Geoinformatica (2016) 20:351–384
DOI 10.1007/s10707-015-0239-5

Finding optimal region for bichromatic reverse nearest neighbor in two- and three-dimensional spaces

Huaizhong Lin[1] · **Fangshu Chen**[1] · **Yunjun Gao**[1] · **Dongming Lu**[1]

Received: 17 April 2015 / Revised: 27 September 2015
Accepted: 5 October 2015 / Published online: 19 October 2015

Abstract The MaxBRNN problem is to find an optimal region such that setting up a new service within this region might attract the maximum number of customers by proximity. The MaxBRNN problem has many practical applications such as service location planning and emergency schedule. In typical real-life applications the data volume of the problem is huge, thus an efficient solution is highly desired. In this paper, we propose two efficient algorithms, namely, OptRegion, and 3D-OptRegion to tackle the MaxBRNN problem and MaxBRkNN in two- and three-dimensional spaces, especially for the 3D-OptRegion, we propose a powerful pruning strategy Fine-grained Pruning Strategy to reduce the searching space. Our method employs three optimization techniques, i.e., sweep line (sweep plane in a three-dimensional space), pruning strategy (based on upper bound estimation), and influence value computation (of candidate points), to improve the search performance. In a three-dimensional space, we additionally use a fine-grained pruning strategy to further improve the pruning effect. Extensive experimental evaluation using both real and synthetic datasets confirms that both OptRegion and 3D-OptRegion outperform the existing algorithms significantly under all problem instances.

Keywords Spatial databases · Reverse nearest neighbor query · Three dimensional space

✉ Fangshu Chen
youyou_chen@foxmail.com

Huaizhong Lin
linhz@zju.edu.cn

Yunjun Gao
gaoyj@zju.edu.cn

Dongming Lu
ldm@zju.edu.cn

[1] College of Computer Science and Technology, Zhejiang University, Hangzhou, People's Republic of China

1 Introduction

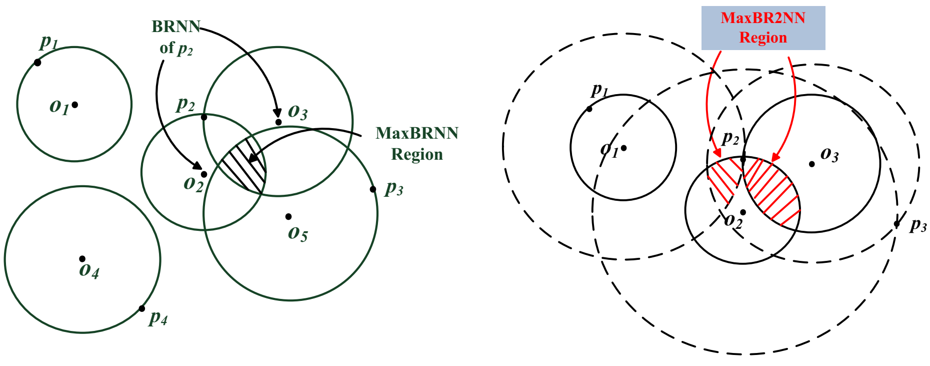
Given a database, an RNN (Reverse Nearest Neighbor) query returns the data points that have a given query point as their nearest neighbor (The RNN query was first introduced in [15]). A BRNN (Bichromatic Reverse Nearest Neighbor) is the bichromatic version of RNN, in which all data points consist of the service point set P and the customer point set O. For a service point $p \in P$, a BRNN query finds all the points $o \in O$ whose nearest neighbor in P is p. Those customer points o in O constitute the influence set of p and the influence value of p equals to the cardinality of the influence set. For example, in Fig. 1a, for a service point p_2, its BRNN, i.e., the influence set, is $\{o_2, o_3\}$.

The MaxBRNN problem [4, 5] aims to find the region S in which all the points have the maximum influence value, namely the cardinality of BRNN set of all points p in S is maximized in a space. The MaxBRNN can be regarded as an optimal region search problem and has attracted much research efforts.

The MaxBRNN problem has many interesting real life applications, such as service location planning and emergency schedule. For example, in Fig. 1a, there are five customer points o_1 to o_5 and four stores p_1 to p_4 in a city. Now a company wants to set up a new store and the objective is to find a location that can attract as many customers as possible under the assumption that the customers are more interested in visiting a convenient store based on the distances. We draw a circle for each customer point o_i ($1 \leq i \leq 5$), centered at o_i and the distance between o_i and its nearest store as radius. The MaxBRNN problem is translated to find the region with maximum overlapped circles, which is the intersection of three circles of o_2, o_3, and o_5, to set up a new store.

There have been several algorithms [5, 17, 26, 29] proposed to deal with the MaxBRNN problem in the literature. However, all these algorithms degrade significantly as the dataset becomes very large, hence an efficient solution is highly desired.

The MaxBRNN problem assumes that each customer only access his nearest service. However, in reality, a customer may choose to access his k-nearest services. To handle this situation, MaxBRNN can be generalized to the MaxBRkNN problem which finds an optimal region such that setting up a service in this region guarantees the maximum number of customers who would have this service as one of their k-nearest services.



(a) An example of BRNN and Max-BRNN problem (b) An example of MaxBR2NN problem

Fig. 1 Examples of MaxBRNN in two-dimensional space

In this paper, we propose two efficient algorithms called OptRegion and 3D-OptRegion to solve the MaxBRNN an-d MaxBR***k***NN problem in two- and three-dimensional spaces. Our methods employ three optimization techniques, i.e., sweep line (sweep plane in a three-dimensional space), pruning strategy (based on upper bound estimation), and influence value computation (of candidate points), to improve the search performance.

Our major contributions (excluding the contributions in conference version [16]) can be summarized as follows:

1. We extend the algorithm OptRegion to solve the MaxBR***k***NN problem in Euclidean Space.
2. We propose an efficient algorithm, namely, 3D-OptRegion, to solve the Max-***3D***-BRNN problem in three-dimensional space, which can be applied for arbitrary *Lp*-norm spaces. The sweep (plane) technique is adopted in the algorithm to find overlapping spheres quickly.
3. We propose an effective fine-grained pruning strategy in three-dimensional space, by which the majority of candidate points can be pruned without evaluation.
4. We give out the correctness analysis of algorithm OptRegion and 3D-OptRegion, and we analyze the accuracy of upper bound estimation technique used in OptRegion.
5. We conduct extensive experiments with both real and synthetic data sets to demonstrate the performance of our proposed algorithms.

This paper is a significant extension to its preliminary conference version [16]. Compared to the preliminary version, we extend the query to the three-dimensional space and BR*k*NN query ($k \geq 1$), propose an effective pruning strategy for the three-dimensional space, and present the theoretical analysis and experimental confirmation of the accuracy of our upper bound estimation.

The MaxBR***k***NN problem is necessary in real life applications, for example, when planning a new convenience store, considering that the customers can not only be attracted to its nearest store, but also the sub-nearest and the k-nearest ones, then the MaxBR***k***NN query becomes very necessary.

The Max-3D-BRNN problem also has many real life applications, we give three typical examples here: **1)** In some emergency applications such as in the earthquake in China, we often need fast response for the MaxBRNN search to quickly place the supply/service centers for rescue or relief jobs, the location of the customer points $o \in O$ may be changing dynamically, thus we have to consider time as another dimension since at different moments the locations are different. **2)** Another example is sensor network distribution. Suppose that we plan to add a new cluster node for the sensors to communicate with each other and the sensors locate in different locations and altitude in a mountain, naturally the locations have to be described in three-dimensional space. **3)** In the location planning example, a customer considers not only the distance to the store, but also the time driving there, and we can also consider customer preferences as other dimensions. Thus, it's significant to extend the MaxBRNN problem to three-dimensional space or even higher spaces. In this paper, we propose an effective pruning strategy for the three-dimensional space MaxBRNN problem and it is easy to extend to high dimensional space (we leave the MaxBRNN problem in high dimensional space as one of our future works).

The rest of the paper is organized as follows. A survey of related work is given in Section 2. Section 3 formulates the MaxBRNN, MaxBR*k*NN, and Max-***3D***-BRNN problems. Section 4 and Section 5 describe the algorithms for MaxBRNN in two- and three-dimensional spaces respectively. Section 6 analyzes the time complexity and correctness of the algorithms. Section 7 evaluates the proposed algorithm and the pruning strategy through extensive experiments with real and synthetic datasets, and we conclude the paper in Section 8.

2 Related work

There are two types of RNN queries, namely, monochromatic and bichromatic RNN [16]. In the monochromatic case, all points are of the same category. In the bichromatic case, the points are of two different categories, such as services and customers. The original RNN problem has been studied in the literature [21–23] and the proposed algorithms are mainly based on some space-partition and pruning strategies. In recent years, the Bichromatic RNN (BRNN) problems have been studied extensively in the road network the ad-hoc network and the continuous environment [3, 6, 10, 12, 14, 18, 20, 25]. In [28] they also generalize the BRNN problem to the land surfaces scenario.

The MaxBRNN problem, which maximizes the number of potential customers for the new service, was first introduced by Cabello et al. in [4, 5], where they call it MAXCOV problem and present a solution for the two-dimensional Euclidean space. They also study other optimization criteria in BRNN queries: MINMAX, which minimize the maximum distance to the associated customers, and MAXMIN, which maximize the minimum distance to the associated customers. MaxBRNN query is challenging in that there exists an infinite number of candidate locations where the new service may be built. Wong et al. [26] define that the MaxBRNN is to find a maximal consistent region containing the optimal locations and present an algorithm called MaxOverlap to solve it. The Algorithm MaxOverlap in [26] utilizes a technique called region-to-point transformation, which is also adopted in our OptRegion algorithm. It transforms the optimal region search problem to an optimal intersection point search problem in order to avoid searching an exponential number of regions. Nevertheless, the MaxOverlap does not scale well because the computation of optimal intersection points is expensive.

Wong et al. in [27] extend the MaxOverlap algorithm [26] for L_p-norm in the two and three-dimensional space. They also extend MaxOverlap to solve the MaxBRkNN problem by considering k-nearest neighbors instead of only one nearest neighbor. This increases the number of intersection points to be computed, hence leading to much more performance deterioration when k is large. In [27], they also compared their algorithms with the Buffer-Adapt algorithm [10] which is originally designed to solve problem MaxBRNN in the L_1-norm (Manhattan Distance space). Algorithm MaxSegment in [17] tries to speed up finding the optimal intersection point by checking intersection arcs of circles in two and three-dimensional space. Experimental result in [17] shows that MaxSegment algorithm is faster than the MaxOverlap algorithm [27] in all cases. However, since they don't use any pruning strategy, there may be a lot of intersection arcs that need to be checked, which is very time-consuming.

