

Efficient Method for Maximizing Bichromatic Reverse Nearest Neighbor*

Raymond Chi-Wing Wong¹, M. Tamer Özsu², Philip S. Yu³, Ada Wai-Chee Fu⁴, Lian Liu¹

¹Hong Kong University of Science and Technology

raywong,lyanliu@cse.ust.hk

³University of Illinois at Chicago

psyu@cs.uic.edu

²University of Waterloo

tozsu@uwaterloo.ca

⁴Chinese University of Hong Kong

adafu@cse.cuhk.edu.hk

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. 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, the fastest known method has exponential time complexity on the data size. Based on some interesting properties of the problem, we come up with an efficient algorithm called MaxOverlap. Extensive experiments are conducted to show that our algorithm is many times faster than the best-known technique.

1. INTRODUCTION

Bichromatic reverse nearest neighbor (BRNN) search has been extensively studied as an important operator in spatial databases [10, 11, 18]. 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 $|o, p'| < |o, p|$. Those 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, consider that we

*This research was supported by an RGC Earmarked Research Grant of HKSAR HKUST and by the Natural Sciences and Engineering Research Council (NSERC) of Canada. 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. Work was performed partly during Raymond’s stay at the University of Waterloo.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '09, August 24-28, 2009, Lyon, France

Copyright 2009 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

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 a set \mathcal{P} of two convenience stores, namely p_1 and p_2 , and a set \mathcal{O} of five customers, 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 $BRNN(p_2, \mathcal{P}) = \{o_3, o_4, o_5\}$.

Consider that a new convenience store p_3 is being set up and the company tries to find a location to maximize the number of customers who will go there. Suppose p_3 is set at a location as shown in Figure 1b. Then, p_3 can attract two customers, namely o_1 and o_2 , which are the BRNN of p_3 . But, suppose p_3 is set up as shown in Figure 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 Figure 1c is better than that of Figure 1b. The company should better set up convenience store p_3 as shown in Figure 1c compared with Figure 1b.

The problem studied in this paper is 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*) such that when p is placed in R , the size of the BRNN of p is maximized. 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 the BRNN of p 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 Figure 1c and Figure 1d are in R , they have the same BRNN.

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, there are the following two disadvantages. Firstly, this adaptation is infeasible, because there are a *large* (or infinite) number of placements in the data space. Secondly, it cannot summarize the region that the customers are interested in. Problem 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 correspondence region) which have the same BRNN set. Due to the same output (i.e., BRNN set) for different points, the com-

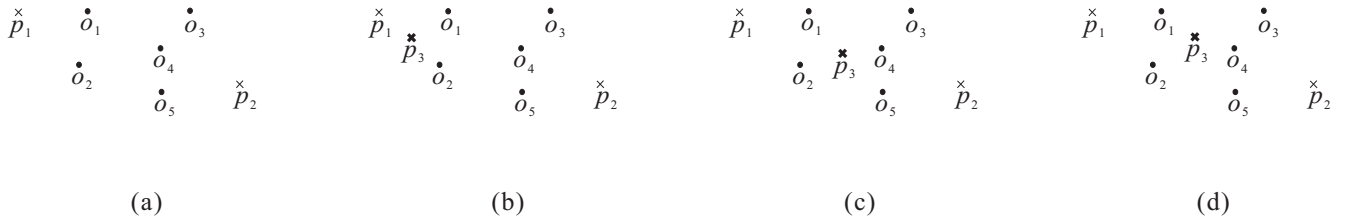


Figure 1: An example

putation in this adaption involves a lot of redundant operations.

Traditional applications which exist in BRNN can also be applied to MaxBRNN [11]. 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 which can attract as many customers as possible. Other service location planning applications can be setting up coffee shops, fastfood restaurants, bank ATMs, gas stations and wireless routers. In addition to daily life 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 is important. In a big event like US presidential campaign, placing police force for security is also important. In a military application, it is also essential to set up some temporary depots for gasoline and food. In profile-based marketing [11], a company wants to set up a new service such as a cell phone plan and similarly it wants to maximize the number of customers who are interested in this plan.

