [2,6], the running time of the *MaxSegment* algorithm can be simplified to $O(|\mathcal{O}| \log |\mathcal{P}| + |\mathcal{O}|(m + \log |\mathcal{O}|) + |\mathcal{O}|m^2 \log m) = O(|\mathcal{O}| \log |\mathcal{P}| + |\mathcal{O}| \log |\mathcal{O}| + |\mathcal{O}|m^2 \log m)$ time.

Similar to the two-dimensional case, due to the popular usage of the R*-tree data structure for the nearest neighbor query and the range query, in our implementation, we adopt the R*-tree data structure for the queries.

Storage Complexity: Similar to Algorithm 4.1, the storage cost of the *MaxSegment* algorithm in a three-dimensional space is $O(R_p + |L| + |Q| + |M|)$ where R_p denotes the size of the R*-tree. Note that $|M|$ is the greatest number of NLSs intersected with a circle generated by a pair of two NLSs. In general, the sizes of L, Q, and M are small. Since $|L| = O(m)$, $|Q| = O(m)$ and $|M| = O(m)$, the storage complexity can be simplified to $O(R_p + m)$. Thus, the major storage cost of Algorithm 5.1 is the cost of storing the R*-tree.

6 Experimental results

In this section, we perform a set of experiments to verify the efficiency of our solution. The algorithms were implemented in C/C++ and the experiments were executed on a PC with an Intel 2.13 GHz CPU and 3 GB memory.

We conducted the experiments on both real and synthetic datasets. The real datasets are available at http://www.rtreeportal.org/spatial.html. We deploy four real datasets called CA, LB, GR, and GM, which contain two-dimensional points representing geometric locations in California, Long Beach Country, Greece, and Germany, respectively. The sizes of the datasets are summarized in Table 1. For datasets containing rectangles, we transform them into points by taking the centroid of each rectangle. For each dataset, each dimension of the data space is normalized to range [0, 10,000]. Since our problem involves two datasets, namely $\mathcal{P}$ and $\mathcal{O}$, we generated four sets for real datasets, namely CA-GR, LB-GR, CA-GM, and LB-GM, representing ($\mathcal{P}$, $\mathcal{O}$) = (CA,GR), (LB,GR),(CA,GM), and (LB,GM), respectively.

Following the setting in [26], in the synthetic datasets, we create point set $\mathcal{P}$ following Gaussian distribution and point set $\mathcal{O}$ following Zipfian distribution. The coordinates of each point are generated in the range [0, 10,000]. In point set $\mathcal{P}$, each coordinate follows Gaussian distribution where the mean and the standard deviation are set to 5,000 and 2,500, respectively. In point set $\mathcal{O}$, each coordinate follows Zipfian distribution skewed toward origin O where the skew coefficient is set to 0.8. All coordinates of each point are generated independently. We created two-dimensional and three-dimensional points in our experiments.

The weight of each client point in both real datasets and synthetic datasets is set to 1 in the following experimental results. We also conducted experiments where the weight of each client point is any positive integer. Since the results are similar, for the interest of space, we only reported the results when the weight is equal to 1. In the experiments, we focus on the study of top-t MaxBRkNN since it is more general than MaxBRNN, MaxBRkNN, and top-t MaxBRNN. There are two parameters in the top-t MaxBRkNN problem, namely k and t. The parameter k is the parameter used in the k-th nearest neighbor of a client point o for the top-t MaxBRkNN problem. The parameter t is the parameter used in determining t regions with the greatest influence values with respect to BRkNN.

MaxOverlap [26] is the best-known algorithm for the MaxBRNN problem, which is 100,000 times faster than *Arrangement* [3,4] when the size of $\mathcal{O}$ is 250. Since the *MaxOverlap* algorithm is better than other algorithms both in terms of the running time and the storage cost, we just compare our proposed algorithm with the *MaxOverlap* algorithm in the experiments.

As indicated earlier, we adopt an R*-tree as an indexing structure for the nearest neighbor search and the k-th nearest neighbor search where the node size is fixed to 1 K byte. The maximum number of entries in a node is equal to 50 and 36 for the dimensionality equal to 2 and 3, respectively. We set the minimum number of entries in a node to be equal to half of the maximum number of entries. In the experiments, we study the effect of dataset cardinality, k and t in terms of two measurements: (1) execution time, and (2) storage.

Table 1 Summary of the real datasets

Dataset	Cardinality
CA	62,556
LB	53,145
GR	23,268
GM	36,334

6.1 Performance in two-dimensional case

In the experiments, the default values for the sizes of $\mathcal{O}$ and $\mathcal{P}$ are given in Table 2.

6.1.1 Effect of cardinality

Figure 11a, b are the results on synthetic datasets in which the size of $\mathcal{O}$ varies from 20,000 to 180,000 and the size of $\mathcal{P}$ is equal to $2|\mathcal{O}|$. Figure 11a shows that our *MaxSegment* algorithm is faster than the *MaxOverlap* algorithm in all cases. As the cardinality increases, the running time of both algorithms also increases. When the size is 180K, the execution time of the *MaxOverlap* algorithm is 4,500 s while the *MaxSegment* algorithm is 70 s, which means that the *MaxSegment* algorithm is 60 times faster than the *MaxOverlap* algorithm.

Both the *MaxOverlap* and *MaxSegment* algorithms use the R*-tree to index spatial data. Besides, the *MaxOverlap* algorithm needs to maintain an overlap table for all points of $\mathcal{P}$, and the size is $O(|\mathcal{O}|m)$. Instead of the overlap table, the *MaxSegment* algorithm uses a temporary list to store all intersection points of intersection arcs for an NLC. The size of the temporary list is relatively small and is equal to $O(m)$ where m is the greatest number of intersecting NLCs. Figure 11b shows that the *MaxSegment* algorithm needs less memory than the *MaxOverlap* algorithm.