Zhou et al in [29] generalize the MaxBRNN problem to reflect the real world scenario where customers may have different preferences for different service sites, and present an efficient algorithm called MaxFirst to solve the genera-lized MaxBRkNN problem. The MaxFirst utilizes the branch-and-bound principle, and partitions the space into qua-drants recursively and computes the upper and lower bounds of the size of a quadrant's BRNN. The algorithm then retrieves only in those quadrants that potentially contain an optimal region.

Experimental results show that MaxFirstis much more efficient than the MaxOverlap. Nonetheless, in some situations, there may also be a lot of quadrants need to be processed, and it has poor scalability.

Z. Chen et al in [7] extend the MaxBRNN problem to the Road Network space. P. Ghaemi et al [13] solve the MaxBRNN problem in the *Spatial Network* instead of *Lp-norm* space, they propose two efficient algorithm-EONL and BONL to solve the problem. F. Chen et al in [8] focus on the MaxBRNN problem in the capacity constraint scenario in the Euclidean space. [18, 25] extend the BRNN query to the ad-hoc network and mobile system, they adopt the communication techniques in ad-hoc network to solve the BRNN query.

The *maximizing range sum* (MaxRS) problem studied in [9, 24] also aims to maximize a region's influence value. However, the problem has significant differences with our MaxBRNN problem. First, the region's shape and size is fixed in the MaxRS problem, but in the MaxBRNN problem we know nothing about the result region (or regions).

Second, in the MaxRS problem the extend of the covered region is already known, however, in our problem, we have to calculate the region's radius by computing the RNNs which makes our problem much more complicated.

3 Preliminaries

In this Section, we formalize our problem studied in this paper, and then discuss the region-to-point transformation that is employed to tackle our problem.

3.1 Problem definitions

Suppose we have a set P of service points and a set O of customer points in a space D. Each point $o \in O$ has a weight $w(o)$, which is used to represent the number of customers or importance.

For a point $o \in O$, kNN(o,P) represents the set of the top-k points in P that are nearest to o. For a point s in D ($s \in P$ or not), BRkNN(s,O,P$\cup\{s\}$) represents the set of points in O that take s as one of their k-nearest neighbors in P$\cup\{s\}$. For simplicity, we take $k=1$ in the following discussion, which can be easily extended to the case $k>1$.

Definition 1. (Influence Set/Value) Given a point s, we define the influence set of s to be BRNN(s,O,P$\cup\{s\}$). The influence value of s is equal to $\sum_{o \in \mathrm{BRNN}(s,\, O,\, P \cup \{s\})} w(o)$.

Definition 2. (Nearest Location Region, NLR) Given a customer point o, the nearest location region R of o is defined to be the region centered at o and containing all the point s with $d(o,s) \le d(o,\mathrm{NN}(o,P))$. NN($o$,P) is the nearest neighbor of o in P and $d(x,y)$ is the distance between points x and y. The weight of NLR R $w(R)$ equals to the weight of o.

In the Definition 2, we adopt the notation NLR to capture the general case when considering *Lp*-norm space and three-dimensional space. Minkowski distance can be adopted in computing $d(x,y)$. When considering Euclidean distance, i.e., the *L2*-norm space, in two-dimensional space, the NLR is a circle around o with radius $d(o,\mathrm{NN}(o,P))$.

Definition 3 (Consistent Region, [26]) A region R is said to be *consistent* if the following condition holds: $\forall s,s' \in R, s,s' \notin P$, the Influence value of s is equal to s'.

Following the definition in [26], given a consistent region R, the influence value of R is denoted as $\boldsymbol{I(R)}$ and defined as $\boldsymbol{I(R)}=\sum_{o \in \mathrm{BRNN}(s,\, O,\, P \cup \{s\})} w(o)$, where s denotes an arbitrary new server point in R.

Based on Definition 3 and the above description, we can give out the definition of Maximum Consistent Region as follows: given a consistent region R, we say that R is a maximal consistent region, if there does not exist another consistent region R_0 satisfying the following conditions: (1) $R \subset R_0$, and (2) $\boldsymbol{I(R)=I(R_0)}$**,** we also call the Maximum Consistent Region the Optimal Region.

Problem 1 (MaxBRNN) Given a set P of service points and a set O of customer points in a space D, we want to find an maximum consistent region (optimal region) S such that all points in S have the maximum influence value.

For example, in Fig. 1a, the MaxBRNN (optimal) region S is the intersection of three NLRs o_2, o_3, and o_5, and the influence value of S is the sum of the three NLRs' weights. Informally speaking, the MaxBRNN returns the region with maximum overlapped NLRs.

When considering the k-nearest neighbors of a customer point, the customer point o is associated with a set of k NLRs. The ***i***th NLR is centered at o and contains all the points with distance to o less or equal than the distance between o and its ***i***th nearest neighbor.

Problem 2 (MaxBR***k***NN) ***BRkNN*** of $s \in$P, denoted by ***BRkNN** (s,* P*)*, is a set of points $o \in$ O such that s is one of the ***k*** nearest neighbors of o in P. In MaxBR***k***NN, we want to find the region ***R*** (or area) such that, if a new server s is set up in R, the size of ***BRkNN*** of s (i.e., $\sum_{o \in \mathrm{BR}k\mathrm{NN}(s,\ \mathrm{O},\ \mathrm{P} \cup \{s\})} w(o)$) is maximized. ***I(R)*** is equal to $\sum_{o \in \mathrm{BR}k\mathrm{NN}\ (s,\ \mathrm{O},\ \mathrm{P} \cup \{s\})} w(o)$ in the setting with ***BRkNN***.

Problem 3 (Max-***3D***-BRNN) Given two three-dimensional datasets set P of service points and a set O of customer points in a 3D-space D , the three-dimensional BRNN of $s \in$P , denoted by ***3D-BRNN** (s,* P*)*, is a set of points $o \in$O such that s is one of the nearest neighbors of o in P. In Max-***3D***-BRNN, we want to find a ellipsoidal region ***R*** such that, if a new server s is set up in R, the size of ***3D-BRNN*** of s (i.e., $\sum_{o \in 3D\text{-BRNN}(s,\ \mathrm{O},\ \mathrm{P} \cup \{s\})} w(o)$) is maximized. ***I(R)*** is equal to $\sum_{o \in \boldsymbol{3D}\text{-BRNN}(s,\ \mathrm{O},\ \mathrm{P} \cup \{s\})} w(o)$ in the setting with ***3D-BRNN***.

In Figs. 1b and 2, we give examples of MaxBR***k***NN and Max-***3D***-BRNN query. Figure 1b demonstrates the MaxBR2NN query, for each point $o \in$ O we draw the NLR to its 2NNs, and for each point $p \in$P its ***BRkNN*** set is larger than BRNN, for example, for p_2, its BR2NN$=\{o_1, o_2, o_3\}$. Consequently, the result of MaxBR2NN is more complicated than MaxBRNN, as described in Fig. 1b the MaxBR2NN regions are the red shaded region. Figure 2 describes the same example as Fig. 1a except that it is in the three-dimensional space, the Max-***3D***-BRNN (suppose $\boldsymbol{k=1}$) region is the ellipsoidal region enclosed by three different cambered surface (described in red, blue and green) in Fig. 2.

3.2 Region-to-point transformation

The MaxOverlap algorithm in [26] uses region-to-point transformation, which is also used in our algorithm OptRegion, to solve the MaxBRNN problem. The optimal region search problem is transformed into finding the maximum influence intersection point between any two NLRs, and the maximum influence intersection point can subsequently be mapped into the optimal region. We adopt the following two lemmas from [26], and the proof is omitted here.

Lemma 1. The maximum consistent region (optimal region) returned by the MaxBRNN query can be represented by an intersection of multiple NLRs.

Lemma 2. Let S be the maximum consistent region returned by the MaxBRNN query. If S is the intersection of more than one NLR, then there must exist two NLRs and at least one of their intersection points is contained in S.

Based on lemma 1 and 2, the MaxOverlap algorithm first computes all the intersection points of every pair of NLRs. These intersection points can be regarded as candidate points.

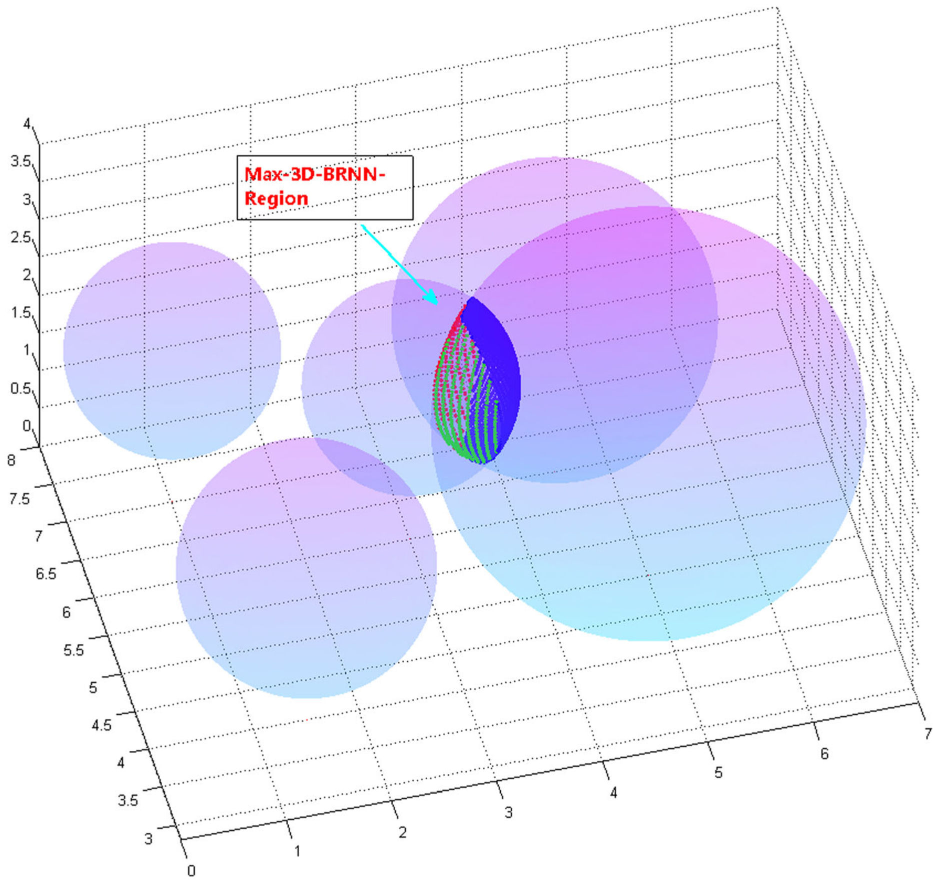


Fig. 2 An example of Max-***3D***-BRNN problem

Then, for each intersection point, the algorithm performs a point query to find all the NLRs covering the point and computes its influence value, from which the point with maximum influence value is found. Last, the optimal region can be identified from the maximum influence intersection point. It is the major drawback in MaxOverlap algorithm that the algorithm needs to perform a point query for every intersection point to compute influence value. The point query takes up the majority of the computation effort.