To the best of our knowledge, there is no *efficient* algorithm for MaxBRNN in the L_2 -norm space. There is only one related work which is closely related to ours, namely [4]. [4] solves MaxBRNN in the L_2 -norm 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 this paper, we propose an alternative algorithm called *MaxOverlap* that finds the region which gives the maximum size of BRNN in $O(|\mathcal{O}| \log |\mathcal{P}| + k^2|\mathcal{O}| + k|\mathcal{O}| \log |\mathcal{O}|)$ time where k can be regarded as an integer much smaller than $|\mathcal{O}|$. Compared to the exponential runtime complexity algorithm mentioned above, our algorithm is much more efficient.

Intuitively, *MaxOverlap* is more efficient than the existing work [4] 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.

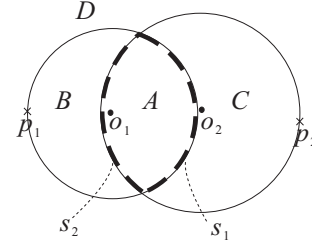


Figure 3: Different regions where different client points are served

Our algorithm runs within 0.1s but the exponential-time algorithm runs for more than 1 day in this dataset.

The rest of the paper is organized as follows. Section 2 formulates the problem MaxBRNN. Section 3 describes the algorithm *MaxOverlap*, and analyzes its performance. Section 4 evaluates the proposed techniques through extensive experiments with real data. Section 5 reviews the previous work. Section 6 concludes the paper with directions for future work.

2. PROBLEM DEFINITION

Suppose we have a set \mathcal{P} of *server points* in a Euclidean space \mathbb{D} (e.g., convenience stores in Figure 1a). We also have another set \mathcal{O} of *client points* in the same space. 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 $w_{max} = \max_{o \in \mathcal{O}} w(o)$, which corresponds to the greatest number of clients at a client point (or location).

We define a *region* to be an arbitrary shape in the space \mathbb{D} . For example, Figure 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 Figure 2a, R_1 , R_2 and R_3 are three regions.

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 , $BRNN(p, \mathcal{P} \cup \{p\}) = BRNN(p', \mathcal{P} \cup \{p'\})$.

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

Since any two possible points in R_1 (e.g., p in Figure 2b and p' in Figure 2c) have the same bichromatic reverse nearest neighbors, R_1 is a consistent region. Similarly, R_2 is a

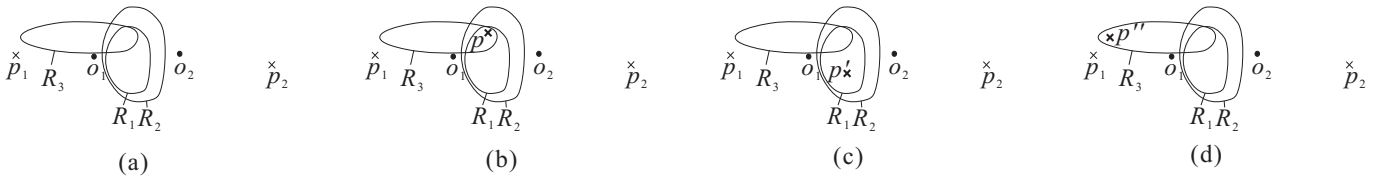


Figure 2: Different regions with the same bichromatic reverse nearest neighbors

consistent region. However, R_3 is not a consistent region because there exists two possible new points p (Figure 2b) and p'' (Figure 2d) in R_3 such that $BRNN(p, \mathcal{P} \cup \{p\}) \neq BRNN(p'', \mathcal{P} \cup \{p''\})$.

Since a consistent region R contains all possible new server points p which have the same bichromatic reverse nearest neighbors, we define the *influence set* [11] of R equal to the bichromatic reverse nearest neighbor of any possible point p in R . This set denotes all client points which are interested in p .

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

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

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

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

DEFINITION 3 (MAXIMAL CONSISTENT REGION). *A consistent region R is said to be a maximal consistent region such that 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 Figure 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 Problem MaxBRNN, we would like to return the maximal consistent region R instead of any non-maximal consistent region because R has its advantage of providing information that all possible points having 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.

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.

There are two challenges in this problem. The first challenge is that, apparently, 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 the maximal consistent region with the greatest influence value.

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

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

For example, Figure 3 shows the same server points and client points as Figure 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 this figure, 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 Figure 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 the appendix. \square

Lemma 1 suggests that we need to consider the region formed by the intersection of some NLCs (like region A) only for problem MaxBRNN. We do not need to consider some regions (like region B) which 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.

