The VLDB Journal (2011) 20:893–919
DOI 10.1007/s00778-011-0230-1

REGULAR PAPER

Maximizing bichromatic reverse nearest neighbor for L_p-norm in two- and three-dimensional spaces

Raymond Chi-Wing Wong · M. Tamer Özsu · Ada Wai-Chee Fu · Philip S. Yu · Lian Liu · Yubao Liu

Received: 23 August 2010 / Revised: 18 March 2011 / Accepted: 26 March 2011 / Published online: 12 April 2011

Abstract Bichromatic reverse nearest neighbor (BRNN) has been extensively studied in spatial database literature. In this paper, we study a related problem called MaxBRNN: find an optimal region that maximizes the size of BRNNs for L_p-norm in two- and three- dimensional spaces. Such a problem has many real-life applications, including the problem of finding a new server point that attracts as many customers as possible by proximity. A straightforward approach is to determine the BRNNs for all possible points that are not feasible since there are a large (or infinite) number of possible points. To the best of our knowledge, there are no existing algorithms which solve MaxBRNN for any L_p-norm space of two- and three-dimensionality. Based on some interesting properties of the problem, we come up with an efficient algorithm called MaxOverlap for to solve this problem. Extensive experiments are conducted to show that our algorithm is efficient.

Keywords Spatial databases · Indexing · Reverse nearest neighbor

This is an extended version of the paper that appeared in the International Conference on Very Large Databases, 2009.

Electronic supplementary material The online version of this article (doi:10.1007/s00778-011-0230-1) contains supplementary material, which is available to authorized users.

R. C.-W. Wong (✉) · L. Liu
The Hong Kong University of Science and Technology, Hong Kong, People's Republic of China
e-mail: raywong@cse.ust.hk

L. Liu
e-mail: lyanliu@cse.ust.hk

M. T. Özsu
University of Waterloo, Waterloo, Canada
e-mail: tamer.ozsu@uwaterloo.ca

A. W.-C. Fu
The Chinese University of Hong Kong, Hong Kong, People's Republic of China
e-mail: adafu@cse.cuhk.edu.hk

P. S. Yu
University of Illinois at Chicago, Chicago, USA
e-mail: psyu@cs.uic.edu

Y. Liu
Sun Yat-Sen University, Guangzhou, People's Republic of China
e-mail: liuyubao@mail.sysu.edu.cn

1 Introduction

Bichromatic reverse nearest neighbor (BRNN) search has been extensively studied as an important operator in spatial databases [11,12,22]. Let $\mathcal{P}$ and $\mathcal{O}$ be two sets of points in the same data space. Given a point $p \in \mathcal{P}$, a BRNN query finds all the points $o \in \mathcal{O}$ whose nearest neighbors (NN) in $\mathcal{P}$ are p, namely, there does not exist any other point $p' \in \mathcal{P}$ such that $|o, p'| < |o, p|$. A set of points o constitute the BRNN *set* (or simply BRNN) of p, denoted by BRNN$(p, \mathcal{P})$.

One of the typical applications of BRNN search is "selection of the best service". For example, we may want to find customers who would be more interested in visiting a convenience store based on their distances. Figure 1a shows the spatial layout of two convenience stores $\mathcal{P}$, namely p_1 and p_2, and five customers $\mathcal{O}$, namely o_1, o_2, o_3, o_4, and o_5. Suppose we want to know which customers are interested in a convenience store p_i. We obtain BRNN$(p_1, \mathcal{P}) = \{o_1, o_2\}$ and BRNN$(p_2, \mathcal{P}) = \{o_3, o_4, o_5\}$.

Next, consider that a new convenience store p_3 is to be set up and the company tries to find a location to maximize the number of customers who will go there. Suppose p_3 is

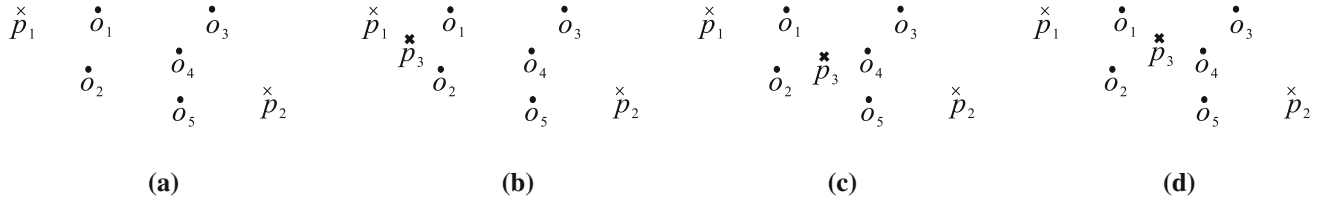


Fig. 1 An example

set at a location as shown in Fig. 1b. Then, p_3 can attract two customers, namely o_1 and o_2, which form the BRNN of p_3. But, suppose p_3 is set up as shown in Fig. 1c. Five customers, namely o_1, o_2, o_3, o_4 and o_5, are in the BRNN of p_3. In other words, different placements of p_3 give different numbers of customers who are interested in p_3. In this case, the placement of p_3 in Fig. 1c is better than that of Fig. 1b. The company should better set up convenience store p_3 as shown in Fig. 1c rather than the location in Fig. 1b. The problem is how to find an optimal location to attract the maximum number of customers.

The above problem can be formulated as follows: We distinguish two sets of points $\mathcal{O}$ and $\mathcal{P}$, where $\mathcal{O}$ is the set of client points and $\mathcal{P}$ is the set of server points. All points have a specific location in a Euclidean space. If a new point p is added to $\mathcal{P}$, we want to find a *region* R (or *area*) that maximizes the size of $p's$ BRNN. We call this problem MaxBRNN. MaxBRNN can be regarded as an optimal *region* search problem. In the convenience store example, $\mathcal{P}$ corresponds to the set of convenience stores and $\mathcal{O}$ corresponds to the customer set. MaxBRNN finds the region R (or area) such that, if a new convenience store p is set up in R, the size of $p's$ BRNN is maximized. Note that different placements of p in region R have the same BRNN. For example, suppose R is the optimal region. If the placements of p_3 as shown in Fig. 1c, d are in R, they have the same BRNN.

MaxBRNN can also be applied to traditional BRNN applications [12]. Service location planning problem and profile-based marketing are two examples. Our motivating example is a service location planning problem where a convenience store is regarded as a service and the objective is to find a location to open a new convenience store that can attract as many customers as possible. Other service location planning applications might be setting up coffee shops, fastfood restaurants, bank ATMs, gas stations, and wireless routers.

Most existing works on BRNN focus on finding the BRNN of a *given* point p. A naive adaptation of these techniques to MaxBRNN can be to find the BRNN of *all possible* placements. However, this adaptation is infeasible, because there are a *large* (or infinite) number of placements in the data space. Furthermore, it cannot summarize the region that the customers are interested in. MaxBRNN returns a *single* region in which every point has the same BRNN set. However, the naive adaptation returns a lot of points (in the corresponding region) that have the same BRNN set. Due to the same output (i.e., BRNN set) for different points, the computation in this adaptation involves a lot of redundant operations.

There is no *efficient* algorithm for MaxBRNN for L_2-norm for two-dimensional Euclidean space. There is only one work [4] that is closely related to ours that solves MaxBRNN in the L_2-norm for two-dimensional space and gives an algorithm whose running time is $O(|\mathcal{O}| \log |\mathcal{P}| + |\mathcal{O}|^2 + 2^{\gamma(|\mathcal{O}|)})$ where $\gamma(|\mathcal{O}|)$ is a function on $|\mathcal{O}|$ and is $\Omega(|\mathcal{O}|)$. Since the running time is exponential in terms of $|\mathcal{O}|$, this algorithm is not scalable with respect to dataset size.

In addition to everyday applications, MaxBRNN also applies in some emergency schedules (e.g., natural disaster, sudden big event, and military application). In a large scale natural disaster such as the earthquake in China, placing supply/service centers for rescue or relief jobs are important. In a big event like the US presidential campaign, placing police forces for security is also important. In a military application, it is essential to set up some temporary depots for gasoline and food. These applications involve road networks or physical transportation distance and therefore depending on the situation, the space could be two-dimensional or three-dimensional, and one can choose among L_1-norm, L_2-norm, or other metric space.

Different metrics are used in the spatial database literature. A popular one is the L_2-norm, which we have used to illustrated our problem. However, there can be applications where other metric spaces are found useful. In particular, Minkowski or L_p-norm metrics of different orders have been used. Among these, L_1 and L_2 norms are probably the most important metrics in spatial databases [17]. To the best of our knowledge, no algorithms exist that solve MaxBRNN with other Mindowski metrics.

In profile-based marketing [12], a company wants to set up a new service such as a car selling or stock selling service, and similarly, it wants to maximize the number of customers who are interested in this service. Here, each client point is considered as a client preference, while each server point is considered as a service. In the car application, there are some dimensions like (1) the number of passenger seats in a car, (2) the engine capacity of a car, and (3) the cargo volume. In the stock application, the dimensions can be different properties of stocks, such as return, volatility, and daily turn-over.

Since there would typically be multiple dimensions for a car, this problem will involve an n-dimensional metric space. There are other applications in an n-dimensional space where $n \geq 1$. Another such application is document repositories, where each client point is a document written by a given author while each server point is a document written by some other author. In this application, some authors would like to write a document that can attract the attention from other authors (due to similar topic interests). In the field of information retrieval (IR), a document is often treated as a high-dimensional vector containing a number of feature attributes. Some feature attributes can be different levels of the relevance of topics, such as "Spatial Databases", "Graph Queries", and "Privacy". The similarity measure studied in IR is also based on a distance function. Hence, we note that there are important applications which can benefit from MaxBRNN in different metric spaces and for n-dimensions where $n \geq 1$. To the best of our knowledge, there have been no works that study MaxBRNN in the three-dimensional space.

In this paper, we address these three issues. We propose an alternative algorithm called *MaxOverlap* that solves MaxBRNN in any Minkowski metric of order 1 or above and solves MaxBRNN much more efficiently in the L_2-norm space than the best-known algorithm. *MaxOverlap* finds the region that gives the maximum size of BRNN in $O(|\mathcal{O}| \log |\mathcal{P}| + k^2|\mathcal{O}| + k|\mathcal{O}| \log |\mathcal{O}|)$ time in the L_2-norm space where k can be regarded as an integer much smaller than $|\mathcal{O}|$. Compared with the algorithm mentioned above [4], our algorithm is much more efficient.

Intuitively, *MaxOverlap* is more efficient because it utilizes the principle of *region-to-point transformation*. It transforms the optimal *region* search problem to an optimal *point* search problem. In the point search problem, instead of searching all possible points in the space, *MaxOverlap* can search a *limited* number of points and find the optimal point efficiently. Finally, it can map the optimal point that it finds to the optimal region in the original problem. Since the total number of points considered in the *point* search problem is limited (more specifically, at most $2k|\mathcal{O}|$) while the total number of regions in the *region* search problem is exponential in terms of $|\mathcal{O}|$, *MaxOverlap* is more efficient than the existing algorithm [4] (which relies heavily on regions). Our experimental results show that *MaxOverlap* performs 1,000,000 times faster than the exponential-time algorithm in a dataset with 250 tuples. Our algorithm runs within 0.1 s but the exponential-time algorithm runs for more than 1 day over this dataset.

We also extend *MaxOverlap* to handle three-dimensional spaces and demonstrate that it can handle them as well as the two-dimensional spaces. This is the first attempt to tackle the more general problem of n-dimensional spaces for $n > 2$. However, how to address the problem for $n > 3$ is left for future work.

The rest of the paper is organized as follows. Section 2 formulates MaxBRNN problem in L_2-norm for two-dimensional spaces. Section 3 describes algorithm *MaxOverlap* for this problem and analyzes its performance. Sections 4 and 5 describe how *MaxOverlap* can be extended to any Minkowski metric of order one or above and also for the three-dimensional space. Section 6 evaluates the proposed techniques through extensive experiments with real data. Section 7 reviews the previous work and provides comparisons with our proposals. Section 8 concludes the paper with directions for future work.

2 Problem definition

Suppose we have a set $\mathcal{P}$ of *server points* in a space $\mathbb{D}$ (e.g., convenience stores in Fig. 1a). We also have another set $\mathcal{O}$ of *client points* in the same space. We denote a distance function between $p \in \mathcal{P}$ and $o \in \mathcal{O}$ in $\mathbb{D}$ by $|p, o|$. In this paper, the distance function we are studying is a metric (i.e., it satisfies the triangle inequality). In the following, for the sake of illustration, we first assume that the metric is the L_2-norm metric and the dimensionality to be considered is 2. In Sects. 4 and 5, we discuss extensions to any metric space and three-dimensional spaces, respectively.

Each client point o is a distinct location that is associated with a *weight*, $w(o)$, which corresponds to the number of clients at location o. For example, o is a residential estate and $w(o)$ is the total number of clients in this estate. Define $w_{\max} = \max_{o \in \mathcal{O}} w(o)$ as the greatest number of clients at a client point (or location).

We define a *region* to be an arbitrary shape in space $\mathbb{D}$. For example, Fig. 2a shows a spatial layout of two client points, namely o_1 and o_2, and two server points, namely p_1 and p_2. In Fig. 2a, R_1, R_2, and R_3 are three regions.

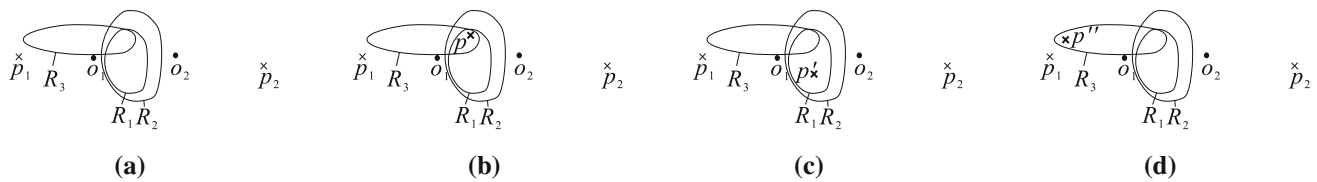


Fig. 2 Different regions with the same bichromatic reverse nearest neighbors

Definition 1 (*Consistent region*) A region R is said to be *consistent* if, for any two possible new server points p and p' in R, $\text{BRNN}(p, \mathcal{P} \cup \{p\}) = \text{BRNN}(p', \mathcal{P} \cup \{p'\})$.

A consistent region R contains all possible points p that have the same bichromatic reverse nearest neighbors. For example, if we start a new server point p as shown in Fig. 2b, $\text{BRNN}(p, \mathcal{P} \cup \{p\}) = \{o_1, o_2\}$. Similarly, starting a new server point p' at another location as shown in Fig. 2c has $\text{BRNN}(p', \mathcal{P} \cup \{p'\}) = \{o_1, o_2\}$. Figure 2d shows that we start a new server point p'' at another possible location, and we have $\text{BRNN}(p'', \mathcal{P} \cup \{p''\}) = \{o_1\}$.

Since any two possible points in R_1 (e.g., p in Fig. 2b and p' in Fig. 2c) have the same bichromatic reverse nearest neighbors, R_1 is a consistent region. Similarly, R_2 is a consistent region. However, R_3 is not a consistent region because there exists two possible new points p (Fig. 2b) and p'' (Fig. 2d) in R_3 such that $\text{BRNN}(p, \mathcal{P} \cup \{p\}) \neq \text{BRNN}(p'', \mathcal{P} \cup \{p''\})$.

Since a consistent region R contains all possible new server points p that have the same bichromatic reverse nearest neighbors, we define the *influence set* [12] of R as the bichromatic reverse nearest neighbor of any possible point p in R. This set denotes all client points that are interested in p.

Definition 2 (*Influence set/value*) Given a consistent region R, we define the *influence set* of R, denoted BRNN-$R(R)$, to be $\text{BRNN}(p, \mathcal{P} \cup \{p\})$ where p is any possible point inside R. The influence value of R, denoted $I(R)$, is equal to $\sum_{o \in \text{BRNN-}R(R)} w(o)$.

For instance, since R_1 is a consistent region, BRNN-$R(R_1) = \{o_1, o_2\}$. Similarly, for another consistent region R_2, BRNN-$R(R_2)$ is equal to $\{o_1, o_2\}$. When $w(o_1) = w(o_2) = 1$. $I(R_1) = I(R_2) = 2$.

A region R is said to *cover* another region R' if all areas of R' are inside R. For example, in Fig. 2a, R_2 covers R_1.

In Fig. 2a, in addition to R_2, there are other *arbitrary* consistent regions that cover R_1. Denoting all possible arbitrary consistent regions is not meaningful. Thus, we define a *maximal consistent region* as follows.