6.1.2 Effect of k

As shown in Fig. 12a, the execution times of both the *MaxOverlap* and *MaxSegment* algorithms increase with k. That is because as k increases, the radius of an NLC increases, and it is more likely that an NLC for a client point overlaps with another NLC, which makes the influence value larger. The experiment shows that the increase in the execution time of *MaxSegment* is smaller than that of *MaxOverlap* when k increases. So, the *MaxSegment*

Table 2 Default cardinalities in synthetic datasets

Dataset	Default value
$\|\mathcal{O}\|$	50K
$\|\mathcal{P}\|$	100K

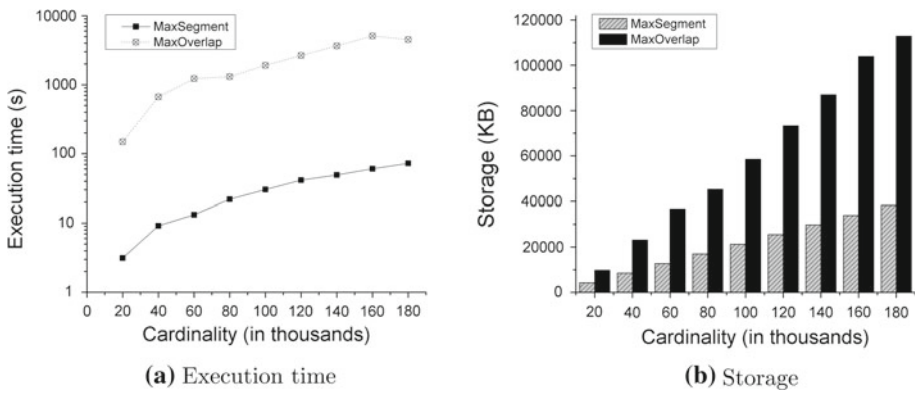


Fig. 11 Effect of cardinality (synthetic datasets). **a** Execution time, **b** storage

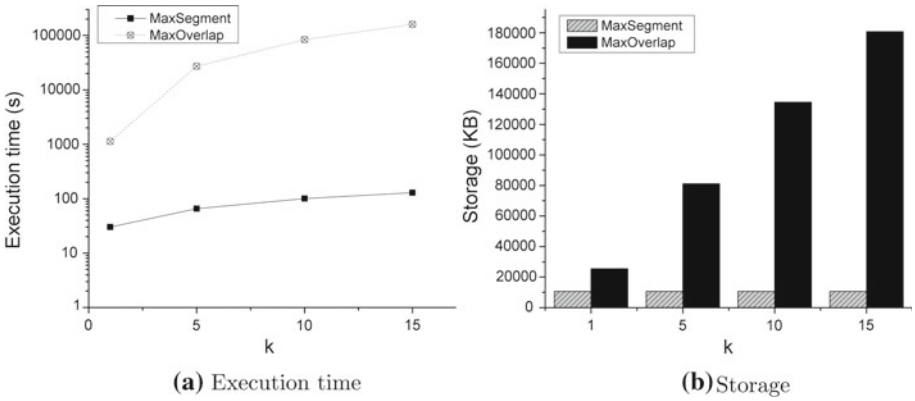


Fig. 12 Effect of k (synthetic datasets). **a** Execution time, **b** storage

algorithm is much scalable with respect to k. Figure 12b shows that the storage of the *MaxSegment* algorithm is almost unchanged whereas the *MaxOverlap* algorithm increases when k increases. That is because k is independent of the R*-tree storage. While k becomes larger, the R*-tree storage remains the same, and the increased storage of the *MaxSegment* algorithm is very small and can almost be omitted. On the other hand, with the increase of k, the overlap table of the *MaxOverlap* algorithm becomes larger as there are more intersected NLCs. So, the *MaxSegment* algorithm needs less storage space than the *MaxOverlap* algorithm.

6.1.3 Effect of t

We conducted experiment with t values of 1, 5, 10, and 15. The execution time and the storage remain nearly unchanged for both the *MaxSegment* and *MaxOverlap* algorithms when t changes. That is because we simply keep a queue to store the information about the first t-th greatest values and the cost of storing this queue is very insignificant compared with the overall storage cost.

6.1.4 Effect of real datasets

We conducted experiments on the four sets of real datasets, namely CA-GR, LB-GR, CA-GM, and LB-GM. The results are similar to the synthetic datasets. Figure 13 shows the experimental results when we vary k for dataset CA-GM. Figure 14 shows the experimental results when we vary t for dataset CA-GM. For dataset CA-GR, Fig. 15 shows the results when we vary k, and Fig. 16 shows the results when we vary t.

6.2 Performance in the L_1-norm

In the previous section, we conducted experiments in the L_2-norm space. In Sect. 5, we introduce how to extend the *MaxSegment* algorithm to other metric spaces. In this section, we choose the L_1-norm metric, one of the L_p-norm metric spaces, to study the algorithm performance. The reason why we choose the L_1-norm metric is that the L_1-norm metric is a well-known metric. In the L_1-norm metric, we compared the *MaxSegment* algorithm with the *MaxOverlap* algorithm on synthetic and real datasets. In the synthetic dataset, the size of