In the following, we are interested in maximal consistent regions instead of non-maximal consistent regions. In the following, when we describe regions, we mean maximal consistent regions.

We define some variations on MaxBRNN, namely Max k BRNN, l MaxBRNN and l Max 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 client 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., $\sum_{o \in kBRNN(p, \mathcal{P} \cup \{p\})} w(o)$) is maximized. $I(R)$ is equal to $\sum_{o \in kBRNN(p, \mathcal{P} \cup \{p\})} w(o)$ in the setting with k -BRNN.
2. **l MaxBRNN:** Instead of finding *one* region which gives the greatest size of BRNN, we find l regions which 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. **l Max k BRNN:** l Max 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

The best-known algorithm [4] for problem MaxBRNN runs in exponential time because it heavily relies on searching the *regions* which is exponential in nature. We will propose an algorithm called *MaxOverlap* which utilizes the principle of *region-to-point transformation* and searches a limited number of *points*.

In this section, we present algorithm *MaxOverlap* that follows the properties given in Section 3.1. We also describe how *MaxOverlap* can be extended to problems Max k BRNN, l MaxBRNN and l Max 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 which first create 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 amount 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

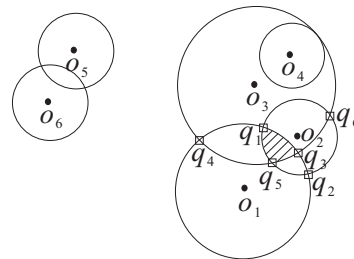


Figure 4: Overlapping NLCs

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

Table 1: Overlap table

with radius equal to $|p, o|$ and insert it into \mathcal{D}' . In addition, we also define 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 which nearest location circle (NLC) is c . Then, based on the *overlapping* relationship among NLCs, we develop an algorithm that finds the region of MaxBRNN query efficiently. For instance, Figure 4 shows six NLCs where we do not show server points $\in \mathcal{P}$ in the figure for clarity. We denote each NLC centered at o_i by c_i for $i = 1, 2, \dots, 6$. 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 .

The overlap relationship gives 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 Figure 4. Table 1 is called an *overlap table*. Each row of the overlap table is called an *entry* in form of $(c, L(c))$. If the context of c is clear, we write the entry as (c, L) .

We define two cases when an NLC c_1 covers another NLC c_2 as shown in Figure 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 (Figure 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 (Figure 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 Figure 4, both NLC c_3 and NLC c_1 cover point q_1 .

LEMMA 2. *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).*

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 Figure 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

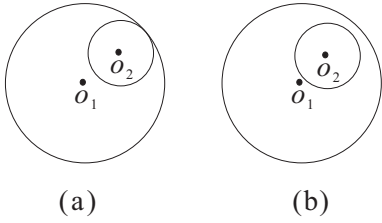


Figure 5: Coverage relationship

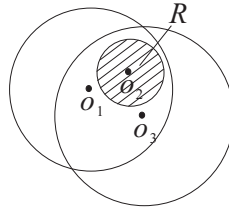


Figure 6: Illustration of the proof of Lemma 3

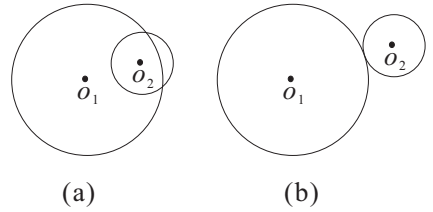


Figure 7: Other overlapping relationships

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 . \square

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 Figure 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. \square

For example, in Figure 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 .

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 for the MaxBRNN query.

In addition to the coverage relationship as shown in Figure 5a, there are other two kinds of overlapping relationships as shown in Figure 7a and Figure 7b. 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. *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 Lemma 3 and Lemma 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 to $|\mathcal{O}|$. To the best of our knowledge, there are no existing polynomial-time algorithms for this MaxBRNN query. We are the first to propose a polynomial-time algorithm for a MaxBRNN query.

The efficiency of our algorithm 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 Figure 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 our problem MaxBRNN is equal to region R which is the intersection of all NLCs in S_q . For example, in Figure 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 our 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 Section 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 Lemma 3 and Lemma 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 Section 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 $O(\beta(|\mathcal{O}|) \cdot |\mathcal{O}|^2)$. It is noted that 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. \square

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 Figure 4, suppose that we know 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 propose algorithm *MaxOverlap* (Algorithm 1) which *interleaves* the execution of these three steps and thus a lot of candidate pairs can be pruned.