Definition 3 (*Maximal consistent region*) A consistent region R is said to be a *maximal consistent region* if and only if there does not exist another consistent region R' where (1) $R' \neq R$, (2) R' covers R, and (3) BRNN-$R(R)$ = BRNN-$R(R')$.

In Fig. 2a, region R_1 is not a maximal consistent region, because there exists another consistent region R_2 covering R_1 where BRNN-$R(R_1)$ = BRNN-$R(R_2)$.

In MaxBRNN, we would like to return the maximal consistent region R instead of any non-maximal consistent region, because R has the advantage of providing information that all possible points with the same BRNN sets are inside R. Returning non-maximal consistent region R' misses the information that some points outside R' have the same BRNN sets as BRNN-$R(R')$. Note that there are an exponential number of maximal consistent regions (in terms of $|\mathcal{O}|$), which makes the problem challenging.

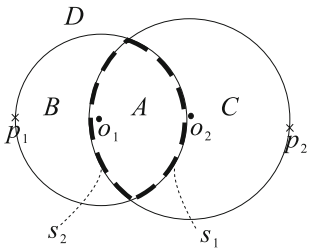


Fig. 3 Different regions where different client points are served

Problem 1 (*MaxBRNN*) Given a set $\mathcal{P}$ of server points and a set $\mathcal{O}$ of client points, we want 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. This problem is called MaxBRNN.

We return one maximal consistent region if there exist any ties. This problem has been shown to be 3SUM-hard [4] which means that if it takes $O(N)$ time to solve 3SUM (as conjectured) where N is the size of the dataset, then it must take at least quadratic time to solve MaxBRNN.

There are two challenges in this problem. The first challenge is that it is difficult to find a maximal consistent region because there are an infinite number of arbitrary consistent regions. The second challenge is that we need to return a maximal consistent region with the greatest influence value.

The first challenge can be addressed easily using a notion of *nearest location circles* that can be used to represent the maximal consistent regions. The second challenge will be addressed in Sect. 3.

Definition 4 (*Nearest location circle* (*NLC*)) Given a client point o, the *nearest location circle* (NLC) of o is defined as the circle centered at o with radius $|o, p|$ where p is the nearest neighbor of o in $\mathcal{P}$.

For example, Fig. 3 shows the same server points and client points as Fig. 2a. The nearest neighbor of o_1 in $\mathcal{P}$ is p_1, and the nearest neighbor of o_2 in $\mathcal{P}$ is p_2. Thus, the circle c_1 centered at o_1 with radius equal to $|p_1, o_1|$ is the nearest location circle of o_1 and the circle c_2 centered at o_2 with radius equal to $|p_2, o_2|$ is the nearest location circle of o_2. In the following, *we adopt a convention that an NLC centered at o_i is denoted by c_i*.

In Fig. 3, we have two NLCs, namely c_1 and c_2. The boundaries of these two NLCs partition the data space into four disjoint regions, namely regions A, B, C, and D. Region A is the region formed by the intersection between the region occupied by c_1 and the region occupied by c_2. Region B is the

region occupied by c_1 excluding the region occupied by c_2. Region C is the region occupied by c_2 excluding the region occupied by c_1. Region D is the region excluding the region occupied by c_1 and the region occupied by c_2.

Suppose a new server p is to be set up. If p is located inside circle c_1, then the nearest neighbor of o_1 in $\mathcal{P}$ will be changed from p_1 to p. Otherwise, p is not a nearest neighbor of o_1. By this reasoning, if p is inside multiple NLCs as shown in region A, then it will become a nearest neighbor of client points corresponding to these NLCs. It is easy to verify that regions A, B, C, and D are consistent and the influence set of regions A, B, C, and D are $\{o_1, o_2\}$, $\{o_1\}$, $\{o_2\}$ and $\{\}$, respectively. Since all of these influence sets are different, we can conclude that regions A, B, C, and D are maximal consistent regions. Note that, if $w(o_1) = w(o_2) = 1$, the influence values of regions A, B, C and D are $I(A) = 2$, $I(B) = 1$, $I(C) = 1$, and $I(D) = 0$, respectively.

In Fig. 3, it is easy to deduce that region A is the solution of MaxBRNN because the influence value of A is maximized. Note that this region is represented by an *intersection* of NLC c_1 and NLC c_2. In the following, we will show that the optimal solution of problem MaxBRNN is represented by an intersection of multiple NLCs only.

Lemma 1 (Intersection representation) *The region R returned by the MaxBRNN query can be represented by an intersection of multiple NLCs.*

Proof The proof can be found in [19]. □

Lemma 1 suggests that we need to consider only the region formed by the intersection of some NLCs (like region A) for MaxBRNN. We do not need to consider regions that are represented by some NLCs *excluding* some other NLCs. For example, by Lemma 1, region B should not be considered since it is represented by the region occupied by c_1 *excluding* the region occupied by c_2. Thus, this lemma reduces the search space significantly.

We are, naturally, interested in maximal consistent regions instead of non-maximal consistent regions. Therefore, any reference to regions from now on will mean maximal consistent regions.

We define some variations on MaxBRNN, namely Max$k$BRNN, lMaxBRNN, and lMax$k$BRNN, as follows.

1. **Max$k$BRNN:** k-BRNN of $p \in \mathcal{P}$, denoted by kBRNN$(p, \mathcal{P})$, is a set of points $o \in \mathcal{O}$ such that p is one of the k nearest neighbors of o in $\mathcal{P}$. In Max$k$BRNN, we want to find the *region* R (or *area*) such that, if a new server p is set up in R, the size of k-BRNN of p (i.e., $\sum_{o \in k\text{BRNN}(p, \mathcal{P} \cup \{p\})} w(o)$) is maximized. $I(R)$ is equal to $\sum_{o \in k\text{BRNN}(p, \mathcal{P} \cup \{p\})} w(o)$ in the setting with k-BRNN.
2. **lMaxBRNN:** Instead of finding *one* region that gives the greatest size of BRNN, we find l regions that give the greatest size of BRNN. Formally, let $\mathcal{R}$ be the set of all possible regions. We would like to return l regions $R \in \mathcal{R}$ with the greatest $I(R)$ values (with respect to BRNN).
3. **lMax$k$BRNN:** lMax$k$BRNN is a mixture of the above two problems. We would like to find l regions $R \in \mathcal{R}$ with the greatest $I(R)$ values (with respect to k-BRNN).

In the following, for the sake of illustration, we will first focus on MaxBRNN. Then, we will extend our discussion to the above variations of the problem.

3 Algorithm

In the L_2-norm space, the best-known algorithm [4] for MaxBRNN runs in exponential time because it heavily relies on searching the *regions* of which there are an exponential number. We propose an algorithm called *MaxOverlap* that utilizes the principle of *region-to-point transformation* and searches a limited number of *points*.

In this section, we present *MaxOverlap* algorithm that follows the properties given in Sect. 3.1. We also describe how *MaxOverlap* can be extended to problems Max$k$BRNN, lMaxBRNN, and lMax$k$BRNN.

3.1 Notation and properties

From Lemma 1, we know that the desired region R can be represented by an intersection of multiple NLCs. Based on this lemma, we propose an algorithm that first creates the data set $\mathcal{D}'$ containing NLCs from the original data set $\mathcal{D}$ containing points. Some NLCs can represent the *region* returned by the MaxBRNN query. Then, we perform the region-to-point transformation and solve the problem by searching a certain number of *points*.

Specifically, first, for each client point $o \in \mathcal{O}$, we find its nearest neighbor $p \in \mathcal{P}$, form an NLC c centered at o with radius equal to $|p, o|$ and insert it into $\mathcal{D}'$. In addition, we also set the weight of NLC c, denoted by $w(c)$, to be $w(o)$. Intuitively, $w(c)$ is the total number of clients at point (or location) o whose nearest location circle (NLC) is c. Then, based on the *overlapping* relationship between NLCs, we develop an algorithm that finds the region of MaxBRNN query efficiently. For instance, Fig. 4 shows six NLCs where we do not show server points in $\mathcal{P}$ for clarity. We denote each NLC centered at o_i by c_i for $i = 1, 2, \ldots, 6$. NLC c_3 overlaps NLCs c_1, c_2, and c_4. We also say that an NLC c_i *covers* another NLC c_j if all areas of c_j are inside c_i. For example, NLC c_3 covers NLC c_4.

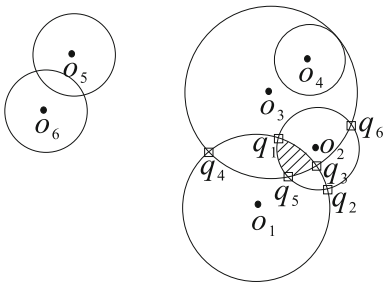


Fig. 4 Overlapping NLCs

Table 1 Overlap table

NLC	$L(c)$
c_1	c_2, c_3
c_2	c_1, c_3
c_3	c_1, c_2, c_4
c_4	c_3
c_5	c_6
c_6	c_5

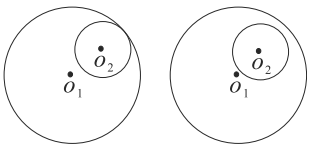


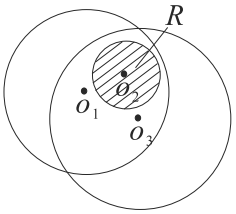
Fig. 5 Coverage relationship

The overlap relationship provides a key to the efficiency of our algorithm. The intuition is that the region of MaxBRNN query is the *intersection* of overlapping NLCs such that the total weight of overlapping NLCs is maximized. Motivated by this observation, for each NLC c, we find a list of NLCs that overlap with c, denoted by $L(c)$. Table 1 shows the list of overlapping NLCs of Fig. 4 and is called an *overlap table*. Each row of the overlap table is called an *entry* in the form of $(c, L(c))$. If the context of c is clear, we write it as (c, L).

We define two cases when an NLC c_1 covers another NLC c_2 as shown in Fig. 5. We say that NLC c_1 covers NLC c_2 *closely* if c_1 covers c_2 and the boundary of c_1 has an intersection point with the boundary of c_2 (Fig. 5a). We say that NLC c_1 covers NLC c_2 *disjointly* if c_1 covers c_2 but there is no intersection point between the boundary of c_1 and the boundary of c_2 (Fig. 5b). An NLC c is said to *cover* a *point* p if p is inside c or is along the boundary of c. For example, in Fig. 4, both NLC c_3 and NLC c_1 cover point q_1.

Lemma 2 (At least one intersection point) *If an NLC covers another NLC, the boundaries of the two NLCs must share at least one point* (*i.e., the NLC must cover another NLC closely*).

Fig. 6 Illustration of the proof of Lemma 3



Proof We prove by contradiction. Suppose that an NLC c_1 covers another NLC c_2 but c_1 and c_2 are disjoint as shown in Fig. 5b. Consider NLC c_1 centered at o_1. We know that there exists a server point $p_1 \in \mathcal{P}$ at the boundary of NLC c_1 such that p_1 is the nearest neighbor of o_1 in $\mathcal{P}$. Similarly, for c_2 centered at o_2, we also conclude that there exists a server point $p_2 \in \mathcal{P}$ at the boundary of NLC c_2 such that p_2 is the nearest neighbor of o_2 in $\mathcal{P}$. Since c_1 covers c_2 disjointly, we deduce that $|p_2, o_1| < |p_1, o_1|$. This leads to a contradiction that p_1 is the nearest neighbor in $\mathcal{P}$ from o_1. □

Let S be a set of NLCs. We define $W(S) = \sum_{c \in S} w(c)$. Let S_o be a set of NLCs whose intersection corresponds to the region R returned by a MaxBRNN query. Consider two cases. *Case 1:* S_o contains only one NLC, which means that the optimal solution comes from a *single* NLC *without* any overlap or intersection with *other NLCs*. It is easy to verify that $W(S_o) = w_{\max}$ where $w_{\max} = \max_{o \in \mathcal{O}} w(o)$. *Case 2:* S_o contains more than one NLC. In this case, the optimal solution is an intersection of more than one NLC and we derive that $W(S_o) > w_{\max}$. The following lemma shows an important property about overlapping NLCs.

Lemma 3 (Vantage point identification) *Let S_o be a set of NLCs whose intersection corresponds to region R returned by a MaxBRNN query. If S_o contains more than one NLC, then there exist two NLCs, say c_1 and c_2, such that region R contains (or covers) at least one intersection point between the boundaries of c_1 and c_2.*

Proof We prove by contradiction. Suppose there do not exist two NLCs, say c_1 and c_2, such that region R contains at least one intersection point between the boundary of c_1 and the boundary of c_2. We deduce that the shape of region R is an NLC c in S_o and all other NLCs in S_o cover c such that the boundary of c does not have any intersection with the boundaries of other NLCs in S_o. An example of this scenario is shown in Fig. 6 where the shaded region corresponds to R and c_2 corresponds to c. By Lemma 2, the above case is impossible since an NLC must not cover another NLC disjointly. This leads to a contradiction. □

For example, in Fig. 4, the shaded region corresponds to the region R returned by MaxBRNN query. R is formed by an intersection operation among NLCs in $S_o = \{c_1, c_2, c_3\}$. There exist two NLCs c_1 and c_2 such that R contains one intersection point between the boundary of c_1 and the boundary of c_2, say q_1.

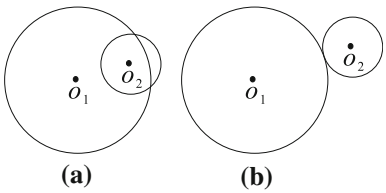


Fig. 7 Other overlapping relationships

From Lemma 3, we can observe that the optimal region R must contain one of the intersection points between at least one pair of NLCs. In other words, intersection points of some (or all) pairs of NLCs can be regarded as candidates in order to perform the MaxBRNN query. We call these intersection points *vantage points*. In the next subsection, we will describe how to make use of vantage points in evaluating a MaxBRNN query.

In addition to the coverage relationship as shown in Fig. 5a, there are two other kinds of overlapping relationships as shown in Fig. 7a, b. From the above three possible overlapping relationships, we observe that, for any two overlapping NLCs c_1 and c_2 in our problem setting, the boundary of c_1 and the boundary of c_2 intersect at least one point and at most two points.

Lemma 4 (Number of intersection points) *If NLC c_1 and NLC c_2 are overlapping, the number of intersection points between the boundary of c_1 and the boundary of c_2 is either one or two.*

3.2 Algorithm MaxOverlap

Based on Lemmas 3 and 4, we develop an algorithm, *MaxOverlap*, which takes $O(|\mathcal{O}|\log|\mathcal{P}|+k^2|\mathcal{O}|+k|\mathcal{O}|\log|\mathcal{O}|)$ time where k can be regarded as a small integer compared with $|\mathcal{O}|$. To the best of our knowledge, there are no existing polynomial-time algorithms for this MaxBRNN query, and *MaxOverlap* is the first polynomial-time algorithm for a MaxBRNN query.

The efficiency of *MaxOverlap* heavily depends on Lemma 3. The algorithm is designed based on the principle of *region-to-point transformation*. We transform the optimal *region* search problem into an optimal *vantage point* search problem where the optimal vantage point can subsequently be mapped into the optimal region. The vantage points for the search are derived from the intersection points among NLCs.

For example, in Fig. 4, the optimal region is the intersection of three NLCs, namely c_1, c_2, and c_3. Our algorithm starts to find a set of vantage points or intersection points between pairs of NLCs such as q_1, q_2, q_3, and q_4 (instead of regions or NLCs). These vantage points are used to determine the optimal region directly. Let S_q be a set of NLCs covering point q. The *influence value* of q is defined to be $\sum_{c\in S_q} w(c)$. If we can find an optimal vantage point (i.e., q with the largest influence value), the optimal region of MaxBRNN is equal to region R, which is the intersection of all NLCs in S_q. For example, in Fig. 4, if we can find the vantage point q_3 with the largest influence value $= 3$ (where $S_{q_3} = \{c_1, c_2, c_3\}$), the optimal region corresponds to the intersection of all NLCs in S_{q_3}.

Formally, we describe the algorithm as follows. Suppose S_o is a set of NLCs whose intersection corresponds to region R returned by a MaxBRNN query. As we mentioned before, it is easy to see that an NLC c with $w(c) = w_{\max}$ corresponds to an optimal solution if S_o contains only one NLC. Let us focus on finding a solution when S_o contains more than one NLC. We know that all NLCs among S_o are overlapping. We develop a three-step algorithm.