4 Algorithms in two-dimensional space

Algorithm OptRegion consists of three steps. First, NLRs are constructed for every customer point. Then, all intersecting NLRs are detected, and the upper bound of influence values for all NLRs are estimated. Last, the exact influence values of candidate points are computed and the point with maximum influence value is found.

We first describe our algorithms in two-dimensional space in Section 4. We depict several techniques used in OptRegion in Section 4.1 to 4.3, and then give the overall algorithm OptRegion in Section 4.4. The extension to three-dimensional space is described in Section 5.

We omit the detailed description of our influence value computation technique here, since it is similar to the algorithm MaxSegment in [17] despite some differences in details and expressions.

4.1 Sweep line

The method to find intersecting NLRs in algorithm MaxOverlap [26, 27] and MaxFirst [29] is as follows. First, an R*-tree is built for all NLRs. Then a range query against the R*-tree for each NLR is performed to find all its intersecting NLRs. But it will take a lot of time to construct R*-tree and perform range query, especially when there is a large amount of customer points.

We adopt the sweep line approach to find all intersecting NLRs, which localize the search range of intersecting NLRs to speed up the processing. Sweep line approach is widely adopted in a variety of computational geometry problems, such as line segment intersection, voronoi diagram construction et al [2].

In sweep line and subsequent prune strategy techniques, we model an NLR by its minimum bounding rectangle (MBR) (as shown in Fig. 5), which means the result is a kind of upper bound estimation. The exact result can be gained in the influence value computation process.

We define the y-interval of an NLR to be its orthogonal projection onto the y-axis, referring [16] for more details. When the y-intervals of a pair of NLRs do not overlap, then they cannot intersect. Hence, only the pairs of NLRs whose y-intervals are overlapping need to be tested for intersection. It is obvious that there exists a horizontal line that intersects both NLRs whose y-intervals are overlapping. So, to find these pairs we imagine sweeping a line downwards over the plane, starting from a position above all NLRs. While we sweep the imaginary line, we keep track of all NLRs intersecting with it, so that we can find the intersecting pairs we need.

The line is called the sweep line and the status of the sweep line is the set of NLRs intersecting with it, as shown in Fig. 3a. The status changes while the sweep line moves downwards, but not continuously. We use the balanced binary search tree [19] in status structure implementation. The details of the technique can be found in the conference version in [16] and we omit it here.

Lemma 3 [16]**.** To find intersecting NLRs correctly, the status structure need to be updated only when the sweep line reaches the event points.

The proof of Lemma 3 can be found in the conference version in [16]. We maintain an event queue to store all the event points, which are the top and bottom points of all NLRs. The event points are ordered by their y-coordinates downwards. When the sweep line moves downward, the event point that is the highest below the sweep line is processed. If two event points have the same y-coordinate, the process order can be arbitrary and the result is the same.

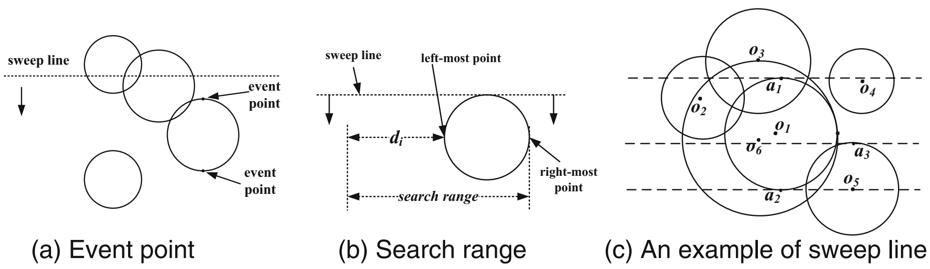


Fig. 3 Sweep line

Lemma 4 [16]**.** When the sweep line reaches the top point of an NLR R, let *xleft* and *xright* be the *x*-coordinate of the left-most and right-most point of NLR *R*, and *di* be the maximum diameter of all NLRs which are currently in the status structure (as shown in Fig. 3b). In the status structure, the NLRs whose left-most points are out of the range [*xleft*−*di, xright*] will not intersect with *R*.

The proof of Lemma 4 can be found in the conference version in [16] and we omit it here. Based on lemma 4, we can perform a range query of [*xleft*−*di*, *xright*] in the status structure to find all NLRs intersecting with NLR *R* at the moment *R* is inserted into the status structure. When the sweep line reaches the bottom event point of *R*, then all intersecting NLRs with *R* have been found and *R* can be deleted from the status structure safely.

Example 1. In Fig. 3c, R_i is the NLR of customer point o_i. When the sweep line reaches point a_1 (the top event point of NLR R_1), we can get the intersecting NLR list of R_1 at this moment is $I_{R1}=\{R_2,R_3,R_4,R_6\}$, for their left-most points locate in R_1's search range. At the same time, R_1 is added to the intersecting NLR list of R_2, R_3, R_4 and R_6. After R_1 is inserted into the status structure, the sweep line moves down and reaches point a_3 (the top event point of NLR R_5), the set I_{R1} is updated to be $I'_{R1}=\{R_2, R_3, R_4, R_5,R_6\}$. Finally, when the sweep line reaches a_2 (the bottom event point of NLR R_1), we delete R_1 from the status structure and the resulting intersecting NLR list of R_1 is I'_{R1}. Here, we have to mention that the resulting I'_{R1} is a superset of the exact intersecting NLR list, e.g., R_4 in I'_{R1} is actually not intersecting with R_1. The exact result can be gained in the subsequent influence value computation process.

Based on the above discussion, we can see how the sweep line algorithm works and the details of the algorithm is omitted here (which can be found in the conference version in [16]).

4.2 Pruning strategy

We divide an NLR into eight subspaces evenly, illustrated as NLR R_1 in Fig. 4, by four dotted lines, horizon, vertical, and two lines intersecting with horizon line by 45° angle. For each subspace, we sum the weight of the NLRs intersecting with the current NLR and locate in that subspace (including the NLR itself), noted as ws_1 to ws_8. Although the partitioning scheme with eight subspaces is described here, other partitioning scheme such as no partitioning, 4 or 16 partitioning can also be used. The selection of partitioning scheme is based on the balance between pre-computing cost and pruning power. With more partitioning subspaces, the pruning power is better but we need much more pre-computing cost (Fig. 5).

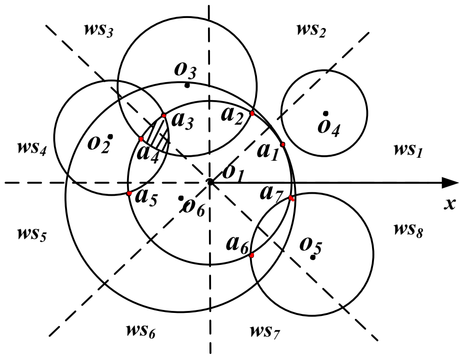


Fig. 4 An example of OptRegion

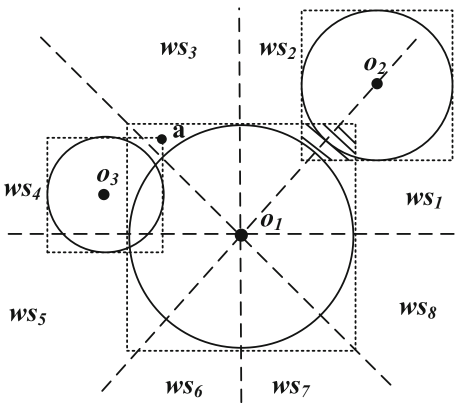


Fig. 5 Upper bound estimation

Lemma 5 [16]**.** The weight sum ws_i ($1 \le i \le 8$) is the upper bound of the maximum influence value of the intersection points in that subspace. $wsMax = \max\{ ws_i, 1 \le i \le 8\}$ is the upper bound of maximum influence value of the intersection points in an NLR.

Lemma 6 [16]**.** Suppose *max* is the maximum influence value found so far, then any NLRs with *wsMax* less than *max* can be safely pruned without further consideration.

While performing upper bound estimation, the intersection points of NLRs are not necessary to be computed. We only need to decide in which subspaces an intersecting NLR may be located with respect to the current NLR by comparing of the customer points' coordinate values, avoiding the time consuming computation of intersection points and corresponding influence values [26]. When checking the intersecting NLR's location, we use MBRs to describe NLRs (Fig. 5) which can reduce the computation complexity without loss of correctness [16].

Based on lemma 5, after we have computed the *wsMax* value of all NLRs, NLRs are processed by the descending *wsMax* order. The NLRs with greater *wsMax* have higher probability containing the optimal intersection point and there are more chances to prune NLRs.

4.3 Influence value computation

We adopt a novel approach to compute the influence value of intersection points quickly, which need much less computation cost than the point query in MaxOverlap [26]. The details of the computation method can be found in [16].

We describe our algorithm in Euclidean space for ease of expression and the algorithm can be easily extended to *Lp*-norm metric space. The only requirement of our algorithm is that the NLR should be convex. The shapes of NLRs for different Minkowski metric are given in Fig. 6.

Definition 5. (Intersection Arc) If R intersects with R_i, then the arc in the perimeter of R within the intersection region is called intersection arc in R. The weight of the intersection arc is equal to the weight of R_i, i.e. $w(o_i)$.

Based on the above explanation, NLR R and R_i are convex in the space. The following lemma explains the intersection arc between R and R_i.

Lemma 7 [16]. If R intersects with R_i, there is only one intersection arc in the perimeter of R with respect to R_i.

Considering an NLR R, all the NLR R_i that intersect with R are mapped to the arc (α_i, β_i) in R. Then, the computation of influence value of an intersection point a in R is transformed to compute the weight sum of the intersection arcs that contain the intersection point a.

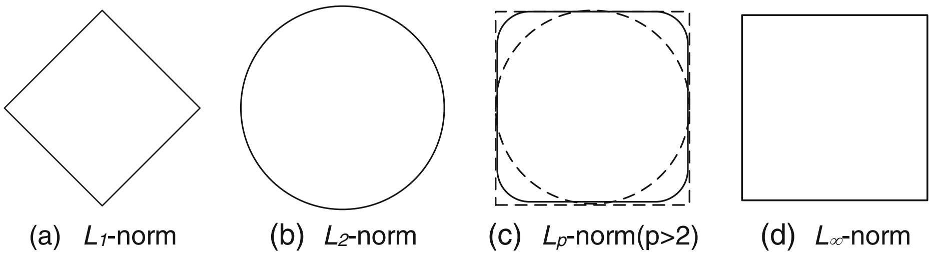


Fig. 6 NLRs for different Minkowski metric

Lemma 8 [16]. Let R be an NLR, the influence value of an intersection point a in R equals to the weight sum of all intersection arc containing a in R (include R itself).