First of all, we construct 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, we process each entry (c, L) 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, we iteratively perform the following steps. First, we remove 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 $c' \in L - A$, we compute all intersection points between c and c' . For each intersection point q , we perform a point query to find all NLCs covering q . Let S be the result of this point query. If $W(S) > I_o$, then we update I_o with $W(S)$ and S_o with S . Next, we insert c into A . Finally, we perform an influence-based pruning step to remove all unnecessary NLCs from consideration. We repeat the above computation until all entries in T are exhausted.

Algorithm 1 Algorithm MaxOverlap

```

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

```

EXAMPLE 1. Consider the example shown in Figure 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 the 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 Figure 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 which is a result of the intersection of $\{c_1, c_2, c_3\}$, and the influence value is 3. \square

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

THEOREM 1. *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 1.

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. Details of the computation can be found in [16].

3.3.2 Complexity

Algorithm 1 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 accomplished 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* [8]), 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 $O(|\mathcal{O}|(k + \log |\mathcal{O}|))$.

With similar reasons, in our implementation, we also adopt the R*-tree to support the range querying. 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 sub-steps.

1. We compute the intersection points between the boundaries of c and c' . As shown in Section 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 NLC (i.e., the greatest size of L of an entry (c, L) in the overlap table T). Performing the above sub-steps

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

Table 2: Summary of the real datasets

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

Table 3: Default synthetic dataset cardinalities

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 $O(k|\mathcal{O}|(k + \log|\mathcal{O}|))$.

The overall running time of Algorithm 1 is $O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + |\mathcal{O}| + |\mathcal{O}|\log|\mathcal{O}| + k|\mathcal{O}|\theta(|\mathcal{O}|)) = O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + k|\mathcal{O}|\theta(|\mathcal{O}|))$

THEOREM 2 (RUNNING TIME). *The running time of Algorithm 1 is $O(|\mathcal{O}|\alpha(|\mathcal{P}|) + |\mathcal{O}|\beta(|\mathcal{O}|) + k|\mathcal{O}|\theta(|\mathcal{O}|))$ where k is the greatest size of L of an entry (c, L) in the overlap table T .*

Algorithm 1 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 of nearest neighbor search, range query search and point search to optimize *MaxOverlap*. For example, since $\alpha(|\mathcal{P}|)$ can be accomplished in $O(\log|\mathcal{P}|)$ [8], $\beta(|\mathcal{O}|)$ 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 $O(|\mathcal{O}|\log|\mathcal{P}| + k^2|\mathcal{O}| + k|\mathcal{O}|\log|\mathcal{O}|)$.

Our algorithm 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 $S_{\mathcal{P}}$ and $S_{\mathcal{C}}$, 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 $O(S_{\mathcal{P}} + S_{\mathcal{C}} + k|\mathcal{O}|)$.

3.4 Extension to Max k BRNN, l MaxBRNN and l Max k BRNN

Up to now, we have discussed how *MaxOverlap* solves the problem of MaxBRNN. Here we discuss how *MaxOverlap* can be extended to problems Max k BRNN, l MaxBRNN and l Max k BRNN. We can simply focus on l Max 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. EMPIRICAL STUDY

We have conducted extensive experiments on a Pentium IV 2.2GHz PC with 1GB memory, on a Linux platform. The algorithms were implemented in C/C++. We deployed four real datasets which are available at <http://www.rtreeportal.org/spatial.html>. The summary of the real datasets is shown in Table 2. Specifically, CA, LB, GR and GM contain 2D points representing geometric locations in California, Long Beach Country, Greece

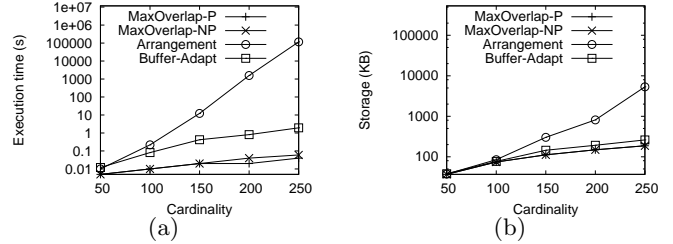


Figure 8: Effect of cardinality (small real data set where $\mathcal{O}=\text{GR}$ and $\mathcal{P}=\text{CA}$)

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 $[0, 10000]$. 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 [17], 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 $[0, 10000]$. In \mathcal{P} , each coordinate follows Gaussian distribution where the mean and the standard deviation are set to 5000 and 2500. In \mathcal{O} , each coordinate follows Zipfian distribution skewed towards 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 2D 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 l Max k BRNN since it is more general than MaxBRNN, Max k BRNN, and l MaxBRNN.