- *Step 1 (Finding intersection point):* We find a set Q of all intersection points between the boundaries of any two overlapping NLCs in the dataset. Details of this step can be found in Sect. 3.3.1.
- *Step 2 (Point query):* For each point $q \in Q$, we perform a point query for q to find a set S of NLCs covering q.
- *Step 3 (Finding maximum size):* We choose the set S obtained in the above step with the largest value of $W(S)$ as our final solution.

It is easy to verify that the final solution chosen in Step 3 is the optimal set S_o with the largest $W(S_o)$ value by Lemmas 3 and 4.

A straightforward implementation for this three-step algorithm is to perform these three steps one-by-one. For the first step, we compare all pairs of NLCs and check whether each pair are overlapping. If so, we find the intersection points of each such pair and insert them into a set Q. Note that there are $O(|\mathcal{O}|^2)$ pairs. For each pair, the checking and processing can be done in $O(1)$ time (these steps will be described in Sect. 3.3.1), Step 1 takes $O(|\mathcal{O}|^2)$ time. Note that $|Q| = O(|\mathcal{O}|^2)$. Then, for Step 2, we perform a point query for each $q \in Q$. Suppose $\beta(N)$ is the running time for a point query over dataset of size N, Step 2 takes $O(\beta(|\mathcal{O}|)\cdot|\mathcal{O}|^2)$ time. It is easy to verify that the running time of Step 3 is $O(|\mathcal{O}|^2)$. Thus, the total running time of this straightforward approach is $O(\beta(|\mathcal{O}|)\cdot|\mathcal{O}|^2)$. We will improve this running time to $O(|\mathcal{O}|\log|\mathcal{P}| + k^2|\mathcal{O}| + k|\mathcal{O}|\log|\mathcal{O}|)$ with some techniques that will be described next.

Although this straightforward approach can find an optimal solution, it is inefficient because Step 1 has to process *all possible pairs* of overlapping NLCs. In fact, some pairs of overlapping NLCs need not be considered and processed in Step 1 if there exists another pair whose intersection has a larger influence value and thus is a better choice as a solution. If we only process those "better" pairs of NLCs instead of all

possible pairs in Step 1, the computation can be improved. The following lemma points to this *influence-based pruning*.

Lemma 5 (Influence-based pruning) *Let I be a lower bound of the optimal influence value I_o (i.e., $I \leq I_o$). An optimal solution is a region that does not involve any NLC c where (c, L) is an entry for c in the overlap table T and $W(L) < I - w(c)$.*

Proof Suppose an optimal solution is a region involving NLC c. Consider an entry (c, L). NLC c overlaps at most $|L|$ NLCs. Let Q' be the set of all intersection points q between the boundaries of c and c' where $c' \in L$. We know that $q' \in Q'$ is covered by at most $|L| + 1$ NLCs (including c itself). The influence value I' of $q' \in Q'$ is at most $W(L) + w(c)$. That is, $I' \leq W(L) + w(c)$. Since an optimal solution is a region involving c, by Lemma 3, there exists one $q' \in Q'$ such that $I' \geq I$. For this q', we derive that $W(L) \geq I - w(c)$, which leads to a contradiction. □

With Lemma 5, we can safely remove all entries (c, L) from the overlap table T where $W(L) < I - w(c)$ if we know the lower bound I of the optimal influence value. For example, in Fig. 4, suppose that we know that the lower bound I is equal to 3 and $w(c) = 1$ for each NLC c. Then, entries for c_4, c_5 and c_6 in Table 1 are removed safely because their corresponding overlapping lists L in T have $W(L) = 1$, which is smaller than $I - w(c) = I - 1 = 3 - 1 = 2$.

In the following, we describe algorithm *MaxOverlap* (Algorithm 1) that *interleaves* the execution of these three steps and thus prunes a lot of candidate pairs.

First of all, the algorithm constructs an overlap table T. Let S_o and I_o be the variables that store the set of the optimal solution and the optimal influence value, respectively, that are found so far. Initially, S_o is set to the NLC c with the largest $w(c)$ value and I_o is set to $w(c)$. Then, each entry (c, L) is processed one-by-one. We introduce a variable A that stores a set of NLCs whose entries have been processed. A is initialized to $\emptyset$. The purpose of maintaining A is to avoid processing each pair of NLCs more than once.

Specifically, the algorithm iteratively performs the following steps. First, it removes an entry (c, L) with the largest $W(L)$ value from T for processing, since it is very likely that NLC c is involved in the optimal solution. For each $c' \in L - A$, it computes all intersection points between c and c'. For each intersection point q, it performs a point query to find all NLCs covering q. Let S be the result of this point query. If $W(S) > I_o$, then I_o is updated with $W(S)$ and S_o is updated with S. Next, c is inserted into A. Finally, the algorithm performs an influence-based pruning step to remove all unnecessary NLCs from consideration. This computation is repeated until all entries in T are exhausted.

Algorithm 1 Algorithm **MaxOverlap**

```
1: for each o ∈ O do
2:    construct an NLC for o
3: end for
4: build the overlap table T
5: choose the NLC c with the largest w(c)
6: S_o ← {c}
7: I_o ← w(c)
8: A ← ∅
9: while there exists an entry in T do
10:    remove an entry (c, L) with the largest value of W(L) from T
11:    for each NLC c' ∈ L − A do
12:       compute the intersection points between the boundary of c and
          the boundary of c' (See Section 3.3.1)
13:       for each intersection point q found do
14:          perform a point query from q to find all NLCs covering q
15:          let S be the result of the above point query
16:          I ← W(S)
17:          if I > I_o then
18:             S_o ← S
19:             I_o ← I
20:          end if
21:       end for
22:    end for
23:    A ← A ∪ {c}
24:    remove all entries (c', L') from T where W(L') < I_o − w(c')
25: end while
```

Example 1 (MaxOverlap) Consider the example shown in Fig. 4 and the overlap table shown in Table 1. Suppose $w(c) = 1$ for each NLC c. Firstly, we choose an NLC, say c_1, with the largest $w(c)$ value. Then, we initialize $I_o = 1$, $S_o = \{c_1\}$, and $A = \emptyset$.

Secondly, we perform the iterative steps as follows. We remove entry (c_3, L), where $L = \{c_1, c_2, c_4\}$, from the overlap table T since it has the largest $W(L)$ value. We then consider processing three possible pairs of NLCs between c_3 and an NLC in L, namely (c_3, c_1), (c_3, c_2), and (c_3, c_4). Take (c_3, c_1) for illustration. We compute the two intersection points between the boundary of c_3 and the boundary of c_1, namely q_4 and q_3 (as shown in Fig. 4).

Then, we perform a point query for q_4 and obtain result $S = \{c_3, c_1\}$, which covers q_4. Then, I is set to $W(S)$ (i.e., 2). Since $I > I_o$, we update $S_o = S = \{c_3, c_1\}$ and $I_o = I = 2$.

In addition to q_4, we also perform a point query for q_3. Similarly, we obtain $S = \{c_1, c_2, c_3\}$ which covers q_3 and $I = 3$. Similarly, since $I > I_o$, we update S_o and I_o to be $\{c_1, c_2, c_3\}$ and 3, respectively.

Next, A is updated to $\{c_3\}$. Since entries for c_4, c_5, and c_6 have their $L's$ such that $W(L) < I_o - w(c)$ (i.e., $1 < 3 - 1$), we perform an influence-based pruning step to remove these entries from the overlap table.

After processing entry (c_3, L), we continue with processing the entry (c_1, L), since it has the second greatest $W(L)$ value. S_o and I_o remain unchanged in this case, and A is updated to $A \cup \{c_1\} = \{c_1, c_3\}$. We do not need to remove any entries with the influence-based pruning in this iteration.

Since only entry (c_2, L) is left in the overlap table, we process it. Since $L - A = \emptyset$, this entry is skipped. No updates are necessary for S_o and I_o, and A is updated to $A \cup \{c_2\} = \{c_1, c_2, c_3\}$. The algorithm terminates.

The final answer can be found from S_o and I_o. That is, the optimal solution is the region that is a result of the intersection of $\{c_1, c_2, c_3\}$, and the influence value is 3.

With Lemma 5, it is easy to verify the following theorem.

Theorem 1 (Correctness) *Algorithm 1 returns the region R with the largest influence value* (*i.e., the optimal solution returned by the MaxBRNN query*).

3.3 Detailed steps

In this section, we first describe how we compute the intersection points between the boundaries of two NLCs. Then, we analyze the complexity of algorithm *MaxOverlap*.

3.3.1 Intersection point computation

In this section, we describe how we compute the intersection points between the boundaries of two NLCs c_1 and c_2. The boundary of an NLC c can be expressed as a mathematical equation. Consider the Cartesian coordinate system. Suppose that c_1 and c_2 are the NLC centered at coordinate (a, b) with radius r and the NLC centered at coordinate (c, d) with radius s, respectively. They can be expressed as $(x-a)^2+(y-b)^2 = r^2$ and $(x-c)^2 + (y-d)^2 = s^2$, respectively.

Suppose c_1 and c_2 overlap. By Lemma 4, we know that the boundaries of c_1 and c_2 have at most two intersection points, say q_1 and q_2. It is easy to derive q_1 and q_2 given the centers and the radii of c_1 and c_2 by elementary mathematical techniques. Note that the intersection point computation for q_1 and q_2 takes $O(1)$ time.

We summarize the above result with the following lemma. This lemma is used to help to explain some concepts in our extensions which will be discussed in Sects. 4 and 5.

Lemma 6 (Intersection point computation) *Given two overlapping NLCs c_1 and c_2, calculating the intersection points between the boundaries of c_1 and c_2 can be done in $O(1)$ time by elementary mathematical techniques.*

3.3.2 Complexity

Algorithm *MaxOverlap* has the following five detailed steps.

Step 1 (NLC construction): For each $o \in \mathcal{O}$, we perform a nearest neighbor query at o to find the nearest point p in $\mathcal{P}$ from o. Then, we create an NLC centered at o with radius $|o, p|$. Let $\alpha(N)$ be the running time of a nearest neighbor query over the dataset of size N. Since there are $|\mathcal{O}|$ data points in $\mathcal{O}$, Step 1 requires $O(|\mathcal{O}|\alpha(|\mathcal{P}|))$ time.

Nearest neighbor queries can be computed in logarithmic time. Since each nearest neighbor query over N two-dimensional data points can be solved in $O(\log N)$ time with an index that consumes $O(N)$ space (e.g., a *trapezoidal map* over the *Voronoi diagram* [9]), each nearest neighbor search over dataset $\mathcal{P}$ requires $O(\log |\mathcal{P}|)$. Thus, the total running time of this step is $O(|\mathcal{O}| \log |\mathcal{P}|)$.

In our implementation, we adopt the R^*-tree [2] which is available in commercial databases to support nearest neighbor queries. Although R^*-tree does not have good worst-case asymptotic performance, it has been shown to be fairly efficient in real cases and has been commonly adopted for nearest neighbor queries. Specifically, we build an R^*-tree $R_{\mathcal{P}}$ over all data points in $\mathcal{P}$ and then perform a nearest neighbor query for each $o \in \mathcal{O}$.

Step 2 (Overlap table construction): For each NLC c centered at o with radius r, we perform a range query from o with a radius r to find all NLCs that overlap with this range. Let L be the result of this range query excluding c. In the overlap table, we create an entry (c, L) for this NLC. Let $\beta(N)$ be the running time of a range query over dataset of size N. Since there are $|\mathcal{O}|$ NLCs, Step 2 requires $O(|\mathcal{O}|\beta(|\mathcal{O}|))$.

In the literature, $\beta(N)$ is theoretically bounded. Let k be the greatest result size of a range query (i.e., the greatest number of NLCs that overlap a given NLC). Since a range query can be executed in $O(k + \log |\mathcal{O}|)$ time [7], Step 2 can be done in $O(|\mathcal{O}|(k + \log |\mathcal{O}|))$.

For similar reasons, in our implementation, we also adopt the R^*-tree to support range queries. Specifically, we build another R^*-tree $R_{\mathcal{C}}$ over all NLCs that are created in the above step, and then perform a range query for each o.

Step 3 (Initialization): We initialize S_o, I_o and A, which takes $O(|\mathcal{O}|)$ time.

Step 4 (Entry Sorting): We sort all entries (c, L) in the overlap table T in descending order of $W(L)$, which takes $O(|\mathcal{O}| \log |\mathcal{O}|)$ time.

Step 5 (Iterative Step): We repeat the following steps until no entry remains in the overlap table T. We pick entry (c, L) with the greatest $W(L)$ value in T, which takes $O(1)$ time. Then, for each $c' \in L - A$, we perform the following substeps.

1. We compute the intersection points between the boundaries of c and c'. As shown in Sect. 3.3.1, the cost of computing the intersection points is $O(1)$.
2. For each intersection point q found in the above step, we perform a point query for q to find all NLCs covering q and obtain S. Let $\theta(N)$ be the running time of a point

query over a dataset of size N. A point query over dataset containing NLCs runs in $O(\theta(|\mathcal{O}|))$. Since there are at most two intersection points for one pair of NLCs, this sub-step requires $O(\theta(|\mathcal{O}|))$ time.

Recall that k is the greatest number of NLCs overlapping with a given NLC (i.e., the greatest size of L of an entry (c, L) in the overlap table T). Performing the above sub-steps requires $O(k\theta(|\mathcal{O}|))$ time. Since there are at most $|\mathcal{O}|$ entries in T, Step 5 takes $O(k|\mathcal{O}|\theta(|\mathcal{O}|))$ time.

With the techniques described in [6], $\theta(|\mathcal{O}|) = O(k + \log|\mathcal{O}|)$ and thus the running time of Step 5 is $O(k|\mathcal{O}|(k + \log|\mathcal{O}|))$.

Theorem 2 (Running time) *The running time of Algorithm MaxOverlap is* $O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + k|\mathcal{O}|\theta(|\mathcal{O}|) + |\mathcal{O}|\log|\mathcal{O}|)$ *where k is the greatest size of L of an entry* (c, L) *in the overlap table* T.

MaxOverlap algorithm makes use of nearest neighbor search, range search and point search, which is an important feature since it opens the opportunity of leveraging the rich literature on these topics to optimize *MaxOverlap*. For example, since $\alpha(|\mathcal{P}|)$ can be accomplished in $O(\log|\mathcal{P}|)$ [9], $\beta(|\mathcal{O}|)$ can be done in $O(k + \log|\mathcal{O}|)$ [7], and $\theta(|\mathcal{O}|)$ can be achieved in $O(k + \log|\mathcal{O}|)$ [6], the running time can be simplified to $O(|\mathcal{O}|\log|\mathcal{P}| + k^2|\mathcal{O}| + k|\mathcal{O}|\log|\mathcal{O}|)$.

MaxOverlap has to store (1) the R^*-tree $R_\mathcal{P}$ built over all points in $\mathcal{P}$, (2) the R^*-tree $R_\mathcal{C}$ built over all NLCs and (3) the overlap table. Let the sizes of $R_\mathcal{P}$ and $R_\mathcal{C}$ be $|R_\mathcal{P}|$ and $|R_\mathcal{C}|$, respectively. In the overlap table, there are at most $|\mathcal{O}|$ entries and each entry has size at most $O(k)$. The storage of the overlap table is $O(k|\mathcal{O}|)$. The space cost of our algorithm is equal to $O(|R_\mathcal{P}| + |R_\mathcal{C}| + k|\mathcal{O}|)$.

3.4 Extension to Max$k$BRNN, lMaxBRNN and lMax$k$BRNN

Up to now, we have discussed how *MaxOverlap* solves the MaxBRNN problem. Here we discuss how *MaxOverlap* can be extended to problems Max$k$BRNN, lMaxBRNN, and lMax$k$BRNN. We can simply focus on lMax$k$BRNN since it is the most general problem among these problems.

The adaptation of *MaxOverlap* is straightforward. We only need to make two modifications in the algorithm. Firstly, we construct an NLC according to the k-th nearest neighbor of o in $\mathcal{P}$ rather than according to the nearest neighbor of o in $\mathcal{P}$ (See lines 1–2 in Algorithm 1). Secondly, we maintain l regions with the greatest influence values, rather than maintaining only one region with the greatest influence value (See lines 16–18 in Algorithm 1).

4 Extension to other L_p norms

We have so far considered L_2-norm. However, there are also real-life applications for MaxBRNN in other metrics. One example is the Manhattan city distance or the L_1-norm space. L_2-norm is in fact a special case of L_p-norm or Minkowski distance, which is a metric on Euclidean space.

Given two points in a n-dimensional Euclidean space, $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$. The Minkowski distance of order p between x and y is defined as:

$$d_p(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

The Minkowski distance of order p corresponds to L_p-norm. Minkowski distance is typically used with p being 1 or 2. The latter is the Euclidean distance, while the former is the Manhattan distance. The case of $p = \infty$ is also known as the Chebyshev distance.