We illustrate briefly the influence value computation technique and the algorithm OptRegion by Example 2 in Fig. 4.

Example 2. In Fig. 4, there are 6 NLRs R_i centered at $o_i \in O$ $(1 \leq i \leq 6)$. The R_1 intersects with R_2, R_3, R_5, and R_6. The intersection points are a_1 to a_7. R_1 intersects with R_6 by arc a_1a_1 in R_1 (it is the whole circle of R_1), with R_3 by arc a_2a_4, with R_4 by a_3a_5, and with R_5 by a_6a_7. To find the point with maximum influence value among a_1 to a_7, we only need compute the maximum overlap weight of the arc a_1a_1, a_2a_4, a_3a_5, a_6a_7, and plus the weight of R_1.

Suppose the weight of each o_i is equal to 1. In Fig. 4, we compute that ws_1=3, ws_2=4, ws_3= 4, ws_4=4, ws_5=3, ws_6=2, ws_7=3, ws_8=3, so the *wsMax* value of R_1 is 4. Similarly, the *wsMax* of R_2 to R_6 are: 4, 4, 3, 3, 4. So, we will first deal with NLR R_1 with the *wsMax* value 4. We can compute the exact influence values of points a_3 and a_4 are 4 in Fig. 4, so, we can prune all the other NLRs without further influence value computation since there is no NLR with the *wsMax* value larger than 4. Consequently the points a_3 and a_4 have the maximum influence value and are corresponding to the optimal region shown as a shaded region in Fig. 4 which is the region overlapped by R_1, R_2, R_3, and R_6.

4.4 OptRegion

Now, we give our algorithm OptRegion in Algorithm 1. The detailed explanation of the algorithm can be found in the conference version in [16].

Algorithm 1 OptRegion

```
input : O := set of customer points
    P := set of service points
output: S := optimal region presented by the overlapping NLRs
1 for each o∈O construct an NLR for o
2 call SweepLine
3 compute the wsMax for all NLRs
4 sort all the NLRs by the wsMax value
5 choose the NLR R with the largest weight
6 s←any point in R, max←w(R)
7 for every R in the NLR set by descending wsMax order
8 if the wsMax of R is less or equals to max
9 break
```

```
  10 influence value computation for all intersection points in $R$,
if the computed value is greater than $max$, the $max$ is replaced and $s$ is
replaced with the intersection point for the corresponding $max$
  11 find the intersection of all NLRs containing $s$, put the id of
these NLRs into $\boldsymbol{S}$
  12 return $\boldsymbol{S}$
```

5 Extension to three-dimensional case

In this section, we will first introduce some techniques we utilize in three-dimensional space, and then we extend the OptReion algorithm to three-dimensional space naturally. At last, we develop a more efficient algorithm with a fine grained pruning strategy, defined as OptRegion with FP, for the three-dimensional space.

5.1 Intersection arc

In the two-dimensional space, the correctness of the algorithm depends on two concepts. The first concept is that the optimal region in the two-dimensional case can be represented by the intersection of multiple nearest location regions (NLRs). The second concept is that all the arcs, each of which is generated by the boundaries of two intersecting NLRs, can be used to find the optimal region.

In the three-dimensional space, we adapt the above two concepts due to some differences between the two-dimensional and the three-dimensional cases. For the first concept, in the three-dimensional case, the optimal region can be represented by the intersection of multiple NLRs, which are spheres here. For the second concept, in the three-dimensional case, the boundaries of two intersecting NLRs generate a circle instead of an arc which is generated by two NLRs in the two-dimensional case. It is pointed in [17] that three spheres, intersecting each other, can generate an arc in the three-dimensional space. Based on all arcs generated by any three spheres intersecting each other, we can find the optimal region accordingly.

Here, we have to consider two special cases. First, if any two spheres in the searching space don't intersect with each other, then the optimal region is the sphere with the largest weight. Second, if there don't exist three spheres intersecting each other, the optimal region can be generated by two intersecting spheres with the largest weight. Since these two cases are trivial and easy to deal with, we assume that there exist three spheres intersecting each other in the searching space.

In a three-dimensional space, the generation of intersection arc is based on three NLRs that intersect each other. Suppose that we are given three NLRs, R_1, R_2, and R_3, each of which intersects with the other two NLRs. The boundary of the intersection of two NLRs R_1 and R_2 (as shown in Fig. 7a) is a circle, c_{12}, which is on a plane denoted by α. Circle c_{12} is called the (R_1, R_2)-*circle* and plane α is called the (R_1, R_2)-*plane*. We say that this plane α is generated from the two NLRs, R_1 and R_2. The plane α intersects with another NLR R_3 and generates a circle c_3 (see Fig. 7b). Circle c_3 is called the R_3-*circle* on plane α. Now, we have two circles c_{12} and c_3 (on the plane α), which have a similar scenario as that in the two-dimensional case. Both circles c_{12} and c_3 generate intersection arcs on the plane α (see Fig. 7b). The detailed computation of the (R_1, R_2)-*plane*, the (R_1,R_2)-*circle* and R_3-*circle* on plane α in a three-dimensional space is given in [17].

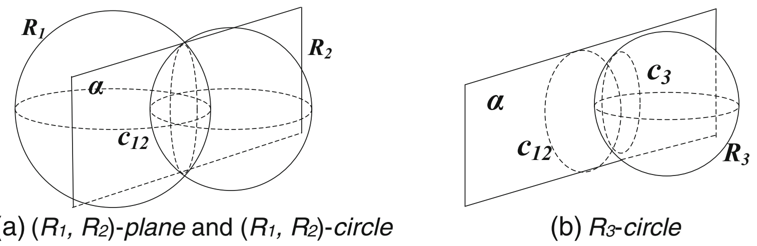


Fig. 7 Intersection arcs in three-dimensional space

As described in [17], there are different orderings of processing the three NLRs. In general, we should consider all possible orderings for any three NLRs. For example, one possible ordering of processing is that we first consider the intersection between R_2 and R_3 and then consider the intersection between the (R_2, R_3)*-plane* and the remaining NLR R_1.

5.2 Sweep plane

In the two-dimensional space, we use a sweep line to help us find out the intersecting NLRs. In a three-dimensional space, we can use a sweep plane to sweep the searching space downward along the *y*-axis as shown in Fig. 8. When a new NLR is inserted into the status structure of the sweep plane and used to search for intersecting NLRs, the search range become a rectangle now, as shown in Fig. 9. We will extend the lemma 4 to its three-dimensional space counterpart.

Lemma 9. Let *xleft* and *xright* be the *x*-coordinate of the left-most and right-most point of an NLR *R*, and *zfront* and *zback* be the *z*-coordinate of the front-most and back-most point of the NLR *R*, and *di* be the maximum diameter of NLRs swept up to now (as shown in Fig. 9). In the status structure, the NLRs whose left-most points are out of the range [$xleft-di$, $xright$] or front-most points are out of the range [$zfront+di$, $zback$] will not intersect with *R*.

The proof of this lemma is obvious, so we omit it here.

The SweepPlane algorithm is similar to the SweepLine algorithm except that, we use a kd-tree [11] instead of the balanced binary search tree as a status structure, so we omit it here.

5.3 Estimation

Similar to the two-dimensional space, we use a three-dimensional MBR (minimum bounding rectangle) to model an NLR and the result is also a kind of upper bound estimation. We divide an NLR into eight subspaces evenly by three planes, illustrated as NLR R_1 in Fig. 10a. Suppose that the center of R_1 is (x_0, y_0, z_0), then the three planes can be denoted as x=x_0, y=y_0 and z=z_0.

Fig. 8 Sweep plane in three-dimensional space

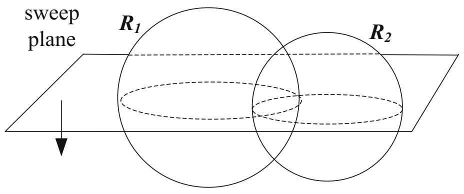
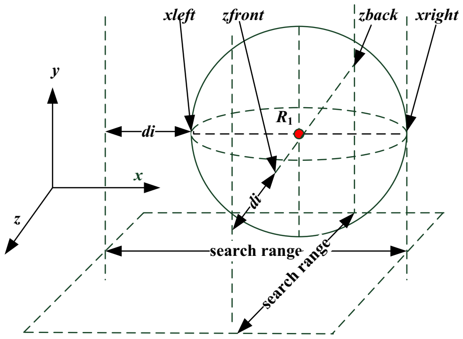


Fig. 9 Upper bound estimation in three-dimensional space



For each subspace of NLR R_1, we sum the weight of the NLRs whose MBRs intersect with the R_1's MBR in that subspace (including R_1 itself), noted as ws_1 to ws_8. Although the partitioning strategy with eight subspaces is described here, other partitioning strategies such as 12 (18, 24) partitioning can also be used. The 12 partitioning is illustrated in Fig. 10b. The selection of partitioning strategy is based on the balance between pre-computing cost and pruning effect. Our experimental result shows that when the 24 partitioning is utilized in a three-dimensional space, the pre-computing becomes very complicated and time consuming.

Lemma 5 and Lemma 6 in the two-dimensional space still hold in three-dimensional space.

5.4 Fine-grained pruning strategy

As described in Section 5.1, we can transform the MaxBRNN problem in a three-dimensional space into a two-dimensional arc search. We map the intersection points of intersection arcs to different angle values. Then, we scan all the intersection points to find the maximum influence value. This is the same as the OptRegion algorithm in a two-dimensional space. The 3D-OptRegion algorithm includes the following three major steps.

Step 1: Construct NLRs for a given dataset.

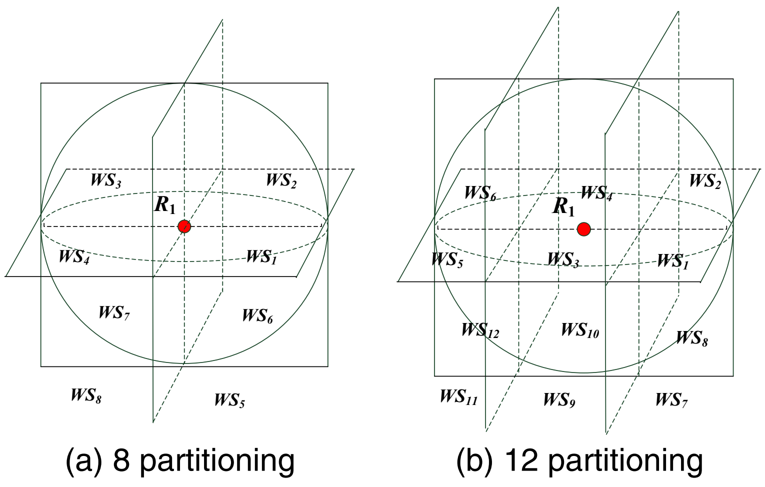


Fig. 10 Partition strategy in three-dimensional space

Step 2: Find out all the intersecting NLRs using SweepPlane algorithm, and construct an intersecting NLR list for each NLR. Estimate the upper bound *wsMax* of maximum influence value of the intersection points in all NLRs. Sort all the NLRs by the descending order of *wsMax* values.