We further divide our proposed algorithm *MaxOverlap* into two algorithms called *MaxOverlap-P* and *MaxOverlap-NP*. *MaxOverlap-P* is our proposed algorithm which 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 [4, 3] called *Arrangement* for problem MaxBRNN. The second one is an algorithm called *Buffer-Adapt* which is originally designed to solve problem MaxBRNN in the L_1 -norm (instead of the L_2 -norm) [9] and is now adapted for problem MaxBRNN in the L_2 -norm. Specifically, in *Buffer-Adapt*, for each client point o , like [9], its nearest location box is constructed (together with its nearest location circle). Then, the box is scaled by $\sqrt{2}$. It is easy to see that, for each client point, its scaled nearest location box 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 [9]. Whenever the

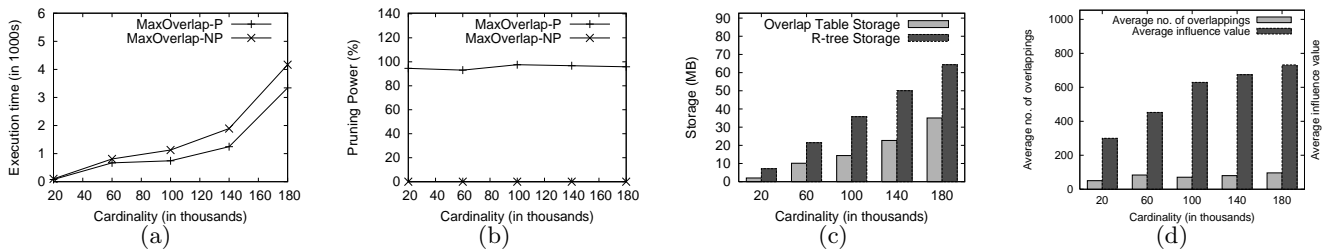


Figure 9: Effect of cardinality (synthetic data set)

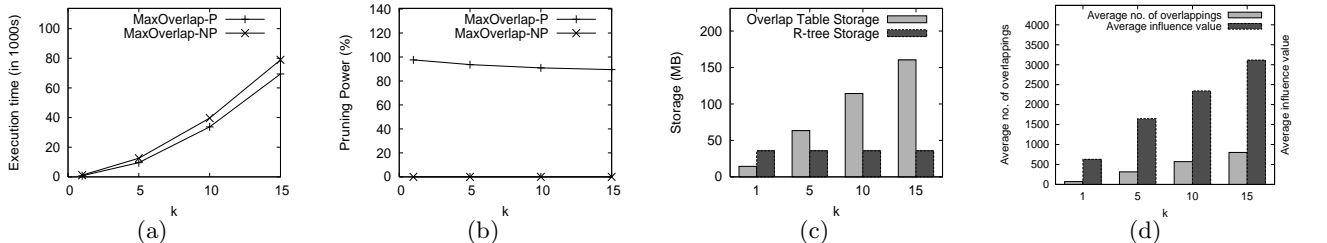


Figure 10: Effect of k (synthetic data set)

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 which 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 bytes¹. 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 occupied by the indexings 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 occupied by *Arrangement* is equal to the memory occupied by the total number of faces between adjacent Voronoi cells used in the algorithm. The memory occupied by *Buffer-Adapt* is equal to the memory occupied by the indexings 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 algorithm *MaxOverlap* in terms of other measurements, namely (3) *pruning power*, (4) *overlap table storage*, (5) *R-tree storage*, (6) *average no. of overlappings* and (7) *average influence value*. *Pruning power* is equal to the number of client points are pruned without any consideration according to the influence-based pruning. *R-tree storage* corresponds to the memory consumption of index-

ing, namely R_P and R_C . *Average no. of overlappings* is the average number of NLCs which overlap with an NLC in the data set. The *average influence value* is the average influence value of regions in the output of the l Max 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 took the average for the results.