In this section, we discuss how *MaxOverlap* can be used with any Minkowski distance of order 1 or above. *MaxOverlap* that was designed for the L_2-norm for two-dimensional Euclidean space cannot be extended directly. This is because with the L_2-norm in 2-dimentsional Euclidean space, there are *at most two* intersection points between two nearest location circles, but with an arbitrary Minkowski metric, it is possible that there are an *infinite* number of intersection points between two nearest location regions (which will be discussed later in this section). We propose some novel techniques to handle this issue.

This section is organized as follows. In Sect. 4.1, we introduce the concept of *nearest location region* (NLR) for Minkowski distance of order 1 or above. A NLR can be considered as a generalization of nearest location circles defined for the L_2-norm. Section 4.2 describes how we extend *MaxOverlap* to handle problem MaxBRNN for Minkowski distance of order 1 or above.

4.1 Nearest location region

In the L_2-norm space, the fundamental concepts used in *MaxOverlap* is nearest location *circles* (NLCs). These circles are very important because they can be used to represent maximal consistent region (according to Lemma 1). Intuitively, *MaxOverlap* finds a set of NLCs the intersection of which gives the greatest influence value.

Similar to the L_2-norm space, in an arbitrary L_p-norm metric space $\mathbb{D}$, we define a fundamental concept called a *nearest location region* (NLR). Similarly, these regions can also be used to represent maximal consistent region. Then, *MaxOverlap* can be re-used. In an arbitrary space $\mathbb{D}$, *MaxOverlap* finds a set of NLRs whose intersection gives the greatest influence value.

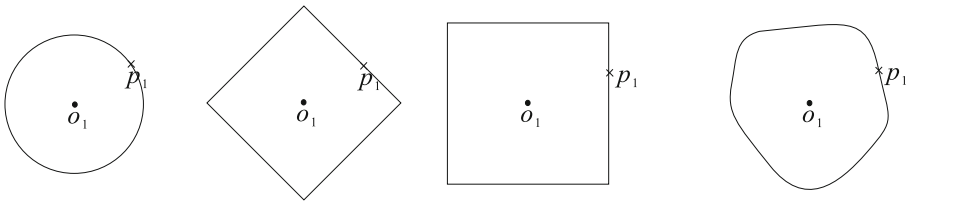


Fig. 8 Nearest location region (NLR) for different Minkowski metrics

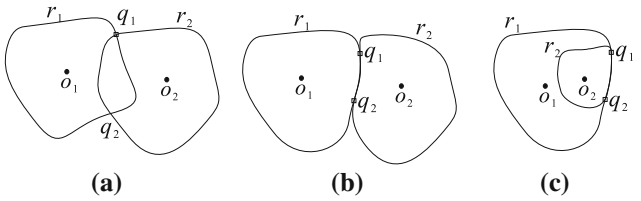


Fig. 9 Illustration of vantage points for any Minkowski distance metric

Definition 5 (*Nearest location region (NLR)*) Given a client point o, and consider Minkowski distance of order s, let p be a point in $\mathcal{P}$ nearest to o. The *nearest location region* (NLR) of o is defined to be a region $R = \{q : d_s(o, q) \leq d_s(o, p)\}$. For each point q along the boundary of the region R, $d_s(o, q) = d_s(o, p)$.

For $s = 2$, the NLR of a client point o is a circle (as discussed in Sect. 2) (See Fig. 8a). For $s = 1$ or L_1-norm, the NLR is a rhombus (or more precisely, a square rotated 45° clockwise) (See Fig. 8b). For $s = \infty$ or the L_∞-norm space, the NLR is a square (See Fig. 8c).

4.2 Extension

Though the NLR can be seen as a counterpart for the NLC for L_2-norm, the *shapes* of NLRs for Minkowski distances of different orders vary. Due to the variation in this shape, some results discussed before can be re-used but some cannot.

It is relatively easy to see that Lemmas 1, 2, 3, and 5 can be rewritten by replacing NLCs with NLRs. For convenience, we re-state these four lemmas as Lemmas 7, 8, 9 and 10, respectively.

Lemma 7 (Intersection representation) *The region R returned by the MaxBRNN query for any Minkowski metric of order 1 or above can be represented by an intersection of multiple NLRs.*

Lemma 8 (At least one intersection point) *If an NLR covers another NLR, the boundaries of the two NLRs must share at least one point (i.e., the NLR must cover another NLR closely).*

Lemma 9 (Vantage point identification) *Let S_o be a set of NLRs whose intersection corresponds to region R returned by a MaxBRNN query. If S_o contains more than one NLR, then there exist two NLRs, say r_1 and r_2, such that region R contains (or covers) at least one intersection point between the boundaries of r_1 and r_2.*

Lemma 10 (Influence-based pruning) *Let I be a lower bound of the optimal influence value I_o (i.e., $I \leq I_o$). An optimal solution is a region that does not involve any NLR r where (r, L) is an entry for r in the overlap table T and $W(L) < I - w(r)$.*

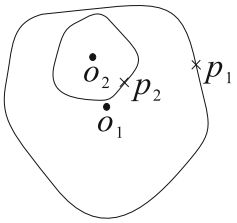
However, Lemmas 4 and 6 cannot be re-stated directly. Consider the L_2-norm metric. Lemma 4 states that there are *at most two* intersection points when two NLCs overlap while Lemma 6 describes how to compute intersection points of two NLCs and its time complexity. In general, for a Minkowski metric, there can be *a lot of* (up to *an infinite number of*) intersection points when two NLRs overlap. For example, Fig. 9a shows two overlapping NLRs r_1 and r_2 where there are only two intersection points. However, Fig. 9b shows two overlapping NLRs r_1 and r_2 where there is a continuous curve (from q_1 to q_2) such that each point along this curve is an intersection point. Thus, there are an infinite number of intersection points along this curve. Similarly, in Fig. 9c, there are an infinite number of intersection points along the curve from q_1 to q_2.

Recall that *MaxOverlap* includes a step that finds all intersection points (what we called vantage points) for problem MaxBRNN. If there are a lot of intersection points, *MaxOverlap* will be very inefficient because it has to compute all of them. In the following, we modify the definition of vantage points in order to improve the efficiency of *MaxOverlap* in any L_p-norm metric space. The major idea is based on the following *End Point Principle*.

Principle 1 (End point principle) *Given a pair of nearest location regions r_1 and r_2, instead of using all intersection points between the boundaries of r_1 and r_2, we choose only the end points of each continuous curve representing intersection points for performing point query in MaxOverlap. Such end points are called the vantage points.*

The above principle suggests that it is sufficient to just use the end points (instead of all intersection points) to find the optimal solution to MaxBRNN in a metric space. We will later show how *MaxOverlap* algorithm can return the optimal solution to MaxBRNN following this principle.

Fig. 10 Proof of Lemma 11



For example, in Fig. 9a, since q_1 and q_2 are the only two intersection points between two overlapping NLRs r_1 and r_2, there are only two end points, namely q_1 and q_2. According to the above principle, *MaxOverlap* can perform point query according to each of these two points. Figure 9b shows that two overlapping NLRs r_1 and r_2 have a continuous curve (from q_1 to q_2) such that each point along this curve is an intersection point. q_1 and q_2 are two end points of this curve. Similarly, in Fig. 9c, q_1 and q_2 are two end points of the curve representing the intersection points between two NLRs r_1 and r_2.

Definition 6 (*Vantage point for Minkowski metric*) Given a pair of nearest location regions r_1 and r_2, we define the *vantage points* between r_1 and r_2 to be all *end points* along each continuous segment (or curve) such that each point along this segment is an intersection point.

For example, in Fig. 9a, q_1 and q_2 are the only two vantage points between r_1 and r_2. In Figs. 9b, c, q_1 and q_2 are also the only two vantage points between r_1 and r_2.

The question is whether given two nearest location regions which overlaps, there exists a continuous curve representing intersection points between the boundaries of these two nearest location regions such that it has two end points? The answer is affirmative.

Lemma 11 *Given two overlapping NLRs, there exists a continuous curve representing intersection points between the boundaries of these two NLRs such that it has two end points.*

Proof We prove by contradiction. Suppose that there does not exist a continuous curve (with two end points) representing intersection points between the boundaries of the two overlapping nearest location regions. Then, one nearest location region covers another nearest location region disjointly (See Fig. 10). Similar to the proof in Lemma 2, it can be concluded that this scenario is not possible. This leads to a contradiction.

In the following lemma (Lemma 12), we show that it is sufficient to just use the vantage points as defined in Definition 6 (instead of all intersection points) to find the optimal solution of problem MaxBRNN for any Minkowski metric. After that, we will show how algorithm *MaxOverlap* following the End Point Principle can return the optimal solution of problem MaxBRNN.

Lemma 12 (Vantage point identification) *Let S_o be a set of NLRs whose intersection corresponds to region R returned by a MaxBRNN query. If S_o contains more than one NLR, then there exist two NLRs, say r_1 and r_2, such that region R contains at least one vantage point between r_1 and r_2.*

Proof We prove by contradiction. Suppose there do not exist two NLRs, say r_1 and r_2, such that region R contains at least one vantage point between r_1 and r_2. By Lemma 9, we know that there exists two NLRs, say r_1 and r_2, such that region R contains at least one intersection point q between the boundary of r_1 and the boundary of r_2 that is *not* equal to any vantage point between r_1 and r_2. Without loss of generality, we assume that q appears along the continuous segment/curve between two end points q_1 and q_2 representing intersection points. Note that q_1 and q_2 are two of the vantage points between r_1 and r_2. For example, consider Fig. 11a showing two NLRs, namely r_1 (centered at o_1) and r_2 (centered at o_2). q is along the curve segment between q_1 and q_2.

Suppose the optimal region R is denoted by the shaded region. There are four possible cases.

- *Case 1:* R covers q_1 and q_2 (e.g., Fig. 11b)
- *Case 2:* R covers q_1 but not q_2 (e.g., Fig. 11c)
- *Case 3:* R covers q_2 but not q_1 (e.g., Fig. 11d)
- *Case 4:* R does not cover q_1 or q_2 (e.g., Fig. 11e)

However, Case 1, Case 2, and Case 3 are impossible because we assume that R does not cover any vantage points where q_1 and q_2 are vantage points. Thus, Case 4 is the only possible case where R does not cover q_1 and q_2. We consider two sub-cases.

Case (a): q appears along the boundary of R (e.g., Fig. 11f). Note that q is along the boundary of R and q is along a continuous segment between q_1 and q_2. The reason why q occurs along the *boundary* of R is that there exists another NLR r_3 covering q such that the boundary of r_3 intersects with the continuous segment between q_1 and q_2. It is easy to verify that q is a vantage point between r_3 and r_1 (or r_2). Figure 11g shows an example that such a rhombus r_3 exists. Thus, R covers a vantage point, which leads to a contradiction.

Case (b): q does not appear along the boundary of R (e.g., the case shown in Fig. 11h). In this case, there exists another point q' that appears along the boundary of R (with the same influence value). This case is similar to Case (a), and R covers a vantage point q', which leads to a contradiction. □

With Lemma 12, it is easy to prove the following theorem.

Theorem 3 (Correctness) *Algorithm MaxOverlap following Principle 1 returns R with the largest influence value (i.e., the optimal solution returned by the MaxBRNN query).*

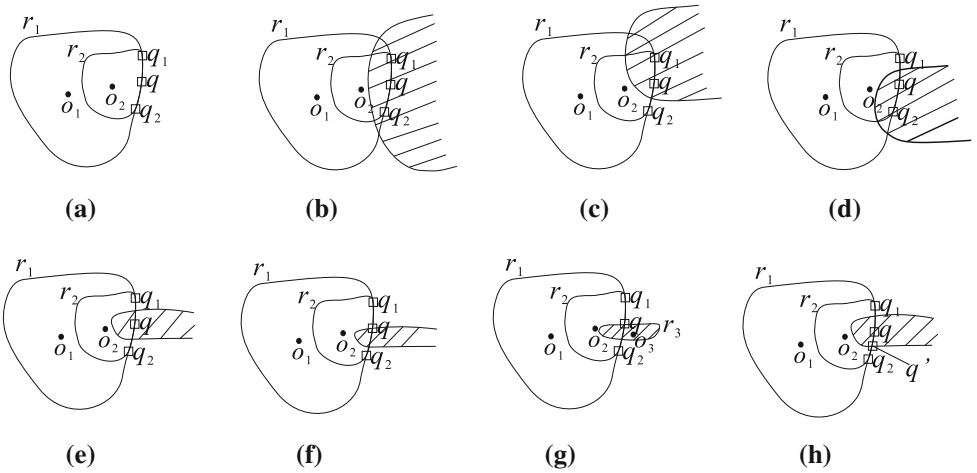


Fig. 11 Illustration of the proof of Lemma 12

Next, we analyze the complexity of *MaxOverlap* for a Minkowski metric of order s.

Similarly, we have the following five detailed steps.

- Step 1 (NLR (or NLC) Construction),
- Step 2 (Overlap Table Construction),
- Step 3 (Initialization),
- Step 4 (Entry Sorting) and
- Step 5 (Iterative Step).

Recall that in the L_2-norm space, Step 1 constructs NLC. For an arbitrary metric space, *MaxOverlap* can construct NLRs using an VP-tree [8]. Similar to Sect. 3.3, we can derive that the time complexity of Step 1 is $O(|\mathcal{O}|\alpha(|\mathcal{P}|))$ where $\alpha(N)$ is the running time of a nearest neighbor query over the dataset of size N in the given metric space.

In Step 2, we follow a similar step and compute the time complexity to be $O(|\mathcal{O}|\beta(|\mathcal{O}|))$ where $\beta(N)$ is the running time of a range query over dataset of size N in the given metric space. Note that the VP-tree requires some adaptions in order to index the NLRs as the data objects.

Step 3 and Step 4 are the same as before. Their complexities are $O(|\mathcal{O}|)$ and $O(|\mathcal{O}|\log|\mathcal{O}|)$, respectively.

Step 5 is the same as before. That is,

–*Step (i):* For each pair of NLRs, namely r_1 and r_2, we compute the vantage points between r_1 and r_2 (instead of intersection points). Note that for Minkowski distance, the boundary of each NLR can be formulated by a set of polynomial equations, and solving for such equations gives us the intersecting points.
–*Step (ii):* For each vantage point q found, we perform a point query for q on the VP-tree used in Step 2.

Let X be the running time of computing the vantage points between two NLRs in $\mathbb{D}$, and let k be the greatest number of NLRs which overlap with a given NLR. Since there are at most $O(k|\mathcal{O}|)$ pairs of NLRs, Step (i) takes $O(k|\mathcal{O}|X)$ time.

Consider Step (ii). The complexity analysis is similar to Sect. 3.3. Let n be the greatest possible number of vantage points between two NLRs in $\mathbb{D}$. Let $\theta(N)$ be the running time of a point query over a dataset of size N in the given L_p-metric space. Thus, since there are at most $O(k|\mathcal{O}|n)$ vantage points, Step (ii) takes $O(k|\mathcal{O}|n\theta(|\mathcal{O}|))$ time.

Combining Step (i) and Step (ii), we deduce that Step 5 takes $O(k|\mathcal{O}|X + k|\mathcal{O}|n\theta(|\mathcal{O}|))$ time.

Theorem 4 (Running time wrt $\mathbb{D}$) *The running time of Algorithm 1 in $\mathbb{D}$ is $O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + |\mathcal{O}|\log|\mathcal{O}| + k|\mathcal{O}|X + k|\mathcal{O}|n\theta(|\mathcal{O}|))$ where k is the greatest size of L of an entry (c, L) in the overlapping table T.*

The above complexity analysis is very general and can be applied to all L_p-metric spaces. For example, consider the L_2-norm metric. Since there are at most two vantage points between two given nearest location circles (or regions), $n = 2$. The computation of vantage points between two NLCs takes $O(1)$ time. Thus, X is equal to $O(1)$ time. The above complexity then reduces to $O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + |\mathcal{O}|\log|\mathcal{O}| + k|\mathcal{O}| \cdot 1 + k|\mathcal{O}| \cdot 2 \cdot \theta(|\mathcal{O}|)) = O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + k|\mathcal{O}|\theta(|\mathcal{O}|) + |\mathcal{O}|\log|\mathcal{O}|)$.

Note that finding intersection points between two nearest location regions is a problem that has been studied in geometry. However, MaxBRNN is not just simply equal to the problem of finding intersection points. In our proposed method called MaxOverlap for MaxBRNN, we first find the set of all *vantage points* (which is a subset of all intersection points between any two nearest location regions) and then perform a point query according to each vantage found in order to obtain the optimal region from the results from all point queries. The challenge in this paper is the proof of the correctness of algorithm MaxOverlap using only the "vantage" points instead of all intersection points, which has not been studied in the literature.