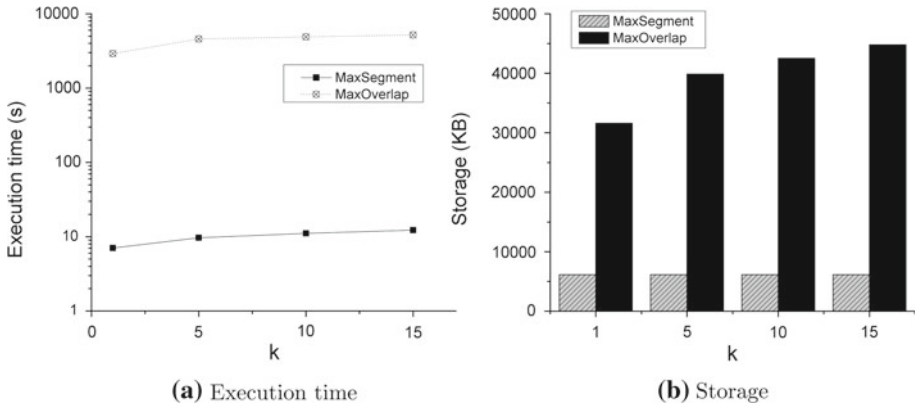


Fig. 13 Effect of k (CA-GM). **a** Execution time, **b** storage

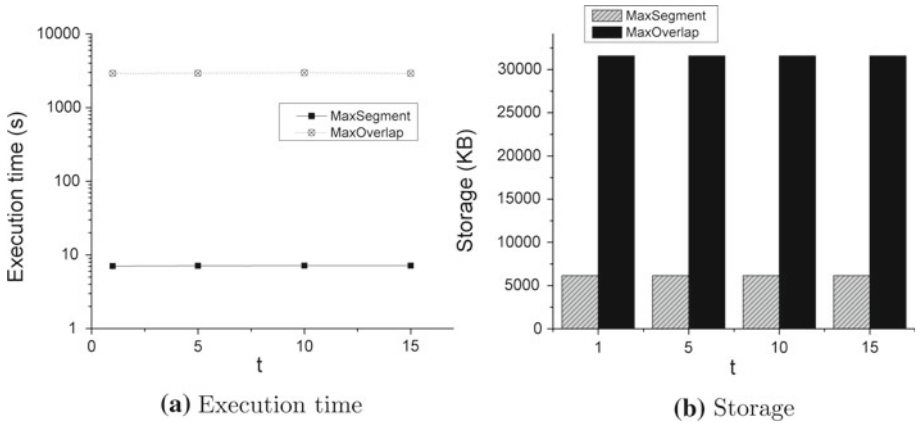


Fig. 14 Effect of t (CA-GM). **a** Execution time, **b** storage

$\mathcal{O}$ varies from 20K to 100K. In the real dataset, we use the CA-GR dataset where the size of $\mathcal{O}$ is 20K. In both synthetic and real datasets, the size of $\mathcal{P}$ is equal to $2|\mathcal{O}|$.

Figure 17 shows the execution times of the *MaxSegment* algorithm and the *MaxOverlap* algorithm for the synthetic dataset when we vary the cardinality of the dataset, the value of k and the value of t, respectively. Figure 18 shows the execution times for real dataset CA-GR. As shown in these figures, the performance of the *MaxSegment* algorithm and the *MaxOverlap* algorithm in the L_1-norm is very similar to that in the L_2-norm. That is, the algorithm execution time increases when either the cardinality size or k increases, but it is not sensitive to t. In both synthetic and real datasets, the *MaxSegment* algorithm is faster than the *MaxOverlap* algorithm.

6.3 Performance in three-dimensional case

Figure 19 shows the results in the three-dimensional space, which are similar to those in the two-dimensional space. As shown in these figures, the *MaxSegment* algorithm is better than the *MaxOverlap* algorithm in the experiments in regard to the cardinality of dataset, t and k. As the cardinality of dataset or k increases, the execution time of both algorithms increases.