Step 3: Process the NLRs by the descending order of the *wsMax* values. For each NLR *R*, we execute the following two sub-steps.

Sub-step 3.1: Find all the other NLRs intersecting with *R*.

Sub-step 3.2: For each NLR R_0 intersecting with *R*, we construct intersection arcs for all the NLRs intersecting with both R_0 and *R*, according to Section 5.1. We compute the influence values of the intersection points, and if the influence value is greater than the maximum influence value, we update the record of the maximum influence value and the corresponding intersection point.

The algorithm described above is a direct extension from a two-dimensional space to a three-dimensional space. However, it has some obvious drawbacks. Let's see an example.

Example 3. In Fig. 11, we have 6 spheres (NLRs). Their intersection relations are shown in Table 1, in which the intersecting subspaces are recorded, assuming the 8 partitioning in Fig. 10a to be used. The symbol ∅ represents that two spheres don't intersect with each other, and the symbol 'All' represents that the sphere intersects with itself. After executing SweepPlane algorithm and upper bound estimation, we get the sorted NLR list with the intersecting lists of each NLR, as shown in Table 2, the numbers outside the brackets denote

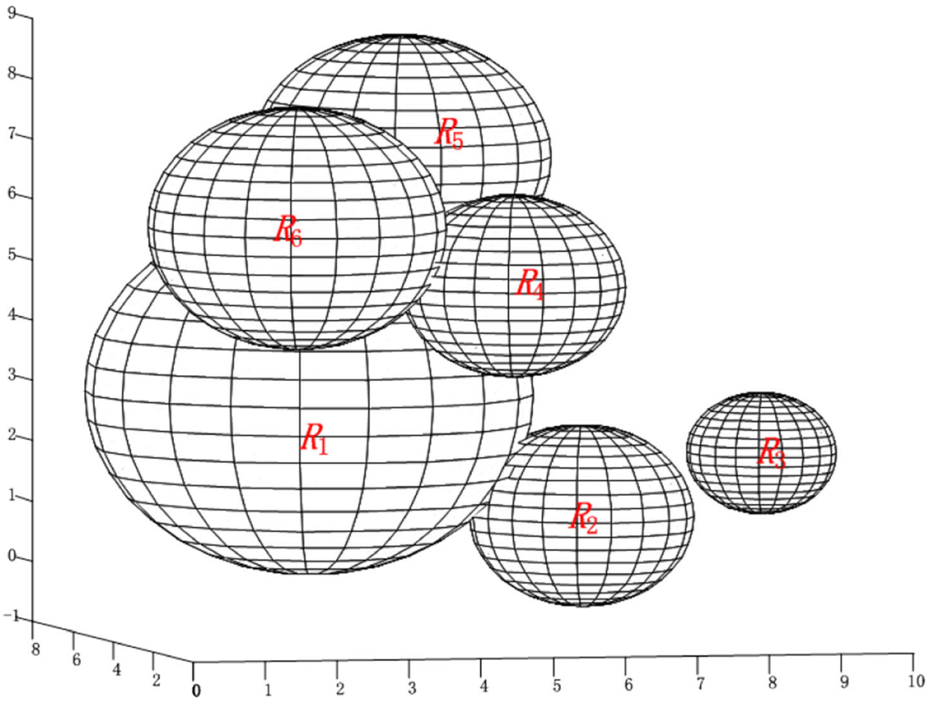


Fig. 11 An example in a three-dimensional space

Table 1 The intersection relations of spheres

Intersection relation	R_1	R_2	R_3	R_4	R_5	R_6
R_1	All	5, 6	Φ	1	1, 2, 3, 4	1, 4
R_2	3, 4	All	Φ	Φ	Φ	Φ
R_3	Φ	Φ	All	Φ	Φ	Φ
R_4	7, 8	Φ	Φ	All	3, 4	4
R_5	5, 6, 7, 8	Φ	Φ	5	All	4, 8
R_6	6, 7	Φ	Φ	6	2, 3, 6	All

the subspace's *ws* values and the numbers inside the brackets denote the corresponding subspace's ids.

According to the 3D-OptRegion algorithm, we first process R_1, and its corresponding sphere pairs: R_1-R_4, R_1-R_5, R_1-R_6, R_1-R_2. In this way, we have to process R_1-R_2 before sphere R_6 and R_4. But the R_1-R_2 pair is obviously the "useless pair", for it does not intersect with any other spheres. Whereas the R_6-R_4 pair in sphere R_6 and R_4 intersects with two other spheres, which means it is more likely to contain the optimal region. So, we should manage to avoid checking these "useless pairs".

The main drawback of the above method is that it should process all the intersecting pairs of an NLR according to the descending *wsMax* value order of NLRs. however, for a given NLR, it may has some dense intersection areas and some sparse areas. We should try to avoid processing these sparse areas.

We introduce a fine-grained pruning strategy to deal with the problem. The processing order of fine-grained pruning strategy is based on the fine grain of subspace of NLRs instead of whole NLR. we sort all the subspaces of all NLRs by the descending order of the weigh sum *ws* of subspaces (*ws* denotes weight sum, i.e. ws_i ($1 \le i \le 8$) if 8 partitioning is used for example, it is the upper bound of the maximum influence value of the intersection points in that subspace). In this way, we can process the NLRs' dense intersection areas at first and avoid processing the "useless pairs".

The proposed fine-grained pruning strategy works as follows. First, We compute the weigh sum *ws* of all subspaces and *wsMax* values of all NLRs, as shown in Table 3 for the example 3. The numbers outside the brackets are *ws* values of subspaces, and the numbers in the brackets denote the ID of NLRs that intersect in this subspace. Second, we sort the NLRs in the descending order of *wsMax*, which is the same as the OptRegion algorithm. Last, when processing the NLRs by descending order of *wsMax*, we utilize a priority-queue to record

Table 2 The NLR list in descending order of the *wsMax* values

NLR id	*wsMax*	Subspaces' *ws* value and *id*							
R_1	3	3(1)	2(4)	1(2)	1(3)	1(5)	1(6)	0(7)	0(8)
R_6	3	3(6)	1(2)	1(3)	1(7)	0(1)	0(4)	0(5)	0(8)
R_4	2	2(4)	1(3)	1(7)	1(8)	0(1)	0(2)	0(5)	0(6)
R_5	2	2(5)	2(8)	1(4)	1(6)	1(7)	0(1)	0(2)	0(3)
R_2	1	1(3)	1(4)	0(1)	0(2)	0(5)	0(6)	0(7)	0(8)
R_3	0	0(1)	0(2)	0(3)	0(4)	0(5)	0(6)	0(7)	0(8)

Table 3 The intersecting lists of NLRs

Subspace id NLR id	Sub1	Sub2	Sub3	Sub4	Sub5	Sub6	Sub7	Sub8
R_1	3(4,5,6)	1(5)	1(5)	2(5,6)	1(2)	1(2)	0	0
R_2	0	0	1(1)	1(1)	0	0	0	0
R_3	0	0	0	0	0	0	0	0
R_4	0	0	1(5)	2(5,6)	0	0	1(1)	1(1)
R_5	0	0	0	1(6)	2(1,4)	1(1)	1(1)	2(1,6)
R_6	0	1(5)	1(5)	0	0	3(1,4,5)	1(1)	0

all subspaces of processed NLRs up to now by descending *ws* values. The subspaces with higher *ws* values are to be processed first, to avoid processing the "useless pairs". In the fine-grained pruning strategy, we sort not only the NLRs, but also their subspaces.

The 3D-OptRegion algorithm is given in algorithm 2. We explain in detail the differences, compared with OptRegion. In line 5 and 6, a priority-queue structure *pQ*, which is used in the fine-grained pruning strategy, is constructed and initialized with the sphere with the largest *wsMax* value. The priority-queue *pQ* has two types of element: sphere and subspace. In line 7, a table *spherePair*, which is initially empty, is constructed to store all sphere pairs processed up to now. The table *spherePair* is used to avoid redundant processing of sphere pairs.

In line 9 and 10, while the priority-queue *pQ* is not empty, the head element, which is with the maximum *wsMax* or *ws* value in the priority-queue, is deleted and stored in variable *curr*. If *curr* is an NLR, we insert all its subspaces and its next NLR in the NLR list into the priority-queue *pQ*, in line 13-15. Only NLR and subspaces whose *wsMax* or *ws* values larger than *max* are inserted into the priority-queue *pQ*, which helps to prune some unpromising NLRs and subspaces. Otherwise, if *curr* is a subspace of an NLR, we process it according to the method described in Section 5.1, in line 16 to 24.

Algorithm 2 3D-OptRegion

```
input : O : = set of customer points
    P : = set of service points
output: S := optimal region presented by the overlapping NLRs
1 for each o∈O construct an NLR for o
2 call SweepPlane
3 compute the wsMax for all NLRs and the ws values of all subspaces
4 sort all the NLRs by the wsMax value
5 construct the priority-queue pQ
6 choose the NLR sphere R with the largest wsMax value, put it in pQ
7 construct the processed sphere pair table spherePair
8 s←any point in R, max←w(R)
9 while pQ is not empty
10 curr←delete the head element from pQ
11 if the wsMax or ws value of curr≤max
12 break // the algorithm terminates
13 if the type of curr is sphere
14 push next NLR of curr in NLR list into pQ if its wsMax value>max
15 push each subspace of curr into pQ if its ws value>max
```

```
16 else if the type of curr is subspace (assuming $R$ is the NLR of curr)
17 for each sphere $R_0$ in the subspace's intersecting list
18 if ($R$, $R_0$) pair hasn't been processed (search in spherePair)
19 add ($R$, $R_0$) to the table spherePair
20 find all spheres $u$ that intersects with both $R$ and $R_0$
21 if the weight sum of $R$, $R_0$, and all spheres $u$>max
22 compute ($R$, $R_0$)-plane and boundary circle $c_{12}$-circle
23 compute intersection circle of all spheres $u$ with ($R$, $R_0$)-plane, and their intersection points with $c_{12}$-circle
24 compute influence value for all intersection points in $c_{12}$-circle.
If the value is greater than max
update max, and replace $s$ by the corresponding intersection point, check pQ and prune all the
elements with wsMax or ws value<max
25 find the overlapping NLRs containing $s$, put the id of these NLRs into S
26 return S
```

We demonstrate the effectiveness and efficiency of the fine-grain pruning strategy in the following example.