Next, we illustrate how the time complexity can be applied in the other two common metric spaces, namely the L_1-norm space (Sect. 4.2.1) and in the L_∞-norm space (Sect. 4.2.2).

4.2.1 L_1-norm space

In the above discussion, of the theoretical time complexity of algorithm *MaxOverlap*, n is the greatest number of vantage points between two NLRs. We will show that n is bounded (specifically, by $O(1)$) in the L_1-norm space.

The nearest location regions in the L_1-norm space are rhombi. Let us illustrate some vantage points with the L_1-norm metric. Consider Fig. 12a showing two rhombi r_1 (centered at o_1) and r_2 (centered at o_2). There is only one continuous (line) segment between q_1 and q_2 such that each point along this segment is an intersection point. The end points along this segment are q_1 and q_2. Thus, they are the vantage points between r_1 and r_2. Consider Fig. 12d showing another two rhombi r_1 and r_2. Since r_1 and r_2 have only two intersection points, namely q_1 and q_2, the vantage points between r_1 and r_2 are these two points.

n is bounded for the L_1-norm. Specifically, n is at least one and is at most four. This can be verified by listing all possible cases where two rhombi overlap as shown in Fig. 12a–i (Note: We do not show any symmetric cases). Thus, we have the following lemma.

Lemma 13 (At most four vantage points) *If rhombus r_1 and rhombus r_2 overlap, the number of vantage points between r_1 and r_2 ranges from* 1 *to* 4.

We now consider the time complexity of *MaxOverlap* for the L_1-norm metric. By Lemma 13, we know that the greatest number of vantage points between two NLRs (n) is 4. As for L_2-norm, we can compute vantage points in $O(1)$ time. Thus, $X = O(1)$. Thus, we have the following theorem about the running time of algorithm *MaxOverlap* for L_1-norm.

Theorem 5 (Running time wrt L_1-norm) *The running time of MaxOverlap for L_1-norm is $O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + |\mathcal{O}|\log|\mathcal{O}| + k|\mathcal{O}|\theta(|\mathcal{O}|))$ where k is the greatest size of L of an entry (c, L) in the overlapping table T.*

Similarly, since $\alpha(|\mathcal{P}|)$ can be accomplished in $O(\log|\mathcal{P}|)$ [9], $\beta(|\mathcal{O}|)$ can be done in $O(k+\log|\mathcal{O}|)$ [7] and $\theta(|\mathcal{O}|)$ can be achieved in $O(k + \log|\mathcal{O}|)$ [6], the running time can be simplified to $O(|\mathcal{O}|\log|\mathcal{P}| + k^2|\mathcal{O}| + k|\mathcal{O}|\log|\mathcal{O}|)$.

4.2.2 L_∞-norm space

The case in the L_∞-norm space is similar to the case in the L_1-norm space. In the L_∞-norm space, the nearest location region of each $o \in \mathcal{O}$ is a *square* centered at o instead of a circle or a rhombi. Similar to the L_1-norm space, the number of vantage points is at most four for each pair of squares. We can easily derive the following theorem about the running time complexity of MaxOverlap. In the L_1-norm space, we use notations α, β, and γ. In the L_∞-norm space, we overload notations α, β, and γ to represent the same concepts.

Theorem 6 (Running time wrt L_∞-norm) *The running time of Algorithm 1 in the L_∞-norm space is $O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + |\mathcal{O}|\log|\mathcal{O}| + k|\mathcal{O}|\theta(|\mathcal{O}|))$ where k is the greatest size of L of an entry (c, L) in the overlapping table T.*

Similarly, using the techniques in [6,7,9], we can simplify the running time to $O(|\mathcal{O}|\log|\mathcal{P}| + k^2|\mathcal{O}| + k|\mathcal{O}|\log|\mathcal{O}|)$.

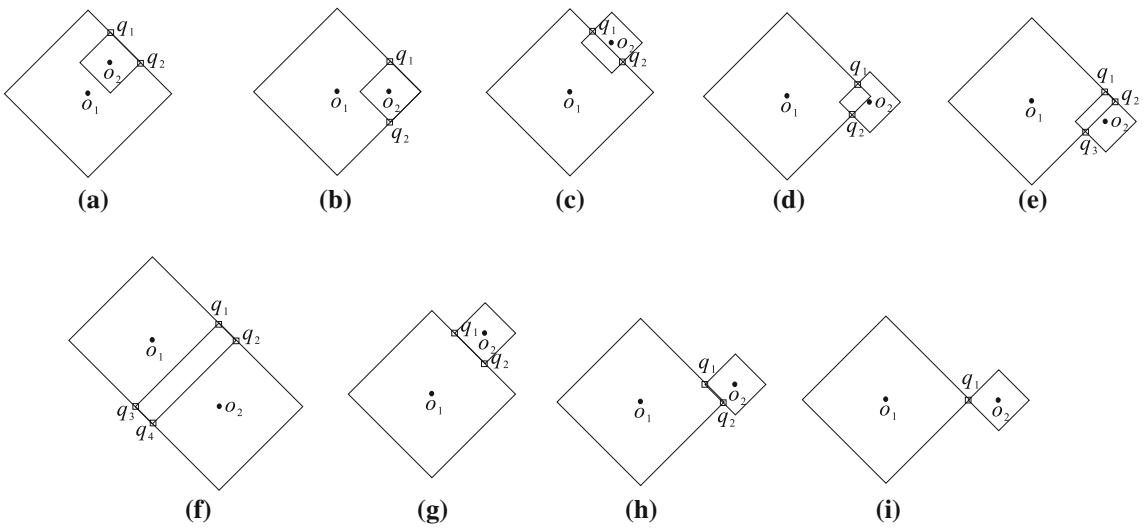


Fig. 12 Different overlapping cases in the L_1-norm space

5 Extension to three-dimensional case

The key idea of the efficiency of for a two-dimensional space is *vantage points*. Since there are a *limited* number of vantage points, the performance of *MaxOverlap* can be guaranteed. In this section, we discuss how we extend the concept of vantage points in a three-dimensional space/case. Unfortunately, the techniques used in the two-dimensional case (which consider *two* nearest location regions) cannot be used *directly* in the three-dimensional case. Some novel techniques (which consider *three* nearest location *spheres*) should be used. We will discuss the novel techniques in detail in this section.

For the sake of illustration, we first discuss how to extend to a three-dimensional case by considering the L_2-norm space in Sect. 5.1. We then briefly discuss adaptations to other metric spaces in Sect. 5.2.

5.1 Three-dimensional case with the L_2-norm space

5.1.1 Nearest location sphere

In a two-dimensional space, each client is associated with a nearest location circle. Similarly, in a three-dimensional space, each client is associated with a *nearest location sphere (NLS)*.

Definition 7 (*Nearest location sphere*) Given a client point o, the *nearest location sphere (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.

Theorem 7 *MaxBRNN problem in the three-dimensional space is 3SUM-hard when the L_2-norm space is used.*

Proof In order to prove this theorem, we consider the following decision problem for MaxBRNN called DMaxBRNN. Given a set $\mathcal{P}$ of server points, a set $\mathcal{O}$ of client points and a positive integer K, does there exist a maximal consistent region R such that, if a new server point p is set up in R, the influence value of R is at least K?

The proof is by transforming a 3SUM hard problem of 3LINES to problem DMaxBRNN. *3LINES:* Given a set L of n lines in a three-dimensional space with integer coefficients, does there exist three lines in L such that these three lines have a point in common.

We set $K = n + 3$. Suppose the three-dimensional space contains three axes, namely x, y, and z. Given an instance of 3LINES, we transform the instance as follows. Firstly, we construct a cuboid Q covering all the vertices of the arrangement of the lines in L such that each face of Q is parallel to the plane containing any two of the three axes. This cuboid can be constructed in the following three steps.

–*Step 1:* Consider two axes, x and y.

- For each line l in L, we do the following sub-steps.
  - We project l on the plane containing x and y axes. Let the projected line be l'.
  - We compute the slope of l' on the plane containing x and y axes.
- We sort the projected lines in the ascending order of their slopes.
- For each pair of adjacent lines in the sorted list, we find the intersection points between these two projected lines.
- For each intersection point, we find the corresponding point in the three-dimensional space by computing the intersection point between two corresponding lines in the three-dimensional space.
- Among all these intersection points, we find the leftmost, rightmost, topmost, and bottommost points. Let the resulting set of these points be I_{xy}.

–*Step 2:* We do similar tasks as Step 1 but we consider the axes y and z. Let the resulting set in this step be I_{yz}.
–*Step 3:* We do similar tasks as Step 2 but we consider the axes z and x. Let the resulting set in this step be I_{zx}.

Let $I = I_{xy} \cup I_{yz} \cup I_{zx}$. According to set I, we construct cuboid Q in $O(n \log n)$ time by finding the minimum value and the maximum value of all points in I for each axis. Let q be the center of Q and d be the diameter of Q. Without loss of generality, we assume that the coordinate of q is $(0, 0, 0)$.

Then, we find the minimum distance $\triangle$ from a vertex of the arrangement of the lines in L to a line in L. This can be done by the following steps. For each point v in I, we compute a variable $\triangle_v$ to be the minimum distance of v to each line in L. Finally, we compute $\triangle$ to be $\min_{v \in I} \triangle_v$.

We then construct two points, b^+ and b^-, such that $b^+ = (0, 0, \beta)$ and $b^- = (0, 0, -\beta)$ where β is a real number to be determined later.

For each line l in L, we find the intersection points between l and Q, namely p_l and $p_l{}'$. We create two spheres. The first sphere (denoted by D_l^+) is a sphere passing through three points, namely p_l, $p_l{}'$, and b^+, while the second sphere (denoted by D_l^-) is a sphere passing through points p_l, $p_l{}'$ and b^-. Let w_l be the width of the intersection between D_l^+ and D_l^- which is illustrated in Fig. 13.

Note that setting different values of β yields different widths w_l. It is easy to verify that β_l is smaller if and only if w_l is larger. Next, we find a large value of β such that w_l is at most $\triangle$ for every line l in L. By elementary trigonometry, we show that β should be set to $\triangle + \triangle^{-1} \cdot d^2$.

Note that every point inside Q is in D_l^+ or D_l^-. Thus, it is in at least n spheres. We conclude that there exists a point in Q in at least $n + 3$ disks if and only if three lines in L intersect in a common point. □

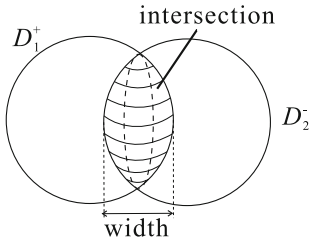


Fig. 13 The width of the intersection between two spheres D_l^+ and D_l^-

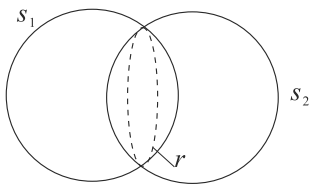


Fig. 14 Illustration for the problem when we define vantage points according to two nearest location spheres only

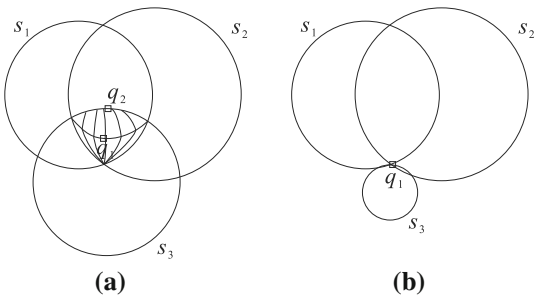


Fig. 15 Illustration for a three-intersecting overlap

5.1.2 Vantage point

In two-dimensional space we define vantage points according to two nearest location circles. It may at first appear that we can define vantage points in three-dimensional space according to two nearest location spheres. However, this is problematic. Suppose that we adopt the definition of vantage points given in Definition 6. That is, given a pair of nearest location spheres s_1 and s_2, the vantage points between s_1 and s_2 are all end points along continuous segment (or curve) such that each point along this segment is an intersection point. Consider Fig. 14 showing that two nearest location spheres, s_1 and s_2, overlap. In this figure, all intersection points between the boundaries of s_1 and s_2 lie on a *ring*, denoted by r. According to Definition 6, since there are no end points for this ring, any point lying on it is a vantage point. Since there are an infinite number of points on this ring, we have an infinite number of vantage points. In other words, MaxOverlap cannot perform efficiently according to this definition.

Consequently, we define vantage points according to *three* nearest location spheres.

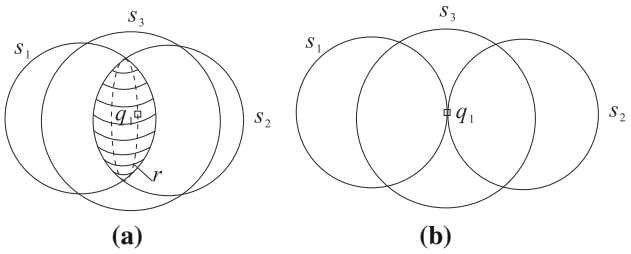


Fig. 16 Illustration for a two-intersecting-and-one-covering overlap

Before we define vantage points, let us consider the overlapping relationships between three nearest location spheres. Given three nearest location spheres, namely s_1, s_2 and s_3, assuming that they are overlapping and have a common intersection, two cases exist:

- *Case 1 (Three-intersecting overlap):* In this case, each of the three nearest location spheres intersects with others as shown in Fig. 15a. In general, the intersection among these three spheres has non-zero volume. A special case is that the intersection is just a single point as shown in Fig. 15b. It is easy to see that there are at least one and at most two intersection points among the *boundaries* of all three spheres. For example, in Fig. 15a, there are two intersection points, namely q_1 and q_2. in Fig. 15b, there is one intersection point, namely q_1.
- *Case 2 (Two-intersecting-and-one-covering overlap):* In this case, there exists two nearest location spheres, say s_1 and s_2, such that one intersects with the other. Besides, the remaining nearest location sphere, say s_3, covers the intersection between s_1 and s_2. This case is illustrated in Fig. 16a. We call s_1 and s_2 the *overlapping spheres*. In general, the intersection among these three spheres has non-zero volume. A special case where the intersection is just a single point is shown in Fig. 16b. It is easy to see that all intersection points between the two overlapping spheres form a *ring*. For example, Fig. 16a shows such a ring r where s_1 and s_2 are overlapping spheres. Fig. 16b shows a special case where the ring is collapsed to a single point q_1.

With Lemma 16, it is easy to conclude the following.

Lemma 14 (Two possible cases) *When three nearest location spheres overlap and they have a common intersection, there are only two possible cases, namely three-intersecting overlap and two-intersecting-and-one-covering overlap.*

All examples shown in Figs. 15 and 16 are possible. But, all examples shown in Fig. 17 are impossible.

We are now ready to define vantage points in a three-dimensional space.

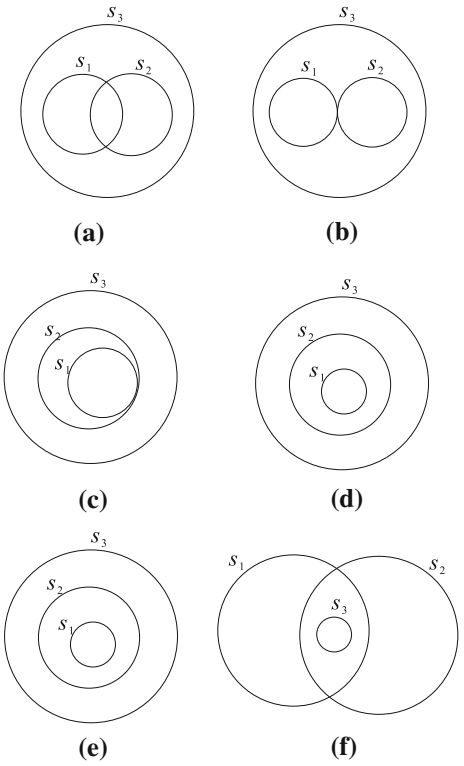


Fig. 17 Impossible overlap relationships

Definition 8 (*Vantage point in three-dimensional space*) Let s_1, s_2 and s_3 be three nearest location spheres. The vantage points among these three spheres are defined as follows.

- If they form a three-intersecting overlap, then the vantage points among these spheres are defined to be all intersection points among these spheres.
- If they form a two-intersecting-and-one-cover overlap, then the vantage point among these spheres is defined to be a randomly selected point along the ring between two-intersecting spheres in this overlap.