We present our results into two parts. The first part, Section 4.1, focuses on the performance comparison among all algorithms. The second part, Section 4.2, focuses on the scalability of our proposed algorithm on datasets with larger cardinality.

4.1 Comparisons

Since algorithm *Arrangement* is not scalable with respect to large datasets, we conduct the comparison experiments with dataset \mathcal{O} of a smaller size, namely 50, 100, 150, 200 and 250, and dataset \mathcal{P} of size equal to $2|\mathcal{O}|$. We adopt the dataset *CA-GR* 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 8a shows that the execution time of all algorithms increases 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 algorithm *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 among NLCs, runs in polynomial-time but *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 a factor of 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 depending on the circles for the L_2 -norm). Secondly, an intersection among a certain set of boxes used in *Buffer-Adapt* does not

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

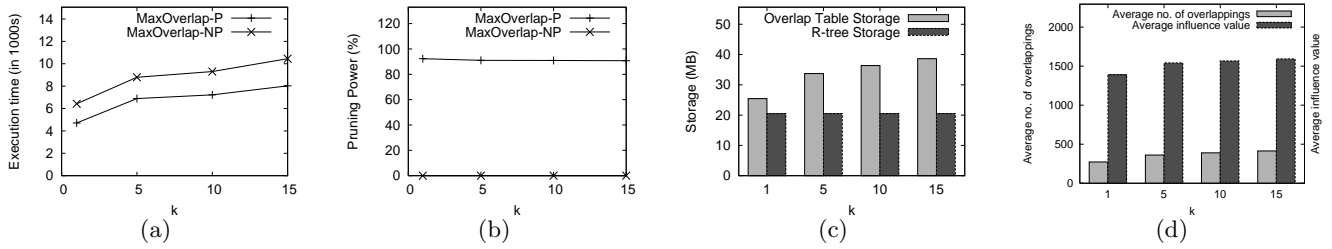


Figure 11: Effect of k (real data set CA-GR)

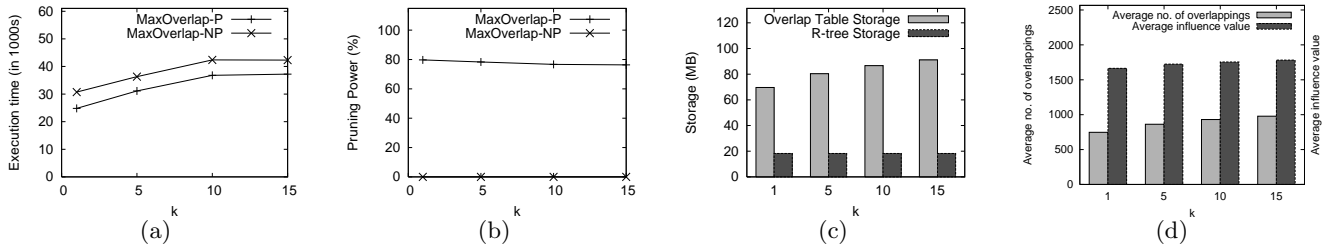


Figure 12: Effect of k (real data set CA-GM)

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 Figure 8b, 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 to 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 [9] 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 of points in the optimal re-

gion 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 $N = 250, 500, 750, 1000$. 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*).

4.2 Scalability

The experiments reported in the previous section demonstrates 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* for 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 of l and k are both 1. Figures 9 and 10 show the results when we vary the cardinality of the dataset (i.e., $|\mathcal{P}|$) and k , respectively.

Effect of Cardinality: Figure 9a shows that the execution times of algorithms *MaxOverlap-P* and *MaxOverlap-NP* increase with $|\mathcal{P}|$ where $|\mathcal{P}| = 2|\mathcal{O}|$. Since *MaxOverlap-P* prunes some client points but *MaxOverlap-NP* does not, *MaxOverlap-P* runs faster than *MaxOverlap-NP*. Figure 9b 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 occupies 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 9c 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 Figure 9d, the average no. of overlappings increases slightly with the car-

dinality 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 Figure 9d, the average influence value increases with the cardinality of dataset because the average number of overlaps is increased.

Effect of k : As shown in Figure 10a, the execution times of *MaxOverlap* increase with k . This is because as k increases, we need to find more nearest neighbors. Figure 10b shows that the pruning power of *MaxOverlap-P* is robust to changes in k . From Figure 10c, 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 10d 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. For 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* (Figure 11 and Figure 12, respectively), while we vary k .