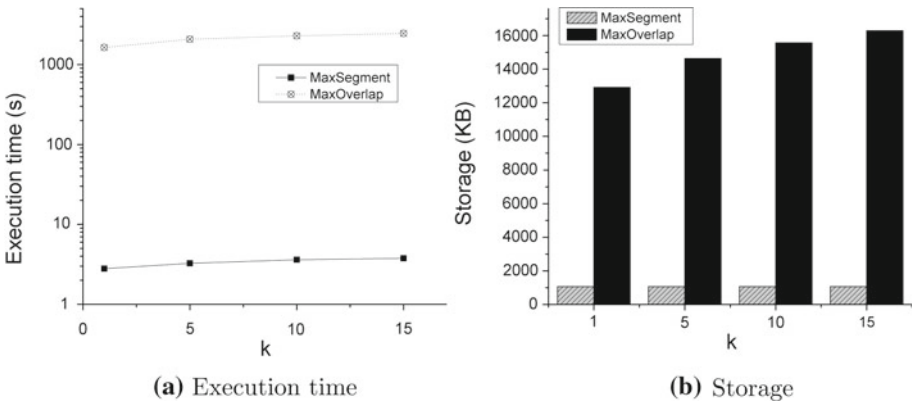


Fig. 15 Effect of k (CA-GR). **a** Execution time, **b** storage

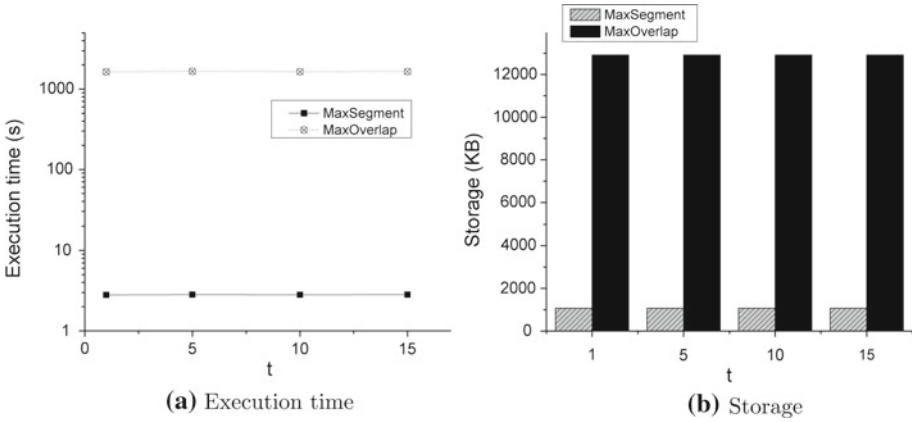


Fig. 16 Effect of t (CA-GR). **a** Execution time, **b** storage

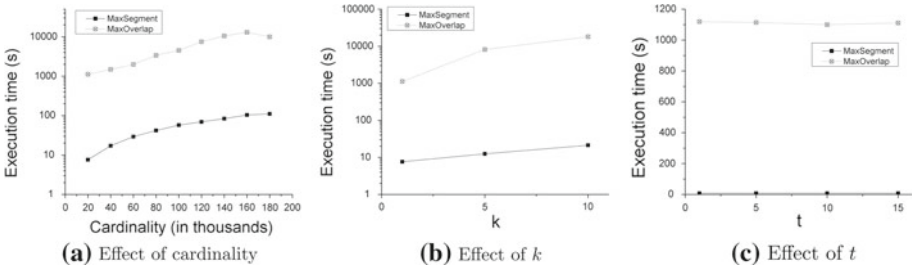


Fig. 17 Execution time for the L_1-norm experiments (synthetic datasets). **a** Effect of cardinality, **b** effect of k, **c** effect of t

Compared with the *MaxOverlap* algorithm, the increase of the *MaxSegment* algorithm is smaller. In addition, both algorithms are not sensitive to t. It is noted that, in Fig. 19a, the execution time of the *MaxOverlap* algorithm in the three-dimensional space is smaller than that in the two-dimensional space with the same cardinality (Please see Fig. 11a). It is because the NLSs scatter sparsely in a higher dimension, which reduces the number of NLSs

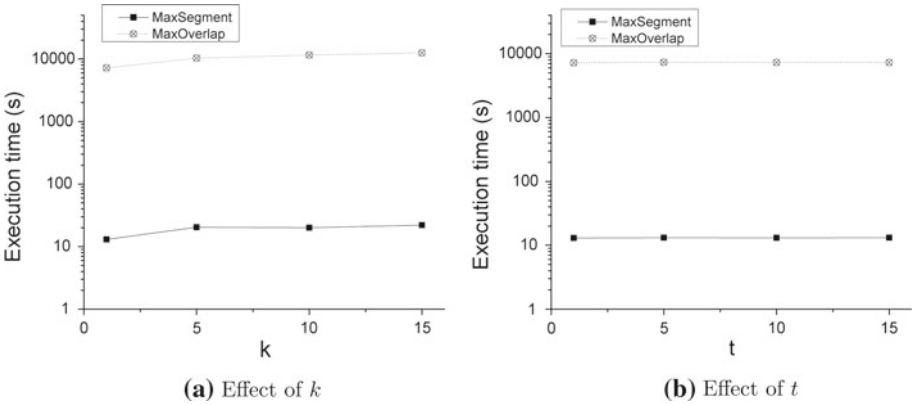


Fig. 18 Execution time for the L_1-norm experiments (CA-GR). **a** Effect of k, **b** effect of t

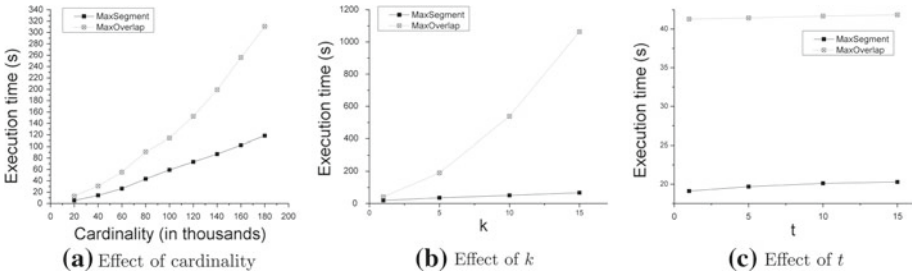


Fig. 19 Execution time for the three-dimensional case (synthetic datasets). **a** Effect of cardinality, **b** effect of k, **c** effect of t

intersected with an NLS. In this synthetic dataset, the influence value in the three-dimensional space is almost smaller than 100 whereas the influence value in the two-dimensional is several thousands.

7 Conclusion

In this paper, we studied the MaxBRNN problem and proposed a new approach called the *MaxSegment* algorithm in the case of the L_2-norm for a two-dimensional space. Then, we extended our algorithm to other variations of the MaxBRNN problem with the consideration of other metric spaces and a three-dimensional space. Finally, we constructed a set of experiments to compare our proposed algorithm with the existing *MaxOverlap* algorithm on both real and synthetic datasets. The experimental results verified the efficiency of our proposed approach. In the future, we would like to study further on MaxBRNN in a higher dimensional space. We would like to consider MaxBRNN in road network databases. One possible direction is to consider how to start a new service by considering a number of trajectories instead of static points [17].

Acknowledgments We thank anonymous reviewers for their very useful comments and suggestions. The work of Yubao Liu, Zhijie Li, Cheng Chen, and Zhitong Chen are supported by the National Natural Science Foundation of China (Grant Nos. 60703111, 61070005, and 61033010), the Science and Technology Planning

Project of Guangdong Province of China (2010B080701062), and the Fundamental Research Funds for the Central Universities (11lgpy63). The research of Raymond Chi-Wing Wong is supported by HKRGC GRF 621309 and DAG11EG05G. Ke Wang's work is partially supported by a Discovery Grant from Natural Sciences and Engineering Research Council of Canada.