Figure 15 shows some examples where three spheres form a three-intersecting overlap. The vantage points are q_1 and q_2 in Fig. 15a while the vantage point is q_1 in Fig. 15b. Figure. 16 shows some examples where three spheres form a two-intersecting-and-one-cover overlap. The vantage point can be any one of the points along the ring r, says q_1, in Fig. 16a while the vantage point is q_1 in Fig. 16b.

It is easy to compute the vantage points as defined above. We next consider whether some previous lemmas can be used. It is easy to see that Lemmas 1, 2, and 5 hold in a three-dimensional space if we consider nearest location spheres (NLSs) instead of nearest location circles (NLCs) with the appropriate restatement of the lemmas as below. Specifically, Lemmas 1, 2, and 5 become Lemmas 15, 16, and 17, respectively. In the following, when we write the *boundary* of a given sphere, for the sake of consistency with our two-dimensional space discussion, we mean the *surface* of the sphere in a three-dimensional space.

Lemma 15 (Intersection representation) *The three-dimensional spatial region R returned by a MaxBRNN query in a three-dimensional space can be represented by an intersection of multiple NLSs.*

Lemma 16 (At least one intersection point) *If an NLS covers another NLS, the boundaries of the two NLSs must share at least one point.*

Lemma 17 (Influence-based pruning) *Let I be a lower bound of the optimal influence value I_o (i.e., $I \leq I_o$). An optimal solution is a three-dimensional spatial region that does not involve any NLS s where (s, L) is an entry for s in the overlap table T and $W(L) < I - w(s)$.*

However, Lemmas 3, 4, and 6 need to be re-stated according to the new definition of vantage points (instead of intersection points) in a three-dimensional space (which we will introduce next).

Lemma 18 (Vantage point computation) *Given three overlapping NLSs s_1, s_2 and s_3, calculating the vantage points among the boundaries of s_1, s_2 and s_3 can be done in $O(1)$ time by elementary mathematical techniques.*

The remaining question is how we re-state Lemmas 3 and 4. One may ask the following questions.

- *What are the minimum number and the maximum number of vantage points when two nearest location spheres overlap?*
- *Is it sufficient to just use vantage points we just defined to find the optimal solution of problem MaxBRNN in the three-dimensional space?*

The first question is related to Lemma 4 and the second question is related to Lemma 3.

We can answer the first question with the following lemma.

Lemma 19 (At least one and at most two intersection points) *There are at least one and at most two vantage points when three nearest location spheres overlap and they have a common intersection.*

Proof Suppose that we are given three nearest location spheres with a common intersection. According to Lemma 14, there are two possible cases, namely three-intersecting overlap and two-intersecting-and-one-covering. If the spheres form a three-intersecting overlap, it is easy to verify that there are at least one vantage point and at most two vantage points in Fig. 15. If they form a two-intersecting-and-one-covering overlap, it is easy to verify that there is exactly one

vantage point in Fig. 16. In conclusion, there is at least one and at most two vantage points when they overlap. □

We can answer the second question with the following lemma.

Lemma 20 (Vantage point identification) *Let S_o be a set of nearest location spheres whose intersection corresponds to a three-dimensional spatial region R returned by a MaxBRNN query. If S_o contains at least three nearest location spheres, then there exist three nearest location spheres, say s_1, s_2 and s_3, such that the three-dimensional spatial region R contains at least one vantage point among the boundaries of three spheres s_1, s_2 and s_3.*

Proof We first make the following claim, which is the same as the current lemma we are proving but focuses on intersection points instead of vantage points.

Claim: "Let S_o be a set of nearest location spheres whose intersection corresponds to a three-dimensional spatial region R returned by a MaxBRNN query. If S_o contains at least three nearest location spheres, then there exist three nearest location spheres, say s_1, s_2 and s_3, such that the three-dimensional spatial region R contains at least one *intersection* point among the boundaries of the three spheres s_1, s_2 and s_3."

This claim can be proven in the same way as Lemma 3.

The next step for the proof of Lemma 20 is to show that the optimal three-dimensional spatial region R contains at least one *vantage* point among the boundaries of some three spheres s_1, s_2 and s_3. Note that every vantage point is an intersection point, but it is possible that an intersection point is not a vantage point.

We prove this by contradiction. Suppose that there do not exist three nearest location spheres such that R contains at least one vantage point among the boundaries of these three spheres. In other words, by the above claim, there exist three nearest location spheres, say s_1, s_2, and s_3, such that R contains at least one *intersection* point q among the boundaries of these three spheres but q is not a vantage point.

According to Lemma 14, we have only two possible cases.

Case 1: Suppose that s_1, s_2, and s_3 form a three-intersecting overlap (See Fig. 15). In this case, it is easy to verify that all intersection points among the boundaries of these three spheres are vantage points. This case is not possible, leading to a contradiction that q_2 is an intersection point but not a vantage point.

Case 2: Suppose that s_1, s_2, and s_3 form a two-intersecting-and-one-covering overlap (See Fig. 16). In this case, it is possible that an intersection point among the boundaries of these three spheres is not a vantage point. We consider two sub-cases.

Case (a): q appears along the boundary of R (e.g., the case shown in Fig. 18a). Note that q is along the boundary of R and q is along the ring for the two overlapping spheres s_1 and s_2. The reason why q occurs along the *boundary* of R is that there exists another sphere s_4 covering q such that the boundary of s_4 intersects with the ring. It is easy to verify that s_1, s_2, and s_4 form a three-intersecting overlap, and thus q is a vantage point among three spheres s_1, s_2, and s_4. This leads to a contradiction that q is not a vantage point.

Case (b): q does not appear along the boundary of R (e.g., the case shown in Fig. 18b). In this case, there exists another point q' that appears along the boundary of R (with the same influence value). This case is similar to Case (a). With the use of q', we also conclude that R covers a vantage point q', which leads to a contradiction. □

Similarly, after knowing the answers of these two questions, it is easy to show that *MaxOverlap* returns a correct solution.

Theorem 8 (Correctness) *MaxOverlap returns a three-dimensional spatial region R with the largest influence value (i.e., the optimal solution returned by the MaxBRNN query) in a three-dimensional space.*

MaxOverlap algorithm for the three-dimensional case is shown in Algorithm 2.

Algorithm 2 Algorithm **MaxOverlap** in a three-dimensional case

```
1: for each o ∈ O do
2:    construct an NLS for o
3: end for
4: build the overlap table T
5: choose the pair of overlapping NLSs, say s1 and s2, with the largest
   W({s1, s2})
6: let p be {s1, s2}
7: So ← {p}
8: Io ← W(p)
9: A ← ∅
10: while there exists an entry in T do
11:    remove an entry (s, L) with the largest value of W(L) from T
12:    for any three NLSs s, s′, s″ ∈ L do
13:       compute the vantage points among the boundaries of s, s′ and s″
14:       for each vantage point q found do
15:          perform a point query from q to find all NLSs covering q
16:          let S be the result of the above point query
17:          I ← W(S)
18:          if I > Io then
19:             So ← S
20:             Io ← I
21:          end if
22:       end for
23:    end for
24:    A ← A ∪ {s}
25:    remove all entries (s′, L′) from T where W(L′) < Io − w(s′)
26: end while
```

Next, we analyze the complexity of *MaxOverlap* in a three-dimensional space. Similarly, we have the following five detailed steps.

- Step 1 (NLR (or NLC) Construction),
- Step 2 (Overlap Table Construction),
- Step 3 (Initialization),
- Step 4 (Entry Sorting) and
- Step 5 (Iterative Step).

Steps 1–4 are the same as that for two-dimensional case, but we consider $\mathbb{D}$ instead of the L_2-norm space. Their complexities are $O(|\mathcal{O}|\alpha(|\mathcal{P}|))$, $O(|\mathcal{O}|\beta(|\mathcal{O}|))$, $O(k|\mathcal{O}|)$, and $O(|\mathcal{O}|\log|\mathcal{O}|)$, respectively, where we overload notations α, β, and γ to represent the same concepts in the three-dimensional case.

Step 5 is similar to the two-dimensional case. That is,

–Step (i): For any three NLSs with a common intersection, namely s_1, s_2, and s_3, we compute the vantage points among s_1, s_2 and s_3.

–Step (ii): For each vantage point q found, we perform a point query for q.

Since the total number of vantage points is $O(k^2|O|)$ and performing a point query for each q is $O(\theta(|\mathcal{O}|))$, Step 5 takes $O(k^2|O|\theta(|\mathcal{O}|))$ time.

Theorem 9 (Running time in a three-dimensional case) *The running time of Algorithm 2 in a three-dimensional case is* $O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + |\mathcal{O}|\log|\mathcal{O}| + k^2|O|\theta(|\mathcal{O}|))$ *where* k *is the greatest size of* L *of an entry* (s, L) *in the overlapping table* T.

5.2 Three-dimensional case with other Minkowski metrics

In Sect. 5.1, we solved the MaxBRNN problem in the three-dimensional case with the L_2-norm space. In this section, we discuss how to apply the techniques discussed in Sect. 5.1 to the three-dimensional case with any arbitrary Minkowski distance metric of order 1 or above.

We first illustrate how the techniques can be extended to two common metric spaces, namely the L_1-norm space and the L_∞-norm space. Under the L_2-norm space, given a client point o, we have the nearest location sphere centered at o. However, under the L_1-norm space, we have an *octahedron* centered at o as shown in Fig. 19a while under the L_∞-norm space, we have a *cube* centered at o as shown in Fig. 19b. The same principle discussed in Sect. 5.1 still applies under the L_1-norm space and the L_∞-norm space. We just replace the sphere used under the L_2-norm space by the octahedron under the L_1-norm space or the cube under the L_∞-norm space.

The techniques can be generalized to any Minkowski distance metric of order 1 or above since we can always construct a convex NLR centered at a client point.

5.3 Other metric spaces

Other metric spaces can be considered under some conditions. In an arbitrary metric space, we can also construct nearest location regions. The set $\{p : d(p, o) = r\}$ is called the sphere of radius r with centre o. This generalizes the notion of spheres in a Euclidean space to arbitrary metric spaces. Such a set forms the nearest location region for o if r is the distance to $o's$ nearest neighbor in terms of the metric. Note that the *sphere* in a metric space need not look like a sphere in Euclidean space. We have seen *spheres* of different shapes for L_s-norms. In general, the nearest location region can be disconnected, or it may be discrete.

In the discussion of our method for the Minkowski metrics, the only properties for nearest location regions that we use are that they should be contiguous and convex. Therefore, if a metric generates convex contiguous nearest location regions, then our method can also be used. One may also consider the possibility of transforming the given metric to one with the above properties while preserving the nearest neighbor characteristics. One such related work is [14].

6 Empirical study

We have conducted extensive experiments to test our proposal. The experiments are run on a Pentium IV 2.2 GHz PC with 1GB memory, on a Linux platform. The algorithms were implemented in C/C++. We deployed four real datasets

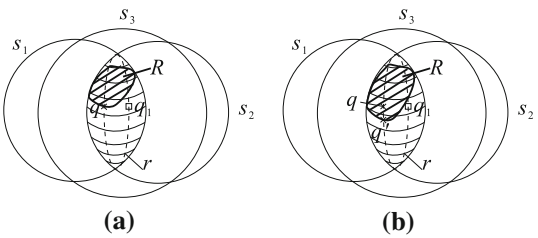


Fig. 18 Illustration for the proof of Lemma 20

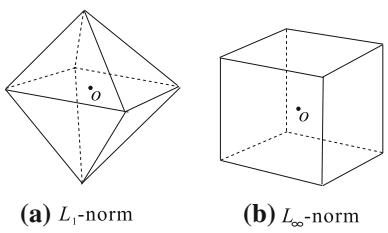


Fig. 19 Nearest location object in different Minkowski metrics for the three-dimensional case

Table 2 Summary of the real datasets

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

that are available at http://www.rtreeportal.org/spatial.html. The summary of the real datasets is shown in Table 2 where CA, LB, GR, and GM contain two-dimensional points representing geometric locations in California, Long Beach Country, Greece, and Germany, respectively. For datasets containing rectangles, we transformed them into points by taking the centroid of each rectangle. For all datasets, 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 of experiments 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.

Similar to [21], we also created synthetic datasets each of which contains $\mathcal{P}$ following Gaussian distribution and $\mathcal{O}$ following Zipfian distribution. The coordinates of each point were generated in the range [0, 10,000]. In $\mathcal{P}$, each coordinate follows Gaussian distribution where the mean and the standard derivation are set to 5,000 and 2,500. In $\mathcal{O}$, each coordinate follows Zipfian distribution skewed toward 0 where the skew coefficient is set to 0.8. In both cases, all coordinates of each point were generated independently. Since we are studying the problem with two-dimensional points, the dimensionality is set to 2 in the dataset generator.

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 report the results when the weight is equal to 1. In the experiments, we focus on the study of lMax$k$BRNN since it is more general than MaxBRNN, Max$k$BRNN, and lMaxBRNN.

We further divide *MaxOverlap* into two algorithms called *MaxOverlap-P* and *MaxOverlap-NP*. *MaxOverlap-P* is our proposed algorithm that considers pruning described in Lemma 5. *MaxOverlap-NP* is similar, but it does not use any pruning according to Lemma 5. In the following, when we describe *MaxOverlap*, we mean both algorithms.

We compare our proposed algorithm with two algorithms. The first one is the best-known algorithm for MaxBRNN problem called *Arrangement* [3,4]. The second one is an algorithm called *Buffer-Adapt* [10] which is originally designed to solve problem MaxBRNN in the L_1-norm (instead of the L_2-norm) and is now adapted for problem MaxBRNN in the L_2-norm. Specifically, in *Buffer-Adapt*, for each client point o, its nearest location box is constructed (together with its nearest location circle) similar to [10]. Then, the box is scaled by $\sqrt{2}$. It is easy to see that the scaled nearest location box of each client point covers its nearest location circle. A variable called *sol* is used to store the optimal solution found so far. Initially, *sol* is set to $\emptyset$. The adapted algorithm starts the ordinary plane-sweep algorithm [10]. Whenever the algorithm finds a set A of intersecting boxes, if the weighted size of A is greater than the weighted size of *sol*, it performs a refinement step that computes a set B of intersecting circles (that are covered by the boxes of A) such that the weighted size of B is maximized. If the weighted size of B is greater than that of *sol*, we update *sol* by B. We iteratively perform the above steps until we process all boxes. The final solution can be found in *sol*.

As indicated earlier, we adopted an R^*-tree [2] as an indexing structure for the nearest neighbor search and the k-th nearest neighbor search where the node size is fixed to 1k bytes.[1] The maximum number of entries in a node is equal to 50, 36, 28, and 23 for dimensionality equal to 2, 3, 4, and 5, respectively. We set the minimum number of entries in a node to be equal to half of the maximum number of entries.

We evaluated the algorithms in terms of two measurements: (1) *execution time* and (2) *storage*.

The execution time corresponds to the time of executing the algorithms. The memory usage of *MaxOverlap* is equal to the memory used by the indexes and the overlap table (i.e., the R^*-tree R_P built over all data points in P, the R^*-tree R_C built over all NLCs and the overlap table). The memory used by *Arrangement* is equal to the memory used by the total number of faces between adjacent Voronoi cells used in the algorithm. The memory used by *Buffer-Adapt* is equal to the memory used by the indexes and the data structures used in the plane-sweep algorithm (i.e., the R^*-tree R_P built over all data points in P, the lookup table built over all nearest location boxes where each entry of the table stores a NLC in addition to a nearest location box and the data structures used in the plane-sweep algorithm).

We have also evaluated *MaxOverlap* in terms of other measurements, namely (3) *pruning power*, (4) *overlap table storage*, (5) *R-tree storage*, (6) *average no. of overlaps*, and (7) *average influence value*.

Pruning power is equal to the number of client points that are pruned without any consideration given to the influence-based pruning. *R-tree storage* corresponds to the memory consumption due to indexing, namely $R_{\mathcal{P}}$ and $R_{\mathcal{C}}$. *Average no. of overlaps* is the average number of NLCs that overlap with an NLC in the data set. The *average influence value* is the average influence value of regions in the output of the

[1] We choose a smaller page size to simulate practical scenarios where the dataset cardinality is much larger.

lMax$k$BRNN query. If l is equal to 1, the average influence value is equal to the highest influence value.

In the experiments, we study the effect of cardinality, k and l. All experiments were conducted 100 times and we report the average for the results.

We present our results in four parts. The first three parts are based on datasets of dimensionality equal to 2 while the last one is based on datasets of dimensionality equal to 3.

The first part (Sect. 6.1) focuses on the performance comparison between all algorithms in the L_2-norm space. The second part (Sect. 6.2) focuses on the scalability of our proposed algorithm on datasets with larger cardinality in the L_2-norm space. The third part (Sect. 6.3) gives the experimental studies in the L_1-norm space. The fourth part (Sect. 6.4) studies the performance of algorithm *MaxOverlap* in the dataset of dimensionality equal to 3.