Conclusion: We find that *MaxOverlap* performs efficiently compared with the best-known algorithm, *Arrangement*. It utilizes less memory as well. Our proposed algorithm is scalable to large datasets, but *Arrangement* is not.

5. RELATED WORK

Bichromatic reverse nearest neighbor (BRNN) search was first proposed by [11]. The fastest BRNN algorithm is due to [13]. BRNN search has been used to discover the most “influential” server among a number of servers, which has the largest BRNN set [18]. Other works on this problem are [14, 12]. None of the above works, however, can be applied to solve MaxBRNN, because, as explained in Section 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 [4]. [4] considers MaxBRNN for L_2 -norm space. In [4], this problem is shown to be 3SUM-hard where it is proved that solving a 3SUM problem over dataset of size N requires $\Omega(N^2)$ time. That is, it is impossible that we can solve problem MaxBRNN with a subquadratic algorithm. [4] 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 heavily relies 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. Besides, [3] is an extended version of [4] with similar results. It is noted that there are no empirical studies in [4, 3] and this is the only known method for L_2 -norm.

In addition to L_2 -norm space [4], problem MaxBRNN is also studied in L_1 -norm space [9]. The algorithm in [9] 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 are studied in [5, 19]. Suppose we want to start up a server, [5] proposes to find a location p for this new server in order to minimize the maximum distance between p and any client point $o \in \mathcal{O}$. [19] 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 which minimizes the average distance from each client point to its closest server if a new site is built at this location.

6. CONCLUSION

In this paper, we propose an efficient algorithm called *MaxOverlap* to address problem MaxBRNN. We conducted experiments to show the efficiency of our proposed algorithm.

There are a lot of promising research directions. Firstly, it will be interesting to consider dimensionalities greater than two. Secondly, some recent works [15, 17] consider the capacity of each server. In this paper, following the assumption in the literature, each server can serve as many clients as possible. 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.

Acknowledgements: We are grateful to the anonymous reviewers for their constructive comments on this paper.

7. REFERENCES

- [1] N. M. Amato, M. Goodrich, and E. A. Ramos. Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. In *11th ACM-SIAM Sympos. Discrete Algorithms*, 2000.
- [2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.
- [3] S. Cabello, J. M. Diaz-Banex, S. Langerman, and C. Seara. Facility location problems in the plane based on reverse nearest neighbor queries. In *European Journal of Operational Research*, to appear.
- [4] S. Cabello, J. M. Diaz-Banez, S. Langerman, C. Seara, and I. Ventura. Reverse facility location problems. In

Canadian Conference on Computational Geometry, 2005.

- [5] J. Cardinal and S. Langerman. Min-max-min geometric facility location problems. In *22nd European Workshop on Computational Geometry (EWCG)*, 2006.
- [6] B. Chazelle. Filtering search: A new approach to query-answering. In *SIAM. J. Comput.*, 1986.
- [7] B. Chazelle. New upper bounds for neighbor searching. In *Information and Control*, 1986.
- [8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2000.
- [9] Y. Du, D. Zhang, and T. Xia. The optimal-location query. In *SSTD*, 2005.
- [10] J. M. Kang, M. F. Mokbel, S. Shekhar, T. Xia, and D. Zhang. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*, 2007.
- [11] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*, 2000.
- [12] J. Krarup and P. M. Pruzan. The simple plant location problem: Survey and synthesis. In *European Journal of Operational Research*, 1983.
- [13] I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi. Discovery of influence sets in frequently updated databases. In *VLDB*, 2001.
- [14] B. C. Tansel, R. L. Francis, and T. Lowe. Location on networks: A survey. In *Management Science*, 1983.
- [15] L. H. U, M. L. Yiu, K. Mouratidis, and N. Mamoulis. Capacity constrained assignment in spatial databases. In *SIGMOD*, 2008.
- [16] R. C.-W. Wong, T. Ozsü, P. S. Yu, and A. W.-C. Fu. Efficient method for maximizing bichromatic reverse nearest neighbor. In <http://www.cse.ust.hk/~raywong/paper/maxBRNN-technical.pdf>, 2009.
- [17] R. C.-W. Wong, Y. Tao, A. W.-C. Fu, and X. Xiao. On efficient spatial matching. In *VLDB*, 2007.
- [18] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *VLDB*, 2005.
- [19] D. Zhang, Y. Du, T. Xia, and Y. Tao. Progressive computation of the min-dist optimal-location query. In *VLDB*, 2006.