Example 4. The setting is the same as Example 3. Each actions of algorithm, along with the contents of the priority-queue *pQ*, processed sphere pair table *spherePair*, and the *max* value are shown in Table 4.

First, the *spherePair* is empty, *max* is the weight of R_1, and R_1 is inserted into and deleted from *pQ*. Since R_1 is an NLR, we insert its subspaces, denoted as 1-1 to 1-8, and its next NLR R_6 in the NLR list into *pQ*.

Next, we delete 1-1 subspace from *pQ*. We check the intersecting list in Table 3, and process R_1 and R_4 at first. Since there is no record in *spherePair*, we insert (R_1, R_4) into it. We check the two sphere's intersecting list, and find out that R_5 and R_6 intersect with both of them. Because the weight sum of R_1, R_4, R_5, and R_6 is 4, greater than *max*, we execute line 22 to line 24 in 3D-OptRegion algorithm. After the processing of (R_1, R_4), the *max* is updated to 4. We check *pQ*

Table 4 Procedure of 3D-OptRegion

Actions	*pQ*	*spherePair*	*Max*	*S*
insert R_1	R_1	∅	1	$\{R_1\}$
delete R_1 and insert its subspaces and R_6	{1-1, R_6,1-4,1-2,1-3,1-5,1-6,1-7,1-8 }	∅	1	$\{R_1\}$
delete 1-1 and process R_1-R_4	{R_6,1-4,1-2,1-3,1-5,1-6,1-7,1-8 }	$\{R_1$-$R_4\}$	4	$\{R_1, R_4, R_5, R_6\}$
prune *pQ*	∅	$\{R_1$-$R_4\}$	4	$\{R_1, R_4, R_5, R_6\}$
process R_1-R_5, R_1-R_6	∅	$\{R_1$-R_4,R_1-R_5, R_1-$R_6\}$	4	$\{R_1, R_4, R_5, R_6\}$
return *S*	∅	$\{R_1$-R_4,R_1-R_5, R_1-$R_6\}$	4	$\{R_1, R_4, R_5, R_6\}$

and prune all the elements with *wsMax* or *ws* value$\le$*max*=4, resulting the *pQ* being $\varnothing$. Back to the intersecting list of subspace 1-1, we process sphere pair R_1-R_5, and R_1-R_6 in the same way.

Then, subspace 1-1 has been finished, and since *pQ* is $\varnothing$ now, the algorithm terminates and $\boldsymbol{S}$={ $\boldsymbol{R_1}$, $\boldsymbol{R_4}$, $\boldsymbol{R_5}$, $\boldsymbol{R_6}$ } is returned as the result.

6 Analysis

In this section, we will first analyze the time complexity of our proposed algorithms-OptRegion and 3D-OptRegion in sSection 6.1, and then in Section 6.2, we prove the correctness of our algorithm, at last, in Section 6.3, we give out the theoretical analysis and experiment verification of the accuracy of our upper bound estimation technique.

6.1 Time complexity

The OptRegion algorithm has the following three steps.

Step 1 (NLR construction): We use kd-tree [11] to perform nearest neighbor query. We build a kd-tree for all service points in P, which requires $O(|P|\log|P|)$ time and $O(|P|)$ space. Then we use the algorithm ANN in [1] to find the nearest service point over the kd-tree, which requires $O(\log|P|)$ time for each customer point in O. Thus, the construction of NLRs can be done in $O(|P|\log|P|+|O|\log|P|)$.

Step 2 (Find intersecting NLRs and estimate upper bound of influence value): Let d_1 be the maximum number of NLRs whose MBR intersects with a given NLR's MBR, and c be the maximum NLRs intersecting with a sweep line. In the sweep line algorithm, it takes $O(|O|\log|O|)$ time to sort the event queue. For each NLR, it takes $O(d_1+\log c)$ time to test intersection, and $O(\log c)$ time to insert into and delete from the status structure. So, the overall execution time for sweep line algorithm is $O(|O|(d_1+\log|O|))$. For each NLR, its MBR intersects with at most d_1 MBRs, thus it takes $O(d_1)$ time to process each NLR, i.e., $O(|O|d_1)$ time to estimate upper bound of influence value for all NLRs. Thus, this step requires $O(|O|(d_1+\log|O|))$ time.

Step 3 (Find the optimal intersection point): first we need to sort all NLRs by descending *wsMax* order, which re-quires $O(|O|\log|O|)$ time. Actually, we need not fully sort all NLRs, here we only need a priority-queue that can ret-urn the NLR with next maximum *wsMax* value. As soon as the *wsMax* value returned is below the maximum influe-nce value found so far, remained NLRs are all pruned without further processing. Let α_1 $(0\le\alpha_1<1)$ be the prune rate.

There are at most d_1 intersection arcs in each NLR, the influence value computation takes $O(d_1)$ time for each NLR.

Thus, this step requires $O((1-\alpha_1)|O|d_1)$ time.

From the analysis above, we can get the following theorem.

Theorem 1 The overall time complexity of OptRegion in a two-dimensional space is $O(|P|\log|P|+|O|(\log|P|+d_1+\log|O|))$.

As described in Section 5.4, the 3D-OptRegion algorithm comprises the same three steps as the OptRegion algorithm. The time required in step 1 is the same as OptRegion, which takes $\boldsymbol{O}(|P|\log|P|+|O|\log|P|)$.

For step 2, Let d_1 be the maximum number of NLRs whose MBR intersects with a given NLR's MBR, and c be the maximum NLRs intersecting with a sweep plane. In the sweep plane algorithm, it also takes $O(|O|\log|O|)$ time to sort the event queue. For each NLR, it takes $O(d_1+\log c)$ time to test intersection(since the point query in a kd-tree takes $O(\log c)$ time), and $O(\log c)$ time to insert into and delete from the status structure(a kd-tree). So, the overall execution time for sweep plane algorithm is also $O(|O|(d_1+\log|O|))$. For each NLR, its MBR intersects with at most d_1 MBRs, thus it takes $O(d_1)$ time to process each NLR, i.e., $O(|O|d_1)$ time to estimate upper bound of influence value for all NLRs. Thus, this step requires $O(|O|(d_1+\log|O|))$ time.

For step 3, let d_2 be the maximum number of NLRs whose MBR intersects with two given NLRs' MBR (it is obvious $d_2 \le d_1$), and α_2 ($0 \le \alpha_2 < 1$) be the prune rate of fine-grained pruning strategy. In a three dimensional space, it is easy to know that given two NLRs R and R_0, the computation of the (R,R_0)-plane and (R,R_0)-circle takes $O(1)$ time. It takes $O(d_1 d_2)$ time to process an NLR, So, the time required in step 3 is $O((1-\alpha_2)|O|d_1 d_2)$. Thus, we can get the following theorem.

Theorem 2 The overall time complexity of 3D-OptRegion is $O(|P|\log|P|+|O|(\log|P|+\log|O|+d_1 d_2))$.

6.2 Correctness demonstration

The correctness of OptRegion and 3D-OptRegion is guaranteed by the following five aspects:

1) Based on Lemma 3, 4, the sweep line algorithm can find all the intersecting pairs of NLRs in a two-dimensional space. Based on Lemma 9 and the demonstration in Section 5.2, the sweep plane algorithm can find all the intersecting spheres in a three-dimensional space.
2) Based on Lemma 5 ,6, and the analysis in Section 4.2, the space dividing pruning strategy can prune the NLRs correctly and the upper bound estimation can derive a tight upper bound. According to the demonstration in Section 5.3, the 8-partition and 12-partition strategies can pruning the searching space efficiently in three-dimensional space, and Example 3 in Section 5.4 demonstrate the correctness and effectiveness of our Fine-grained Pruning Strategy (FPfor short). In Section 6.3 we will demonstrate the accuracy of Upper Bound Estimation in a more formal way.
3) According to Lemma 7,8 and Example 4 in Section 5.4, the influence value computation process can compute th-e influence value of intersection points correctly in two and three-dimensional spaces.
4) Based on properties in [17], the transformation of three NLRs' intersection in a three-dimensional space into tw-o circles' intersection in a two-dimensional space is correct.
5) According to Algorithm 1 in Section 4.4 and Algorithm 2 in Section 5.4, OptRegion and 3D-OptRegion return the MaxBRNN region correctly in two-and three-dimensional spaces respectively.

Above all, the correctness of OptRegion and 3D-OptRegion can be concluded as theorem 3. The detailed proof can be derived from the above description.

Theorem 3 OptRegion and 3D-OptRegion return the MaxBRNN region correctly in two- and three-dimensional spa-ces respectively.

6.3 Accuracy of upper bound estimation

We utilize some approximations in the upper bound estimation in algorithm OptRegion. We approximate each NLR by its MBR to test intersection with each other. Now we will present the accuracy analysis about the approximation strategies. To be brevity and without loss of generality, we only analyze the accuracy for 4-partition strategy in a two-dimensional space.

Lemma 10. Suppose the dataset is uniform distributed. Given two NLRs *A* and *B*, if B's MBR intersects with one subspace of A's MBR (assuming 4-partitioning is used), then the probability of B really intersecting with this subspace of A is $\pi(r1+r2)\,\frac{{}^{2}+8r1*r2+3*\pi r_2^2}{4(r1+2r2)}\,2$, here r_1 and r_2 are radius of *A* and *B* respectively.

Proof. Suppose *A, B* are the center of the NLRs, and B's MBR intersects with the MBR of subspace 1 of A without loss of generality. We can imagine that *B*'s MBR moves along the MBR of the subspace 1 of *A*, so the trajectory of point *B* constitutes the shaded square, as shown in Fig. 12a. For every NLR with radius r_2 and its center inside the shaded square, its MBR intersects with the MBR of subspace 1 of A. The area of this shaded square is $(r_1+2r_2)^2$.