6.1 Performance in the L_2-norm metric

Since *Arrangement* algorithm is not scalable with respect to large datasets, we conduct the comparison experiments with smaller $\mathcal{O}$, namely 50, 100, 150, 200, and 250, and $\mathcal{P}$ of size equal to $2|\mathcal{O}|$. We adopt the *CA-GR* dataset where $(\mathcal{P}, \mathcal{O}) = (CA, GR)$. Since these values are smaller than the cardinality of the real datasets, we sample the data points accordingly. We set $l = 1$ and $k = 1$. Figure 20a shows that the execution times of all algorithms increase with the cardinality of datasets. The execution time of *Arrangement* is much greater than that of *MaxOverlap*. *MaxOverlap* performs 1,000,000 times faster than *Arrangement* when $|\mathcal{O}| = 250$. In particular, *MaxOverlap* only runs within 0.1s but *Arrangement* runs more than 1 day on this dataset. This is because *MaxOverlap*, which makes use of the overlap relationships between NLCs, runs in polynomial-time while *Arrangement*, which considers Voronoi cells (which is exponential in terms of the total number of client points), runs in exponential time. In addition, *Buffer-Adapt* performs slower than *MaxOverlap* by orders of magnitude. This is because *Buffer-Adapt* has some drawbacks related to the L_1-norm. Firstly, it introduces an additional overhead related to the boxes for the L_1-norm (which does not exist in our original problem that depends on the circles for the L_2-norm). Secondly, an intersection among a certain set of boxes used in *Buffer-Adapt* does not imply an existence of an intersection among the corresponding circles (which can represent the optimal solution). Since *Buffer-Adapt* has "larger" boxes (compared with circles), it is more likely that the boxes are intersecting. Thus, more refinement steps need to be done redundantly. Furthermore, in these small data sets, the execution times of *MaxOverlap-P* and *MaxOverlap-NP* are similar.

For the same setting, in Fig. 20b, the storage usage of all algorithms increase with the cardinality of the dataset. The storage requirement of *Arrangement* is much larger than that of *MaxOverlap*. For example, when $|\mathcal{O}| = 250$, *Arrangement* consumes about 28 times more memory than *MaxOverlap*. This is because *Arrangement* needs to keep track of Voronoi cells, which consumes considerably more memory compared with the indexing structures used in *MaxOverlap*. In addition, the storage of *Buffer-Adapt* is slightly greater than that of *MaxOverlap* due to the additional storage overhead of boxes.

As mentioned above, there exists work [10] that addresses problem MaxBRNN in the L_1-norm and this can be adapted to the problem in the L_2-norm. In the following, we apply the original algorithm (for the L_1-norm) to the problem in the L_2-norm directly without any adaption. We call this algorithm *Buffer*. Specifically, algorithm *Buffer* returns a *single point* that maximizes its influence value in the L_1-norm only. We took this point for the MaxBRNN query in the L_2-norm and obtained the (weighted) size of BRNN set. The (weighted) size corresponds to the influence value returned by *Buffer*. Since *Buffer* is not designed for the L_2-norm, the influence value with respect to the L_2-norm is smaller than the influence value returned by algorithm *MaxOverlap* (or *Buffer-Adapt*). In all experiments with the same setup as above, we found that, on average, the influence value of the point returned by *Buffer* in the L_2-norm is about 26.9% of the optimal influence value returned by *MaxOverlap* (or *Buffer-Adapt*), which suggests that *Buffer* cannot be used as an approximate algorithm for problem MaxBRNN in the L_2-norm.

We also tried to adapt *Buffer* so that the adapted approach returns the optimal region (in form of box) in L_1-norm. Then, we pick a sample set of points in the optimal region to perform the MaxBRNN queries in the L_2-norm and obtain the average (weighted) size of BRNN sets, which corresponds to the influence value returned by *Buffer*. Let N be the sample size. We tested this adapted approach with $N = 250, 500, 750, 1{,}000$. On average, the influence value returned by *Buffer* is at most 34.69% of the optimal influence value returned by algorithm *MaxOverlap* (or algorithm *Buffer-Adapt*).

6.2 Scalability

The experiments reported in the previous section demonstrate that *MaxOverlap* is considerably better than *Arrangement* and *Buffer-Adapt* in terms of execution time and memory consumption. In this section, we study the scalability of *MaxOverlap* using both synthetic datasets and real datasets (recall that *Arrangement* is not scalable to handle large datasets). The default values of the synthetic datasets are shown in Table 3. In these experiments, we do not sample the real datasets; instead, the cardinality of the real datasets used in the experiments are shown in Table 2. The default values

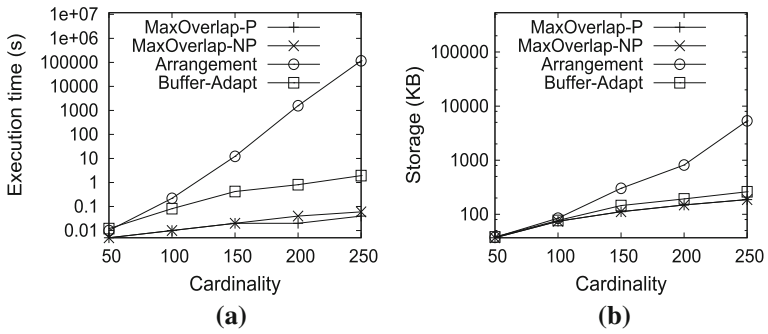


Fig. 20 Effect of cardinality (small real data set where $\mathcal{O}$=GR and $\mathcal{P}$=CA)

Table 3 Default synthetic dataset cardinalities

	Default value
($\|\mathcal{O}\|$)	50k
($\|\mathcal{P}\|$)	$2\|\mathcal{O}\|$

of l and k are both 1. Figures 21 and 22 show the results when we vary the cardinality of the dataset (i.e., $|\mathcal{P}|$) and k, respectively.

Effect of cardinality: Figure 21a shows that the execution times of *MaxOverlap-P* and *MaxOverlap-NP* algorithms increase with $|\mathcal{P}|$ where $|\mathcal{P}| = 2|\mathcal{O}|$. Since *MaxOverlap-P* prunes some client points while *MaxOverlap-NP* does not, *MaxOverlap-P* runs faster than *MaxOverlap-NP*. Figure 21b shows that nearly 99% of client points are pruned by Lemma 5 in *MaxOverlap-P*. We observe that although nearly 99% of client points are pruned, the performance gain for *MaxOverlap-P* is not nearly 99%. This is because the client points c processed in *MaxOverlap-P* has large sizes of $L(c)$, which takes a lot of execution time. However, the client points c' pruned in *MaxOverlap-P* but processed in *MaxOverlap-NP* has small sizes of $L(c)$. Thus, the performance gain is not nearly 99%. Figure 21c depicts the expected increase in the storage requirement for the overlap table and the R-tree storage with the cardinality of dataset, because both $\mathcal{O}$ and $\mathcal{P}$ increase in size. In Fig. 21d, the average no. of overlappings increases slightly with the cardinality of dataset, $|\mathcal{P}|$. With more data points, it is more likely that an NLC overlaps with another NLC. Thus, the average number of overlaps increases. In addition, from Fig. 21d, the average influence value increases with the cardinality of dataset because the average number of overlaps is increased.

Effect of k: As shown in Fig. 22a, the execution times of *MaxOverlap* increase with k. This is because as k increases, we need to find more nearest neighbors. Figure 22b shows that the pruning power of *MaxOverlap-P* is robust to changes in k. From Fig. 22c, the overlap table storage increases with k but the R-tree storage remains unchanged. With a larger k value, it is more likely that an NLC for a client point overlaps with another NLC. Then, the number of NLCs that overlap with an NLC is larger. Thus, the overlap table size is larger. Since k is independent of the R-tree storage, the R-tree storage remains unchanged. Figure 22d shows that the average number of overlaps and the average influence value increase with k. As we described, a larger value of k increases the chance of NLC overlaps, and thus the average number of overlappings. With a larger average number of overlaps, it is more likely that the average influence value in the l regions returned by the query is larger.

Effect of l: We have conducted experiments with l values of 1, 5, 10, and 15. The execution time, the pruning power, the overlap table storage, the R-tree storage, the average number of overlaps, and the average influence value all remain nearly unchanged when l is increased. In the interest of space, we omit the figures.

Effect of real datasets: We have conducted experiments on the four sets of real datasets, namely CA-GR, LB-GR, CA-GM, and LB-GM. The results are also similar to synthetic datasets. For space reasons, we only show the figures for CA-GR and CA-GM. Figures 23 and 24 show the results for CA-GR while Figs. 25 and 26 show the results for CA-GM.

Conclusion: We find that *MaxOverlap* is more efficient than the best-known algorithm, *Arrangement*. It utilizes less memory as well. Our proposed algorithm is scalable to large datasets, while *Arrangement* is not.

6.3 Extension to the L_1-norm space

In the previous sections, we reported experiments in the L_2-norm space. In this section, we choose the L_1-norm metric and study the performance of our extension into L_1. The reason why we chose the L_1-norm metric is that it is well-known and commonly used. We compared our proposed algorithm *MaxOverlap* (which can be adapted to solve problem MaxBRNN in the L_1-norm space) with algorithm *Buffer* (which is an algorithm for solving the same problem in the L_1-norm space).

Figure 27 shows the execution times of *MaxOverlap* and *Buffer* for synthetic datasets. We vary the cardinality of the

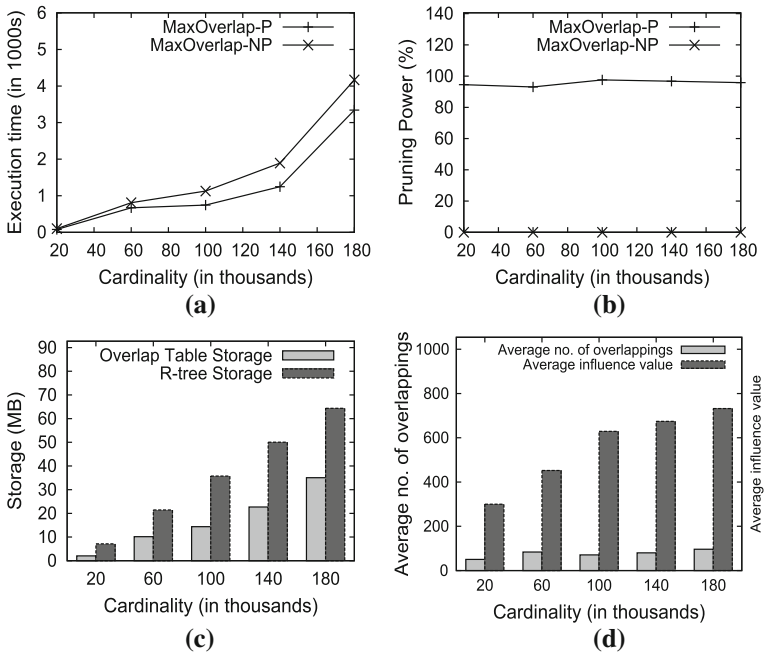


Fig. 21 Effect of cardinality (synthetic data set)

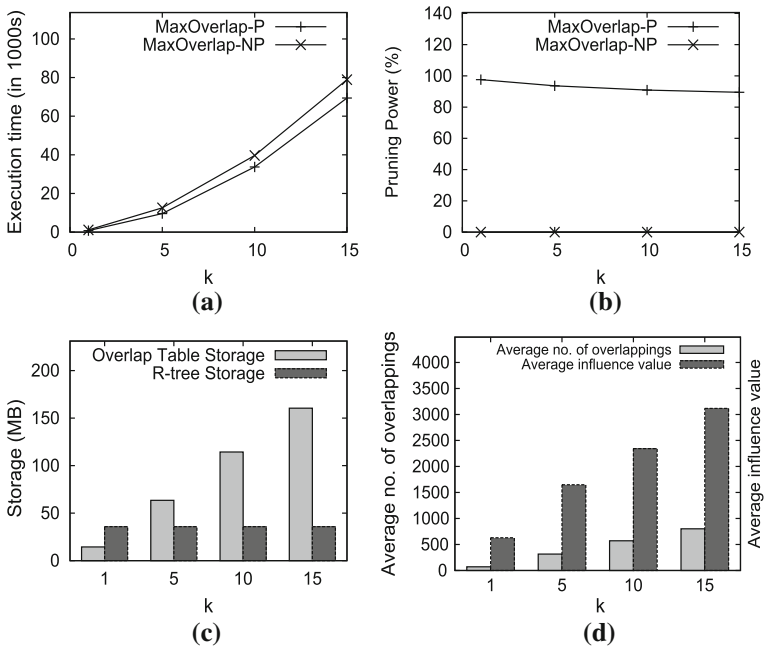


Fig. 22 Effect of k (synthetic data set)

dataset, parameter k and parameter l where the results can be found in Fig. 27a, b ,c, respectively. Figure 28 shows the execution times for real dataset CA-GR.

In these figures, the execution time of *MaxOverlap* is higher than that of *Buffer*. In the previous sections, we showed that *MaxOverlap* has good performance in the L_2-norm space. Since *MaxOverlap* can be adapted to solve problem MaxBRNN in *any* metric space, and thus it is very *general*, it is not tailor-made to solve MaxBRNN problem in any particular metric space. Thus, it cannot make use of some properties in this particular metric space to speed up the computation. Thus, in the L_1-norm space, the execution time of *MaxOverlap* is higher than that of *Buffer* as shown in the figures. We note, however, that the difference in real dataset is slight. Coupled with the fact that *MaxOverlap* can work in any metric space while *Buffer* cannot, it is obvious that *MaxOverlap* is more robust across a wider range of applications.

As before, the execution times of all algorithms increase with the cardinality of the dataset and k, and they remains nearly unchanged when l increases.

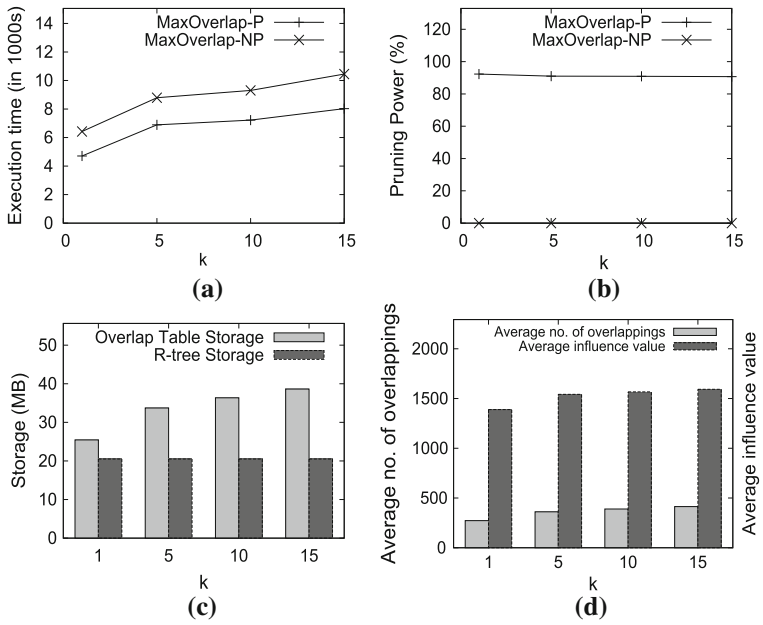


Fig. 23 Effect of k (real data set CA-GR)

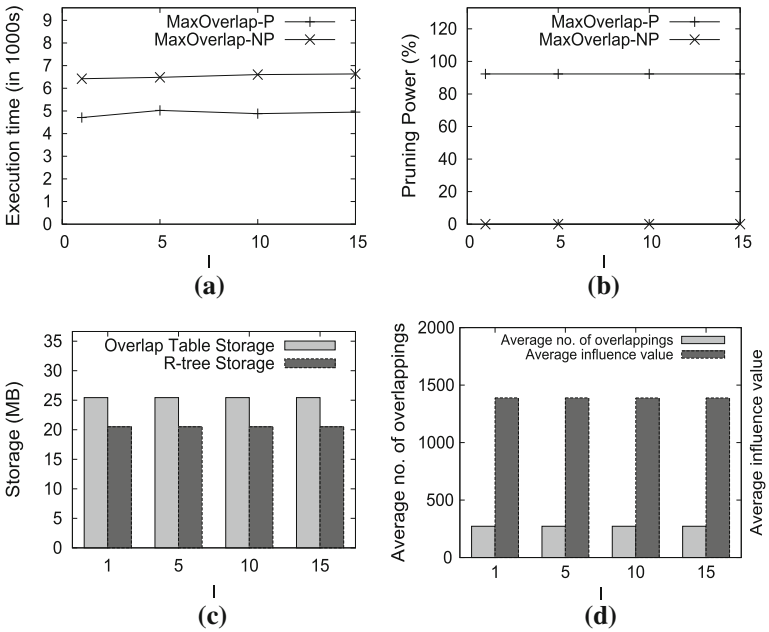


Fig. 24 Effect of l (real data set CA-GR)

6.4 Extension to three-dimensional case

In this section, we report the results of experiments in the three-dimensional space by using the L_2-norm metric. Similarly, we also implemented algorithm *Arrangement* to work in the three-dimensional case. Since it is not scalable, we conducted experiments in a small dataset where the cardinality of the synthetic dataset is 250. The execution time of *Arrangement* is 212,566 s while the execution time of *MaxOverlap* is 0.07 seconds. Thus, *MaxOverlap* runs faster than *Arrangement* by several orders of magnitude.