8 Appendix: Computation of (s_1, s_2)-circle, (s_2, s_2)-plane and s_3-circle on plane in a three-dimensional space

In Sect. 5.3, given three NLSs, namely s_1, s_2, and s_3 as shown in Fig. 10, we describe the concepts of the (s_1, s_2)-circle, the (s_2, s_2)-plane, and the s_3-circle on a plane. In this section, we describe how we compute these three concepts.

How to Compute (s_1, s_2)*-Circle:* In the following, we want to describe a method to compute the (s_1, s_2)-circle given two NLS s_1 and s_2.

Assume NLS s_1 centered at $o_1(x_1, y_1, z_1)$ with radius r_1 and NLS s_2 centered at $o_2(x_2, y_2, z_2)$ with radius r_2. Then, we would have the following equations.

$$(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 = r_1^2 \tag{1}$$
$$(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = r_2^2 \tag{2}$$

From Eqs. (1) and (2), we have: $-2x(x_1 - x_2) + (x_1^2 - x_2^2) - 2y(y_1 - y_2) + (y_1^2 - y_2^2) - 2z(z_1 - z_2) + (z_1^2 - z_2^2) = r_1^2 - r_2^2$ Next, we have

$$x(x_1 - x_2) + y(y_1 - y_2) + z(z_1 - z_2) = \frac{r_1^2 - r_2^2 + x_2^2 + y_2^2 + z_2^2 - x_1^2 - y_1^2 - z_1^2}{-2} \tag{3}$$

We assume that the (s_1, s_2)-circle c_{12} is centered at $o_{12}(x_{12}, y_{12}, z_{12})$ with radius r_{12}. In the following, we need to compute the coordinates of the three-dimensional point o_{12} and the radius r_{12}. Since the point o_{12} is along the line segment between o_1 and o_2, we can assume that $\overrightarrow{o_1 o_{12}} = \lambda \overrightarrow{o_1 o_2}$, $(0 < \lambda < 1)$. Then, we have

$$(x_1 - x_{12}, y_1 - y_{12}, z_1 - z_{12}) = \lambda(x_1 - x_2, y_1 - y_2, z_1 - z_2) \tag{4}$$

That is,

$$x_{12} = x_1 + \lambda(x_2 - x_1),\ y_{12} = y_1 + \lambda(y_2 - y_1),\ z_{12} = z_1 + \lambda(z_2 - z_1) \tag{5}$$

Since the point $o_{12}(x_{12}, y_{12}, z_{12})$ is on the plane α, we can put x_{12}, y_{12} and z_{12} in Eq. (5) into Eq. (3) (i.e., replacing x, y, z, respectively). Then, we have

$$\begin{aligned}&(x_1 + \lambda(x_2 - x_1))(x_1 - x_2) + (y_1 + \lambda(y_2 - y_1))(y_1 - y_2)\\ &+(z_1 + \lambda(z_2 - z_1))(z_1 - z_2)\\ &= \frac{r_1^2 - r_2^2 + x_2^2 + y_2^2 + z_2^2 - x_1^2 - y_1^2 - z_1^2}{-2}\end{aligned}$$

Next, we have $\lambda = \frac{r_1^2 - r_2^2}{2(x_2 - x_1)^2} + \frac{1}{2}$. Then, we put λ into Eq. (5) and compute the coordinates of the three-dimensional point o_{12}. That is,

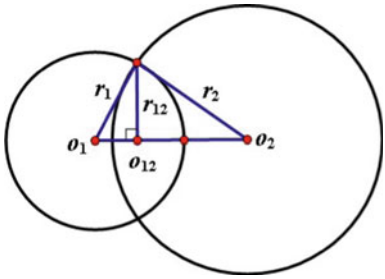


Fig. 20 A sectional drawing for s_1, s_2 and c_{12}

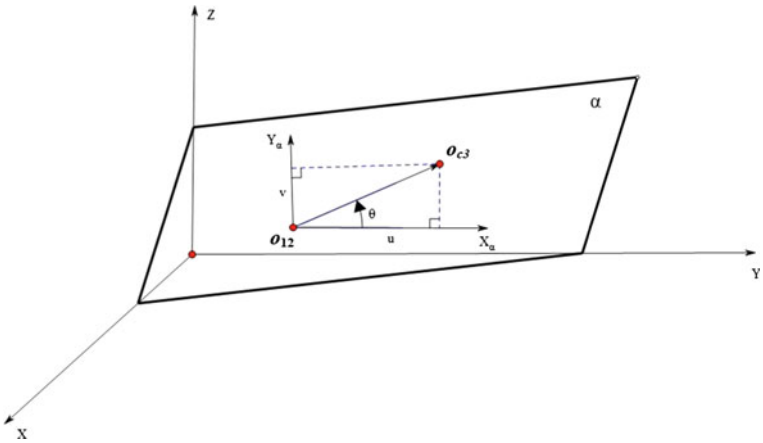


Fig. 21 The computation for the circle c_3

$$\begin{cases} x_{12} = x_1 + \left(\frac{r_1^2 - r_2^2}{2(x_2 - x_1)^2} + \frac{1}{2}\right)(x_2 - x_1) \\ y_{12} = y_1 + \left(\frac{r_1^2 - r_2^2}{2(x_2 - x_1)^2} + \frac{1}{2}\right)(y_2 - y_1) \\ z_{12} = z_1 + \left(\frac{r_1^2 - r_2^2}{2(x_2 - x_1)^2} + \frac{1}{2}\right)(z_2 - z_1) \end{cases} \tag{6}$$

where $o_1(x_1, y_1, z_1)$, $o_2(x_2, y_2, z_2)$, r_1, and r_2 are known.