On the other hand, by moving NLR *B* around the subspace 1 of *A*, the trajectory of point *B* constitutes another shaded region, as shown in Fig. 12b. For every NLR with radius r_2 and its center inside this shaded area, it intersects with the subspace 1 of A. we can see that this area includes a quarter of a circle with radius r_1+r_2, two rectangles with the same length r_1 and width r_2, and three small quarters of circles with radius r_2. Thus, the area of this region is

$$\frac{\pi(r1+r2)^2}{4}+2r1*r2+3*\frac{\pi r_2^2}{4}$$

The probability of really intersecting is the ratio of the two shaded region.

$$\mathrm{P}=\frac{\pi(r1+r2)^2+8r1*r2+3*\pi r_2^2}{4(r1+2r1)^2} \tag{1}$$

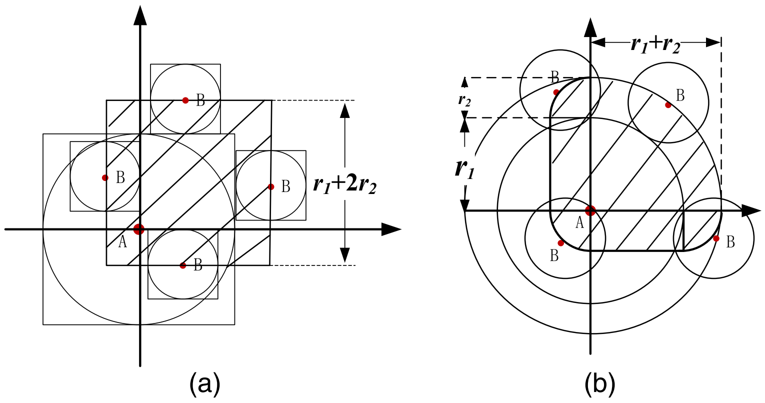


Fig. 12 Accuracy analysis of MBR approximation

We use variable a to denote the ratio of r_1 and r_2 (i.e., r_1/r_2), then we can rewrite Eq. (1) to

$$\mathrm{P}=\frac{\pi}{4}+\frac{2-\frac{\pi}{2}}{a+\frac{4}{a}+4} \tag{2}$$

For Eq. (2), when a ranges from (0, +∞), P ranges in (π/4, 3π/16+0.25]. If a=1, the value of P is about 83.3 %.

We conduct experiment on the uniform distributed dataset to confirm Lemma 10. For each NLR, says, R_i , suppose N_{ij} is the number of NLRs really intersect with R_i's jth subspace and ws_{ij} is the upperbound of this subspace, so, the subspace estimation precision rate of R_i can be expressed as follows:

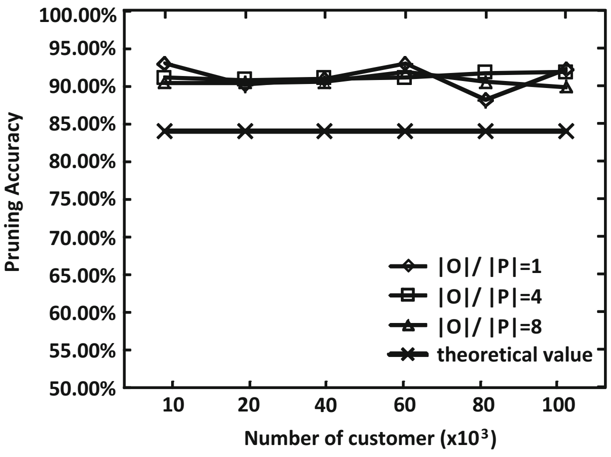
$$\mathrm{P_i}=\frac{\sum_{j=1}^{4}\frac{N_{ij}}{ws_{ij}}}{4} \tag{3}$$

And consequently, we derive the subspace estimation precision rate as follows(N is the number of NLRs),

$$\mathrm{P}=\frac{\sum_{i=1}^{N}P_i}{N} \tag{4}$$

The experiment result is as shown in Fig. 13, the top three lines are the values of P computed as shown in Eq. (4) with different |O| / |P|, the x axis denotes the value of N. The experimental values of P changes around 92 %, this is a little higher than the theoretical range about (0.79, 0.84]. This maybe because that the theoretical analysis suppose that the data points are generated totally randomly and they are distributed densely in the whole plane, and the number of the data points is infinite, however in the experiments, the NLRs may have some relationships with the customer and service points, and the cardinality of our experimental data set is limited to 10^6. The experiments suggest that our upper bound estimation can have greater

Fig. 13 Experimental result of estimation accuracy



accuracy than the theoretical values, and it further proves that our pruning strategy is reasonable and practical.

7 Performance study

In this section, we will first introduce the experimental environment and settings, and our evaluation criterion. And then in Section 7.2 and we give out the experimental results in two-dimensional space. Section 7.3 and 7.4 are the extension to the preliminary conference version [16], we evaluate the effect of parameter k in MaxBR***k***NN problem and evaluate our 3D-OptRegion algorithm in three-dimensional space.

7.1 Experimental settings

We have conducted extensive experiments to evaluate the performance of our algorithm OptRegion and 3D-OptRegion. We compare our algorithm with the state-of-the-art algorithm MaxOverlap[26], MaxSegment[17], and MaxFirst[29] for the MaxBRNN problem. All experiments show that our algorithm outperforms other algorithms significantly.

The algorithms are implemented in C++, in which we reused part of the C++ code of MaxOverlap. The C++ code of MaxOverlap is got from the authors. We implement the algorithm MaxSegment and MaxFirst in C++ according to the description in their papers. All experiments are carried out on a Linux machine with an Intel Core2 Duo 2.9 GHz CPU and 2GB memory.

The performance evaluation is performed using both synthetic and real datasets. The synthetic datasets follow Uniform, Gaussian and Zipf distributions, and the number of customer points ranges from 50 to 200K. The customer dataset and the service dataset have the same distribution. Two real world two-dimension datasets LB and CA, which contain 2D points representing geometric locations in Long Beach Country and California respectively, are used in the experiments for OptRegion. The LB and CA datasets are available at http://www.rtreeportal.org/spatial.html. For real data sets, the number of customer point is in the range from 10 to 40K. We partition the real datasets into two parts, the first 40K points are customers and the remaining 20K points are services. We set the cardinality of P (service set) to be half the cardinality of O (customer set) for both synthetic and real datasets, since in reality the number of services is always much fewer than that of customers, we can also use other ratios, too, e.g. |P|/|O|<1/2 which will not affect the overall performance. The statistics about datasets are summarized in Table 5.

The weight of each customer point in both synthetic and real datasets is set to 1. When the weights of customer points are other than 1, the experimental results are similar and we omit it here. The k value of a MaxBRkNN query varies from 1 to 15, and its default value is 1.

Table 5 Datasets

Dataset	Cardinality	Dataset	Default value	Range
(a) Real datasets		(b) Synthetic datasets		
CA	62,556	| O |	50K	10~200K
LB	53,145	| P |	| O |/2	5~100K

We first conduct experiments to evaluate the pruning strategy with different partitioning methods in a two-dimensional space in Section 7.2.1. In Section 7.2.2, to gain insight of the performance promotion of our algorithm, we compare algorithms from two aspects of running time: 1) overall running time, which includes the execution time for all three steps in Section 6.1; 2) running time without preprocessing, which includes the time of finding intersecting NLRs, estimating upper bound of influence value, and finding the optimal intersection point, i.e. the step 2 and step3 in Section 6.1. In Section 7.3, we evaluate the effect of parameter k from two aspects: 1) the running time of different algorithms with different k values; 2) the pruning power of algorithm OptRegion with different k values. In Section 7.4, we evaluate the performance of 3D-OptRegion in a three-dimensional space. First we demonstrate the effectiveness of the fine-grained pruning strategy, and then we compare OptRegion with the MaxSegment algorithm from the aspect of running time.

7.2 Experiment results in two-dimensional space

7.2.1 Pruning strategies in two-dimensional space

The upper bound estimation of influence value in OptRegion is based on the space partitioning of an NLR. Different partitioning will have different pruning effect. We explore three strategies here, i.e. 1-partition, 4-partition and 8-partition, in which an NLR is partitioned into 1, 4, and 8 subspaces respectively. We depict the pruning effect and running time of different strategies in Fig. 14 and Table 6 respectively, which show that 4 and 8-partition are superior to 1-partition. The experiments are conducted on the synthetic dataset with 80K customer points. Compared to 1-partition, both 4 and 8-partition have more powerful pruning effect. We have to compute exact influence value of more than 1300 NLRs against all 80K NLRs when 1-partition is used, whereas only 253 and 140 NLRs really need exact influence value computation when 4 and 8-partition are used respectively. The prune rate α is 99.83 % for 8-partition.

From Table 6 we can find that, as the increase of subspace number, the time percentage of exact influence value computation decreases, but the time percentage of upper bound estimation increases. There are almost no differences between the running time of 4 and 8-partition. Although 8-partition strategy can prune more NLRs, the pruning procedure needs more computation. Except the 1-partition strategy, we can choose either of the other two pruning strategies.

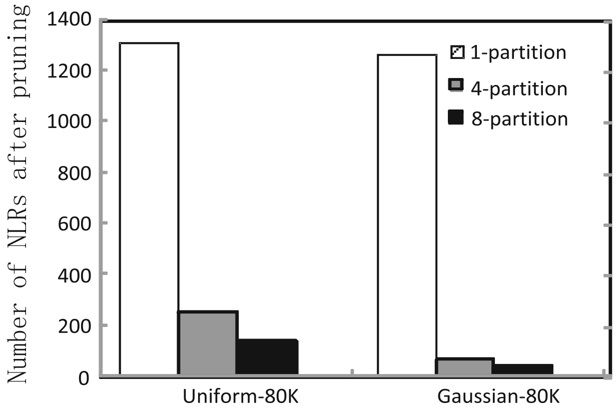


Fig. 14 Pruning effect of different strategies

Table 6 Running time of different pruning strategies

		1-partition	4-partition	8-partition
Uniform 80K	number of NLRs not pruned	1302	253	140
	time percentage of upper bound estimation	70.04 %	78.20 %	82.76 %
	time percentage of influence compute	16.30 %	4.71 %	1.31 %
	total time (without preprocessing)	1.67s	1.38s	1.45s
Gaussian 80K	number of NLRs not pruned	1261	69	40
	time percentage of upper bound estimation	76.60 %	82.10 %	82.77 %
	time percentage of influence compute	7.90 %	0.45 %	0.38 %
	total time (without preprocessing)	1.58s	1.3s	1.29s

7.2.2 Performance in two-dimensional case

Overall running time The experiment results are given in Fig. 15 (which is a reduced version of the conference version in [16]) for synthetic and real datasets with different point set cardinalities. Our algorithm OptRegion outperforms MaxOverlap, MaxSegment, and MaxFirst significantly for all circumstances. The performance promotion is up to about one order of magnitude. As the number of point set increases, the overall running time of the other algorithms increase dramatically, whereas OptRegion has a relativelymuch smaller increase rate, showing its good scalability. The reason of the remarkable performance improvement can be found in [16].

Running time without preprocessing The experiment results are given in Fig. 16. Without the preprocessing time, OptRegion is about one time faster than MaxFirst and several times faster than MaxOverlap and MaxSegment. Comparing Fig. 16a and b, we observe that data distribution affects the algorithms performance. All algorithms spend more time on datasets with Gaussian distribution than uniform distribution, for there are much more intersecting NLRs and intersection points in the dense area of Gaussian distribution dataset.