In the following, we show experimental results with larger datasets. Since *Arrangement* is not scalable, in the following, we only show the results of *MaxOverlap*. Figure 29a, b show that, as expected, the execution time of *MaxOverlap* increases with the cardinality of the dataset and parameter k. Figure 29c shows that the execution time of *MaxOverlap* remains nearly unchanged when l changes. The explanations are similar to what we reported before. We note that the execution time of *MaxOverlap* in the three-dimensional space is much larger than the execution time of *MaxOverlap* in the two-dimensional space (shown in the previous experimental

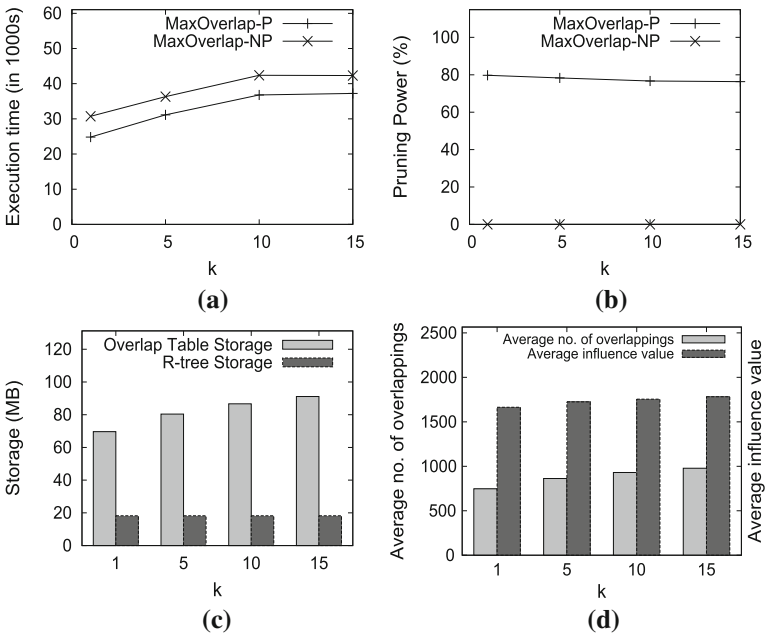


Fig. 25 Effect of k (real data set CA-GM)

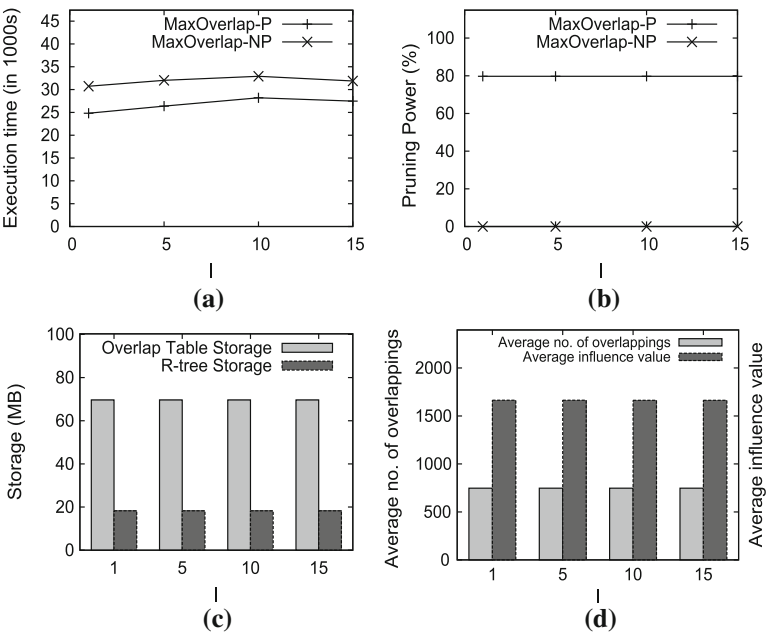


Fig. 26 Effect of l (real data set CA-GM)

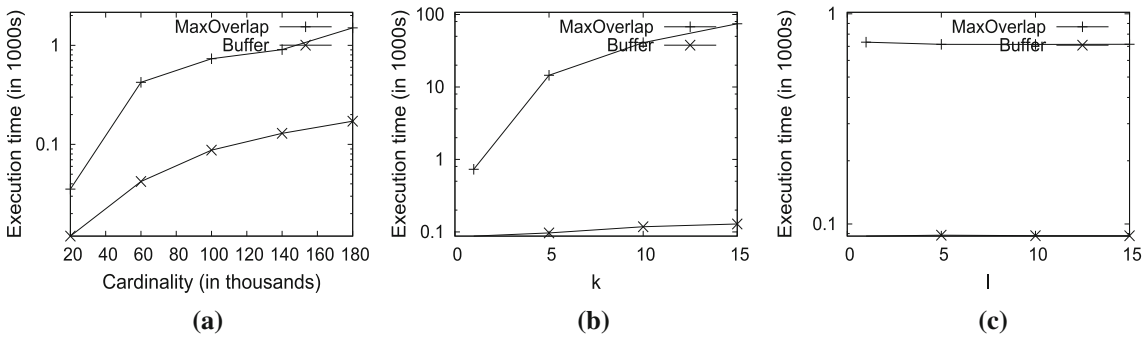


Fig. 27 Comparison between MaxOverlap and buffer in the L_1-norm space (synthetic data set)

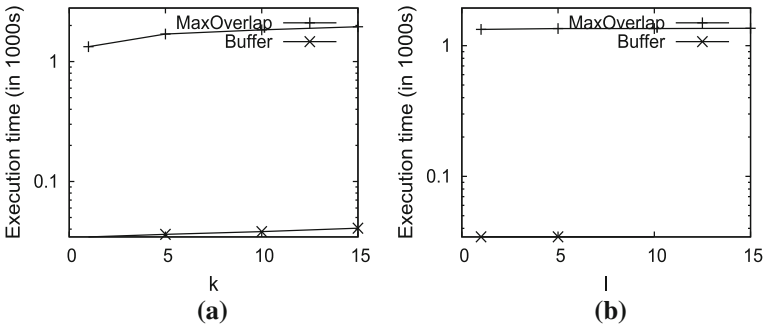


Fig. 28 Comparison between MaxOverlap and buffer in the L_1-norm space (real data set CA-GR)

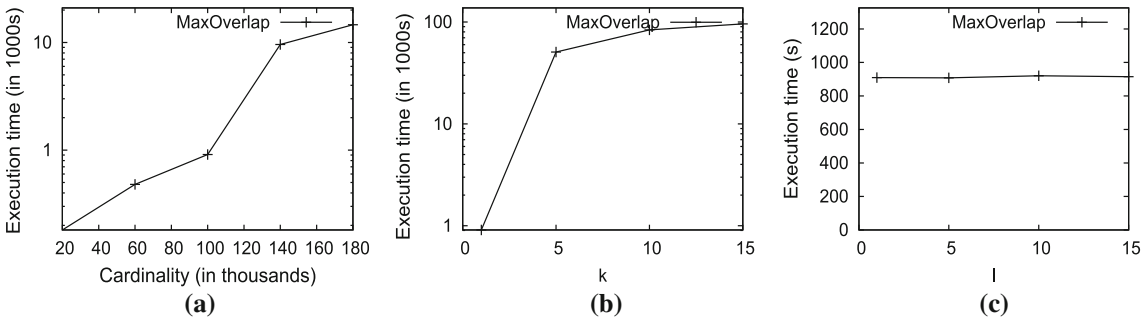


Fig. 29 Performance of MaxOverlap in the three-dimensional space (synthetic data set)

results). This is because it has to process more dimensions in the three-dimensional space.

7 Related work

Bichromatic reverse nearest neighbor (BRNN) search was first proposed by Korn and Muthukrishnan [12]. The fastest BRNN algorithm is due to Stanoi et al. [15]. BRNN search has been used to to discover the most "influential" server among a number of servers, which has the largest BRNN set [22], and to address other problems [13,16]. None of the above works, however, can be applied to solve MaxBRNN, because, as explained in Sect. 1, there are a large (or infinite) number of points in the data space, and it is infeasible to perform a large (or infinite) number of MaxBRNN queries.

The existing work which are closely related to ours is that of Cabello et al. [3,4], which considers MaxBRNN for L_2-norm in the two-dimensional space. In that work, the problem is shown to be 3SUM-hard. If it takes $O(N)$ time to solve 3SUM (as conjectured) where N is the size of the dataset, then it must take at least quadratic time to solve MaxBRNN. Cabello et al. proposes a method based on the *arrangement* of NLCs of the client points. This method involves three major steps. The first step is to construct a set of NLCs for client points. Similar to our method, this step can be done in $O(|\mathcal{O}| \log |\mathcal{P}|)$ time. The second step is to find an arrangement according to a set of NLCs. The best-known efficient method to find an arrangement [1] has the running time of $O(N^2)$ time where N is the number of points in the dataset. In our case, since each point corresponds to an NLC, N is equal to $|\mathcal{O}|$. The third step is to find the best region by *traversing* from a Voronoi cell to another cell by the *face* between these two cells iteratively. Since the algorithm relies heavily on the total number of possible faces between adjacent Voronoi cells used in the arrangement and the total number of possible faces is $O(2^{\gamma(|\mathcal{O}|)})$ where $\gamma(|\mathcal{O}|)$ is a function on $|\mathcal{O}|$ and is $\Omega(|\mathcal{O}|)$, the method is exponential in terms of $|\mathcal{O}|$. Specifically, the complexity is $O(|\mathcal{O}| \log |\mathcal{P}| + |\mathcal{O}|^2 + 2^{\gamma(|\mathcal{O}|)})$. In other words, this method is not scalable with respect to dataset size. Cabello's proposal is the only known method for L_2-norm but it does not include any empirical studies.

MaxBRNN problem is also studied in the L_1-norm metric [10]. This algorithm finds an optimal *location* l instead of a region maximizing the influence value of l, but the work is limited to the L_1-norm.

Some other related problems have been studied [5,23]. The first one [5] proposes to find a location p for a new server in order to minimize the maximum distance between p and any client point $o \in \mathcal{O}$. The second one [23] proposes the min-dist optimal location query. Given a set $\mathcal{P}$ of servers, a set $\mathcal{O}$ of client points and a spatial region Q, the min-dist optimal location query returns a location in Q that minimizes the average distance from each client point to its closest server if a new site is built at this location.

In the preliminary version of this paper [20], we introduce the MaxOverlap problem and proposed the *MaxOverlap* algorithm in the two-dimensional case when the metric space

is the L_2-norm space. In his paper, we extend the problem definition to any Minkowski metric of order 1 or above for two- and three-dimensional spaces and propose appropriate algorithmic solutions. These problem had not been studied previously despite their importance and practical relevance.

8 Conclusion

In this paper, we propose an efficient algorithm called *MaxOverlap* to address MaxBRNN problem. We experimentally show the efficiency of our proposed algorithm. We also extend our approach to any metric space and the three-dimensional spaces.

There are a lot of promising research directions. Firstly, it would be interesting to consider higher dimensions. Secondly, some recent works [18,21] consider the capacity of each server, while we have so far assumed that each server can serve as many clients as possible, consistent with the assumptions in the literature. This was done to be able to have baseline comparisons with existing work. With the consideration of server capacity, each client point may be associated with a farther server (instead of the nearest server) for forming an NLC in order to satisfy the capacity requirement.

Acknowledgments We are grateful to the anonymous reviewers for their constructive comments on this paper. The research of Raymond Chi-Wing Wong and Lian Liu is supported by HKRGC GRF 621309 and NSFC project no. 61070005. The research of M. Tamer Özsu is supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada. The research of Philip S. Yu is supported by US NSF through grants IIS 0905215, DBI-0960443, OISE-0968341, and OIA-0963278. The research of Yubao Liu is supported by NSFC project no. 61070005. All opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

1. Amato, N.M., Goodrich, M., Ramos, E.A.: Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. In: Proceedings of the 11th ACM-SIAM Symposium on Discreate Algorithms, pp. 705–706 (2000)
2. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The *R**-tree: an efficient and robust access method for points and rectangles. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 322–331 (1990)
3. Cabello, S., Diaz-Banex, J.M., Langerman, S., Seara, C.: Facility location problems in the plane based on reverse nearest neighbor queries. Eur. J. Operat. Res. **202**(1), 99–106 (2010)
4. Cabello, S., Diaz-Banez, J.M., Langerman, S., Seara, C., Ventura, I.: Reverse facility location problems. In: Canadian Conference on Computational Geometry, pp. 68–71 (2005)
5. Cardinal, J., Langerman, S.: Min-max- min geometric facility location problems. In: 22nd European Workshop on Computational Geometry (2006)
6. Chazelle, B.: Filtering search: a new approach to query-answering. SIAM J. Comput. **15**, 703–724 (1986)
7. Chazelle, B.: New upper bounds for neighbor searching. Inf. Control **68**(1–3), 105–124 (1986)
8. Chiueh, T.: Content-based image indexing. In: VLDB (1994)
9. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer, Berlin (2000)
10. Du, Y., Zhang, D., Xia, T.: The optimal-location query. In: SSTD, pp. 163–180 (2005)
11. Kang, J.M., Mokbel, M.F., Shekhar, S., Xia, T., Zhang, D.: Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In: Proceedings of the International Conference on Data Engineering, pp. 806–815 (2007)
12. Korn, F., Muthukrishnan, S.: Influence sets based on reverse nearest neighbor queries. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 201–212 (2000)
13. Krarup, J., Pruzan, P.M.: The simple plant location problem: survey and synthesis. Eur. J. Operat. Res. **12**(1), 36–57 (1983)
14. Shahabi, C., Kolahdouzan, M.R., Sharifzadeh, M.: A road network embedding technique for *k*-nearest neighbor search in moving object databases. GeoInformatica **7**(3), 255–273 (2003)
15. Stanoi, I., Riedewald, M., Agrawal, D., Abbadi, A.E.: Discovery of influence sets in frequently updated databases. In: Proceedings of the International Conference on Very Data Bases (VLDB). (2001)
16. Tansel, B.C., Francis, R.L., Lowe, T.: Location on networks: a survey. Manage. Sci. **29**(4), 482–497 (1983)
17. Tao, Y., Faloutsos, C., Papadias, D.: Spatial query estimation without the local uniformity assumption. GeoInformatica **10**(3), 261–293 (2006)
18. Yiu, L.H.U.M.L., Mouratidis, K., Mamoulis, N.: Capacity constrained assignment in spatial databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 15–28 (2008)
19. Wong, R.C.-W., Ozsu, T., Fu, A.W.-C., Yu, P.S., Liu, L., Liu, Y.: Maximizing bichromatic reverse nearest neighbor for lp-norm in two- and three-dimensional spaces. In: http://www.cse.ust.hk/~raywong/paper/maxRNNtechnicalReport.pdf (2011)
20. Wong, R.C.-W., Ozsu, T., Yu, P.S., Fu, A.W.-C.: Efficient method for maximizing bichromatic reverse nearest neighbor. In: Proceedings of the International Conference on Very Data Bases (VLDB). (2009)
21. Wong, R.C.-W., Tao, Y., Fu, A.W.-C., Xiao, X.: On efficient spatial matching. In: Proceedings of the International Conference on Very Data Bases (VLDB), pp. 579–590 (2007)
22. Xia, T., Zhang, D., Kanoulas, E., Du, Y.: On computing top-t most influential spatial sites. In: Proceedings of the International Conference on Very Data Bases (VLDB), pp. 946–957 (2005)
23. Zhang, D., Du, Y., Xia, T., Tao, Y.: Progressive computation of the min-dist optimal-location query. In: Proceedings of the International Conference on Very Data Bases (VLDB), pp. 643–654 (2006)