The radius r_{12} can be computed as follows. As shown in Fig. 10a, we can easily image the following sectional drawing in Fig. 20 for NLS s_1, NLS s_2, and circle c_{12}. So, we have $r_{12} = \sqrt{r_1^2 - |o_1 o_{12}|^2} = \sqrt{r_1^2 - \lambda^2 |o_1 o_2|^2}$, where o_1, o_2, r_1 and λ are known. Thus, we derive the (s_1, s_2)-circle.

It is easy to verify that the above computation of the (s_1, s_2)-circle takes $O(1)$ time.

How to Compute (s_1, s_2)*-Plane and* s_3*-Circle:* In the following, we describe how to compute the (s_1, s_2)-plane, says α, and the s_3-circle on plane α, says c_3.

Assume that NLS s_3 is centered at $o_3(x_3, y_3, z_3)$ with radius r_3. Assume the center of circle c_3 in the three-dimensional space is $o_{c3}(x_{c3}, y_{c3}, z_{c3})$ and its radius is r_{c3}. From Fig. 10, it is easy to know that both points o_{12} and o_{c3} are on the plane α. We can build a coordinate system, $X_\alpha Y_\alpha$, on the two-dimensional plane α whose origin is o_{12}. The relationship between the three-dimensional space and the two-dimensional plane α can be shown in Fig. 21. From

Fig. 21, we can know the coordinates of point o_{c3} in the two-dimensional space (X_α, Y_α) correspond to u and v, respectively. Before computing u and v, we need to find the vectors $\overrightarrow{X_\alpha}$ and $\overrightarrow{Y_\alpha}$ that correspond to the axes of the coordinate system on the two-dimensional plane α. From Fig. 10a, we can know the normal vector $\overrightarrow{N} = \overrightarrow{o_2} - \overrightarrow{o_1}$ to the plane α. That is, we have $\overrightarrow{N} = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$. This normal vector can denote the (s_1, s_2)-plane.

Next, we construct two vectors as follows.

$$\begin{cases} X_\alpha = (y_2 - y_1, x_1 - x_2, 0) \\ Y_\alpha = \left(\frac{-(x_2-x_1)(z_2-z_1)}{(x_2-x_1)^2+(y_2-y_1)^2}, \frac{-(y_2-y_1)(z_2-z_1)}{(x_2-x_1)^2+(y_2-y_1)^2}, 1 \right) \end{cases} \tag{7}$$

It is easy to verify the vectors $\overrightarrow{X_\alpha}$, $\overrightarrow{Y_\alpha}$, and $\overrightarrow{N}$ are perpendicular to each of the other two vectors. Then, we can take $\overrightarrow{X_\alpha}$ and $\overrightarrow{Y_\alpha}$ as the axes of the coordinate system on the two-dimensional plane α. It is also easy to know that $\overrightarrow{o_{c3}o_3} = \varphi \overrightarrow{N}$, where φ is a real number and $\overrightarrow{N}$ is the normal vector to the plane α. Then, we have the following Eq. (8) in which x_N, y_N, and z_N denote the vector components of vector $\overrightarrow{N}$, respectively.

$$x_3 - x_{c3} = \varphi x_N, y_3 - y_{c3} = \varphi y_N, z_3 - z_{c3} = \varphi z_N \tag{8}$$

Since $\overrightarrow{o_{12}o_{c3}} \perp \overrightarrow{N}$,we have $(x_{c3} - x_{12})x_N + (y_{c3} - y_{12})y_N + (z_{c3} - z_{12})z_N = 0$. Next, we can replace x_{c3}, y_{c3} and z_{c3} using Eq. (8).

Then, we have $\varphi = \frac{x_N(x_3-x_{12})+y_N(y_3-y_{12})+z_N(z_3-z_{12})}{x_N^2+y_N^2+z_N^2}$. Next, we put φ into Eq. (8) and have $o_{c3}(x_{c3}, y_{c3}, z_{c3})$. That is,

$$\begin{cases} x_{c3} = x_3 - \varphi x_N \\ y_{c3} = y_3 - \varphi y_N \\ z_{c3} = z_3 - \varphi z_N \end{cases} \tag{9}$$

where x_3, y_3, z_3, x_N, y_N, z_N, and φ are known. From Fig. 21, we have u, v and the radius of c_3 as follows.

$$\begin{cases} u = |o_{12}o_{c3}| \cdot \cos\theta = \frac{\overrightarrow{o_{12}o_{c3}} \cdot \overrightarrow{X_\alpha}}{|X_\alpha|} \\ v = |o_{12}o_{c3}| \cdot \sin\theta = \frac{\overrightarrow{o_{12}o_{c3}} \cdot \overrightarrow{Y_\alpha}}{|Y_\alpha|} \\ r_{c3} = \sqrt{r_3^2 - |o_3o_{c3}|^2} \end{cases} \tag{10}$$

where r_3, o_3, o_{c3}, $\overrightarrow{X_\alpha}$ and $\overrightarrow{Y_\alpha}$ are known. Thus, we derive the s_3-circle.

It is easy to verify that the computation of the (s_1, s_2)-plane and the s_3-circle takes $O(1)$ time.