Appendix: Proof of Lemma 1

Before we give a proof of Lemma 1, we first describe how we can represent each maximal consistent region by two operations of NLCs, namely *intersection* and *exclusion*. We denote an intersection operation and an exclusion operation by \cap and $-$, respectively.

In Figure 3, as we described before, there are four maximal consistent regions, namely regions A, B, C and D . Let U be the universe of the space \mathbb{D} (i.e., a region covering all possible points in \mathbb{D}). Let \mathcal{C} be the set of all NLCs.

Each maximal consistent region can be represented by a series of intersection operations and exclusion operations over some NLCs in \mathcal{C} and the universe U . For example, region A is the region formed by the intersection between the

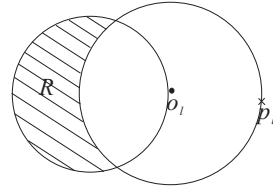


Figure 13: Illustration for the proof of Lemma 1

region occupied by c_1 and the region occupied by c_2 . We can represent region A by $c_1 \cap c_2$. Region B is the region occupied by c_1 excluding the region occupied by c_2 . We represent region B by $c_1 - c_2$. Similarly, region C and region D are represented by $c_2 - c_1$ and $U - c_1 - c_2$, respectively.

If we introduce the universe U to represent each maximal consistent region, regions A, B, C and D are represented by $U \cap c_1 \cap c_2$, $U \cap c_1 - c_2$, $U \cap c_2 - c_1$ and $U - c_1 - c_2$.

Formally, we have the following lemma.

LEMMA 6 (CONSISTENT REGION REPRESENTATION).

Suppose R is a maximal consistent region. There exists a set $\mathcal{C}' = \{c_1, c_2, \dots, c_l\}$ such that $\mathcal{C}' \subset \mathcal{C}$ and R can be represented by $U \diamond c_1 \diamond c_2 \diamond \dots \diamond c_l$ where \diamond is an operation equal to \cap (an intersection operation) or $-$ (an exclusion operation).

Proof: A formal proof can be found in [16]. \square

Note that the representation (in form of $U \diamond c_1 \diamond c_2 \diamond \dots \diamond c_l$) in the above lemma can also denote some regions which may not be consistent. In Figure 3, the region R occupied by the entire NLC c_1 which is not consistent can be represented by $U \cap c_1$.

Besides, it can be proved that there exists a minimal set \mathcal{C}' which can represent a maximal consistent region R . Details can be found in [16]. In the following, we assume that we adopt the minimal set to represent a maximal consistent region R .

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

Proof: Let R be the optimal region which has the greatest influence value. By Lemma 6, R can be expressed by $U \diamond_1 c_1 \diamond_2 c_2 \diamond_3 \dots \diamond_l c_l$ where \diamond_i is the i -th operation equal to \cap (an intersection operation) or $-$ (an exclusion operation).

The lemma claims that all \diamond_i operations are equal to the intersection operations (i.e., \cap). We prove by contradiction. Suppose that there exists an operation which is an exclusion operation (i.e., $-$). Without loss of generality, we assume that the last operation \diamond_l is equal to $-$. Figure 13 shows an example that \diamond_l is $-$ where c_l is the NLC of o_l . Let $S = BRNN-R(R)$. Note that S does not contain o_l because R is outside circle c_l .

Let \bar{o}_l be \cap . There exists another region R' represented by $U \diamond_1 c_1 \diamond_2 c_2 \diamond_3 \dots \bar{o}_l c_l$. If R' is consistent, $BRNN-R(R') = S \cup \{o_l\}$. Thus, $I(R') > I(R)$. This leads to a contradiction that R is the region with the greatest influence value. If R' is not consistent, there exists a consistent region R'' covered by R' where (1) R'' is represented by $U \diamond_1 c_1 \diamond_2 c_2 \diamond_3 \dots \bar{o}_l c_l \diamond_{l+1} c_{l+1} \dots \diamond_{l+N} c_{l+N}$, (2) c_i is an NLC for $i \in [l+1, l+N]$ and (3) $\diamond_i = \cap$ for $i \in [l+1, l+N]$. Similarly, we deduce that $I(R'') > I(R)$, which leads to a contradiction. \square