References

1. Beckmann N, Kriegel HP, Schneider R, Seeger B (1990) The R*-tree: an efficient and robust access method for points and rectangles. In: Garcia-Molina H, Jagadish HV (eds) Proceedings of the ACM SIGMOD international conference on management of data. Atlantic City, NJ, May 1990, pp 322–331
2. Berg M, Kreveld M, Overmars M, Schwarzkopf O (eds) (2000) Computational geometry: algorithms and applications. Springer, Berlin
3. Cabello S, Diaz-Banex JM, Langerman S, Seara C (2010) Facility location problems in the plane based on reverse nearest neighbor queries. Eur J Oper Res 202(1):99–106
4. Cabello S, Diaz-Banez JM, Langerman S, Seara C, Ventura I (2005) Reverse facility location problems. In: Proceedings of the 17th Canadian conference on computational geometry, Ontario, Canada, Aug 2005, pp 68–71

5. Cardinal J, Langerman S (2006) Min-max-min geometric facility location problems. In: Proceedings of the 22nd European workshop on computational geometry, Delphi, Greece, March 2006
6. Chazelle B (1986) New upper bounds for neighbor searching. Inf Control 68(1–3):105–124
7. Cheema MA, Lin X, Zhang W, Zhang Y (2011) Influence zone: efficiently processing reverse k nearest neighbors queries. In: Abiteboul S, Bohm K, Koch C (eds) Proceedings of the 27th international conference on data engineering. Hannover, Germany, April 2011, pp 577–588
8. Cheema MA, Lin X, Zhang W, Zhang Y, Wang W, Zhang W (2009) Lazy updates: an efficient technique to continuously monitoring reverse kNN. Proc VLDB Endow 2(1):1138–1149
9. Du Y, Zhang D, Xia T (2005) The optimal-location query. In: Medeiros CB, Egenhofer MJ, Bertino E (eds) Proceedings of the 9th international symposium on advances in spatial and temporal databases. Angra dos Reis, Brazil, Aug 2005, pp 163–180
10. Emrich T, Kriegel HP, Kröger P, Renz M, Xu N, Züfle A (2010) Reverse k-Nearest neighbor monitoring on mobile objects. In: Agrawal D, Abbadi AE, Mokbel MF (eds) Proceedings of the 18th ACM SIGSPATIAL international symposium on advances in geographic information systems. San Jose, CA, USA, Nov 2010, pp 494–497
11. Kang JM, Mokbel MF, Shekhar S, Xia T, Zhang D (2007) Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In: Chirkova R, Dogac A, Özsu MT, Sellis TK (eds) Proceedings of the 23rd international conference on data engineering. The Marmara Hotel, Istanbul, Turkey, April 2007, pp 806–815
12. Khoshgozaran A, Shahabi C, Shirani-Mehr H (2011) Location privacy: going beyond K-anonymity, cloaking and anonymizers. Knowl Inf Syst 26(3):435–465
13. Korn F, Muthukrishnan S (2000) Influence sets based on reverse nearest neighbor queries. In: Chen W, Naughton JF, Bernstein PA (eds) Proceedings of the ACM SIGMOD international conference on management of data. Dallas, Texas, USA, May 2000, pp 201–212
14. Korn F, Muthukrishnan S, Srivastava D (2002) Reverse nearest aggregates over data stream. In: Proceedings of the 28th international conference on very large data bases, Hong Kong, China, August 2002, pp 814–825
15. Krarup J, Pruzan PM (1983) The simple plant location problem: Survey and synthesis. Eur J Oper Res 12(1):36–57
16. Lian X, Chen L (2009) Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. VLDB J 18(3):787–808
17. Lu EHC, Lee WC, Tseng VS (2011) Mining fastest path from trajectories with multiple destinations in road networks. Knowl Inf Syst 29(1):25–53
18. Roussopoulos N, Kelley S, Vincent F (1995) Nearest neighbor queries. In: Carey MJ, Schneider DA (eds) Proceedings of the ACM SIGMOD international conference on management of data. San Jose, California, May 1995, pp 71–79
19. Stanoi I, Agrawal D, ElAbbadi A (2000) Reverse nearest neighbor queries for dynamic databases. In: Gunopulos D, Rastogi R (eds) Proceedings of 2000 ACM SIGMOD workshop on research issues in data mining and knowledge discovery. Dallas, Texas, USA, May 2000, pp 44–53
20. Stanoi I, Riedewald M, Agrawal D, Abbadi AE (2001) Discovery of influence sets in frequently updated databases. In: Apers PMG, Atzeni P, Ceri S, Paraboschi S, Ramamohanarao K, Snodgrass RT (eds) Proceedings of the 27th international conference on very large data bases. Roma, Italy, Sept 2001, pp 99–108
21. Tan JSF, Lu EHC, Tseng VS (2012) Preference-oriented mining techniques for location-based store search. Knowl Inf Syst. doi:10.1007/s10115-011-0475-4
22. Tansel BC, Francis RL, Lowe T (1983) Location on networks: a survey. Manag Sci 29(4):482–497
23. Tao Y, Papadias D, Lian X (2004) Reverse kNN search in arbitrary dimensionality. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley J, Schiefer KB (eds) Proceedings of the thirtieth international conference on very large data bases. Toronto, Canada, Sept 2004, pp 744–755
24. Tao Y, Yiu ML, Mamoulis N (2006) Reverse nearest neighbor search in metric spaces. IEEE Trans Knowl Data Eng 18(8):1239–1252
25. Vadapalli S, Valluri SR, Karlapalem P (2006) A simple yet effective data clustering algorithm. In: Clifton CW, Zhong N, Liu J, Wah BW, Wu X (eds) Proceedings of the 6th IEEE international conference on data mining. Hong Kong, China, Dec 2006, pp 1108–1112
26. Wong RCW, Özsu MT, Yu PS, Fu AWC, Liu L (2009) Efficient method for maximizing bichromatic reverse nearest neighbor. Proc VLDB Endow 2(1):1126–1137
27. Wong RCW, Özsu MT, Yu PS, Fu AWC, Liu L, Liu Y (2011) Maximizing bichromatic reverse nearest neighbor for Lp-norm in two- and three-dimensional spaces. VLDB J 20(6):893–919
28. Wong RCW, Tao Y, Fu AWC, Xiao X (2007) On efficient spatial matching. In: Koch C, Gehrke J, Garofalakis MN, Srivastava D, Aberer K, Deshpande A, Florescu D, Chan CY, Ganti V, Kanne CC, Klas

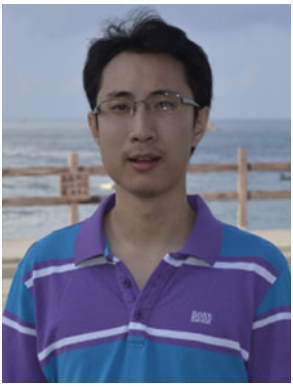
W, Neuhold EJ (eds) Proceedings of the 33rd international conference on very large data bases. University of Vienna, Austria, Sept 2007, pp 579–590
29. Wu W, Yang F, Chan CY, Tan KL (2008) FINCH: evaluating reverse k-nearest-neighbor queries on location data. Proc VLDB Endow 1(1):1056–1067
30. Wu W, Yang F, Chan CY, Tan KL (2008) Continuous reverse k-nearest-neighbor monitoring. In: Meng X, Lei H, Grumbach S, Leong HV (eds) Proceedings of the 9th international conference on mobile data management. Beijing, China, April 2008, pp 132–139
31. Xia T, Zhang D (2006) Continuous reverse nearest neighbor monitoring. In: Liu L, Reuter A, Whang K, Zhang J (eds) Proceedings of the 22nd international conference on data engineering. Atlanta, GA, USA, April 2006, p 77
32. Xia T, Zhang D, Kanoulas E, Du Y (2005) On computing top-t most influential spatial sites. In: Bohm K, Jensen C, Haas LM, Kersten ML, Larson P, Ooi BC (eds) Proceedings of the 31st international conference on very large data bases. Trondheim, Norway, Sept 2005, pp 946–957
33. Yang Y, Hao C (2011) Product selection for promotion planning. Knowl Inf Syst 29(1):223–236
34. Yiu ML, Papadias D, Mamoulis N, Tao Y (2006) Reverse nearest neighbors in large graphs. IEEE Trans Knowl Data Eng 18(4):540–553
35. Zhang D, Du Y, Xia T, Tao Y (2006) Progressive computation of the min-dist optimal-location query. In: Dayal U, Whang K, Lomet DB, Alonso G, Lohman GM, Kersten ML, Cha SK, Kim Y (eds) Proceedings of the 32nd international conference on very large data bases. Seoul, Korea, Sept 2006, pp 643–654
36. Zhang M, Alhajj R (2011) Effectiveness of NAQ-tree in handling reverse nearest-neighbor queries in high-dimensional metric space. Knowl Inf Syst 31(2):307–343
37. Zhang M, Alhajj R (2010) Effectiveness of NAQ-tree as index structure for similarity search in high dimensional metric space. Knowl Inf Syst 22(1):1–26
38. Zhang S, Chen F, Wu X, Zhang C (2006) Identifying bridging rules between conceptual clusters. In: Eliassi-Rad T, Ungar LH, Craven M, Gunopulos D (eds) Proceedings of the twelfth ACM SIGKDD international conference on knowledge discovery and data mining. Philadelphia, PA, USA, Aug 2006, pp 815–820
39. Zhou Z, Wu W, Li X, Lee ML, Hsu W (2011) MaxFirst for MaxBRkNN. In: Abiteboul S, Bohm K, Koch C, Tan K (eds) Proceedings of the 27th international conference on data engineering. Hannover, Germany, April 2011, pp 828–839
40. Zhu L, Li C, Tung AKH, Wang S (2012) Microeconomic analysis using dominant relationship analysis. Knowl Inf Syst 30(1):179–211

Author Biographies

Yubao Liu is currently an associate professor with the Department of Computer Science of Sun Yat-Sen University, China. He received his Ph.D. in computer science from Huazhong University of Science and Technology in 2003, China. He has published more than 40 refereed journal and conference papers. His research interests include database systems and data mining. He is also a member of the China Computer Federation (CCF) and the ACM.

Raymond Chi-Wing Wong received the BSc, MPhil and Ph.D. degrees in Computer Science and Engineering in the Chinese University of Hong Kong (CUHK) in 2002, 2004 and 2008, respectively. He joined Computer Science and Engineering of the Hong Kong University of Science and Technology as an Assistant Professor in 2008. His research interests include database, data mining and security.

Ke Wang received Ph.D. from Georgia Institute of Technology. He is currently a professor at School of Computing Science, Simon Fraser University. Ke Wang's research interests include database technology, data mining and knowledge discovery, with emphasis on massive datasets, graph and network data, and data privacy. Ke Wang has published in more than 100 research papers in database, information retrieval, and data mining conferences. He is currently an associate editor of the ACM TKDD journal.



Zhijie Li received his B.Eng in the Geography and Planning School of Sun Yat-Sen University of China in 2008. He is a graduate student of the Department of Computer Science of Sun Yat-Sen University, China. His research interests include databases and data mining.

Cheng Chen received his B.Eng in the Department of Computer Science of Sun Yat-Sen University of China in 2010. He is a graduate student of the Department of Computer Science of Sun Yat-Sen University, China. His research interests include databases and data mining.

Zhitong Chen received his B.Eng in the School of Mathematics and Computational Science of Sun Yat-Sen University of China in 2010. He is a graduate student of the Department of Computer Science of Sun Yat-Sen University, China. His research interests include databases and data mining.