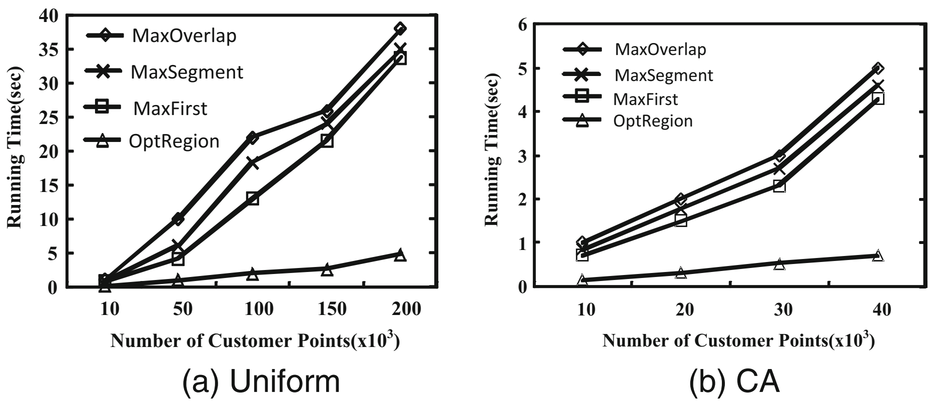


Fig. 15 Overall running time

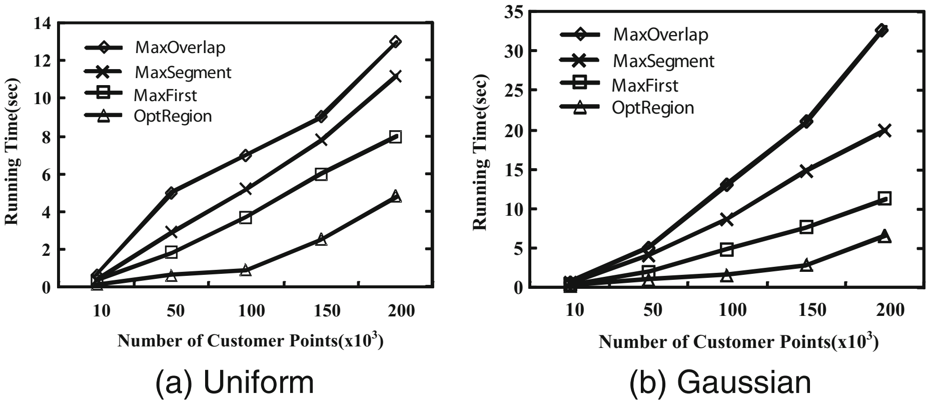


Fig. 16 Running time without preprocessing

7.3 Effect of k

7.3.1 Overall running time comparison

We first evaluate the effect of *k* by the comparison of running time (including preprocessing time) with the state-of-art algorithms. As shown in Fig. 17a–d, no matter the synthetic data sets(Uniform and Gaussian) or the real data sets (CA and LB), the execution times of all

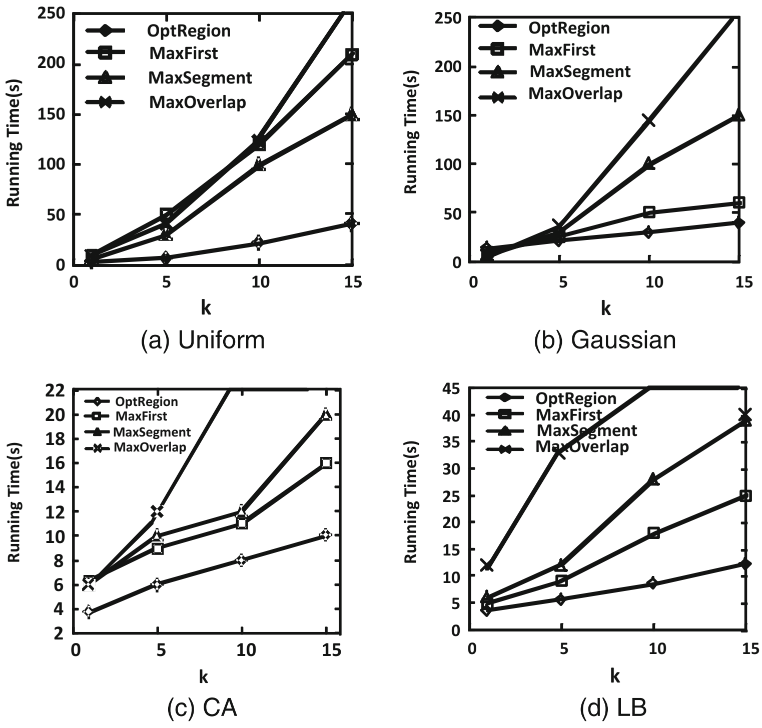


Fig. 17 Effect of *k*

algorithms increase with *k*. This is because as *k* increases, the radius of NLRs increases and it is more likely that an NLR intersects with another NLR, which makes the influence value larger. The experiment shows that the increase in the execution time of OptRegion is smaller than that of the state-of-art algorithms as *k* increases, which means our algorithm OptRegion has much better scalability with respect to *k*.

7.3.2 Result of pruning strategies

We conduct experiments to demonstrate the effect of *k* on different pruning strategies in our algorithm. As shown in Fig. 18, as *k* increases, the number of NLRs that need to be processed after pruning increase in all pruning strategies. This is because as *k* increases, it is more likely that an NLR intersects with another NLR, so the number of intersection dense areas maybe increase, resulting in more NLRs to be processed. From Fig. 18, it can be found that the pruning power of both 4-partition and 8-partition are still significantly better than the 1-partition strategy (in the 8-partition strategy, the NLRs' number is too small to see in the figure when *k*=1).

It may be found that when *k* is larger than 5, the power of 8-partition strategy degrades dramatically. So, when *k* is larger than 5, we'd better choose the 4-partition strategy.

7.4 Performance in three-dimensional case

7.4.1 Effectiveness of fine-grained pruning strategy

We evaluate the Effectiveness of fine-grained pruning strategy by comparing the number of spheres (NLSs) and sphere-pairs processed using the uniform dataset (the results are similar using other datasets and we omit it here), both algorithms adopt the 8-partition estimation strategy. From Fig. 19a, the number of spheres that have to be processed in algorithm 3D-OptRegion with FP (short for fine-grained pruning strategy) is almost half the number in algorithm 3D-OptRegion without FP. Figure 19b shows that the sphere pair number processed in algorithm 3D-OptRegion with FP is about one order less than the number in algorithm 3D-OptRegion without FP. The performance gain of fine-grained pruning strategy mainly comes from the fact that we sort the NLRs' subspace instead of solely NLRs, and we can process the NLRs' dense area first and avoid processing the useless sphere pair.

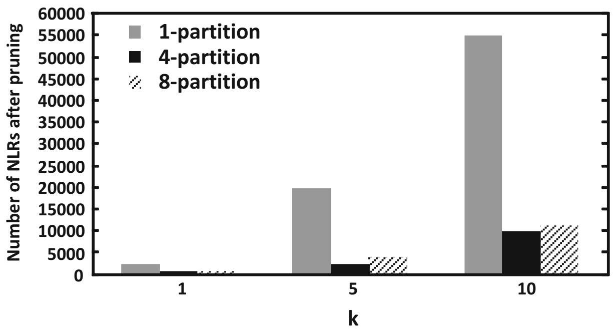


Fig. 18 Effect of *k* in different pruning strategies

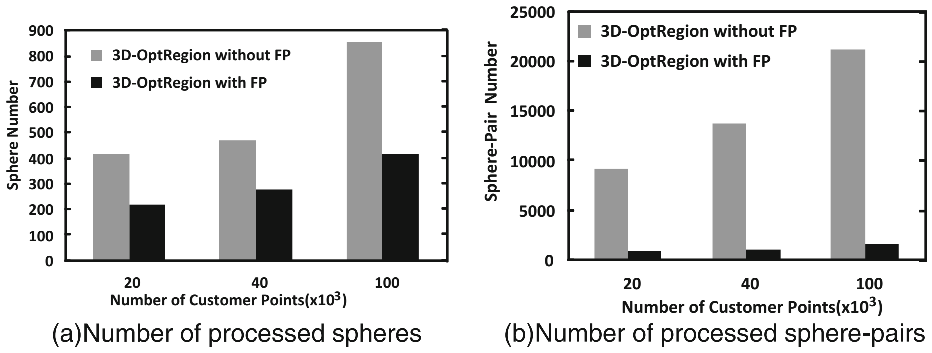


Fig. 19 Effectiveness of fine-grained pruning strategy(FP)

7.4.2 Running time comparison

We compare our algorithm 3D-OptRegion with algorithm MaxSegment, which is the most efficient algorithm for MaxBRNN problem in the three-dimensional space, using the uniform and Gaussian dataset (the results are similar using other datasets and we omit it here). Fig. 20a and b show the comparison of overall running time. Since the two algorithms utilize the same preprocess method, we also conduct experiment for comparison of running time without the preprocessing, as shown in Fig. 21a and b.

It is obvious that algorithm 3D-OptRegion outperforms algorithm MaxSegment constantly under all dataset instances. As shown in Fig. 21a and b, the running time without preprocessing of algorithm 3D-OptRegion is le-ss than half of the time of algorithm MaxSegment. There are two major advantages in 3D-OptRegion that contribute to the remarkable performance improvement:

1) We adopt sweep plane technique to find intersecting NLRs, which localizes the search range, only needs much more simple computation, and avoids the construction of the R-tree of NLRs, which is very time consuming;
2) We adopt a fine-grained pruning strategy, which is much efficient.

7.4.3 Effect of data distribution

To test the best/worst case for our proposed algorithm 3D-OptRegion(we only consider the 3D-OptRegion with FP since it performs better than without FP), we use

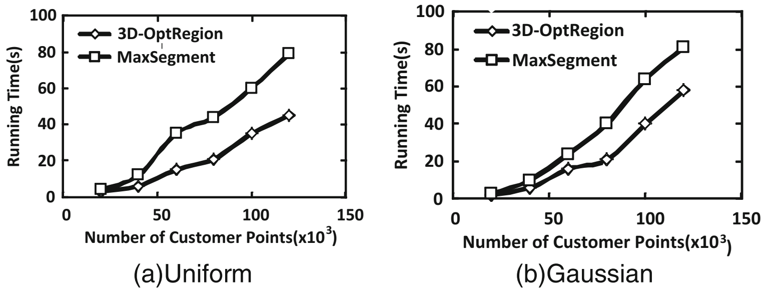


Fig. 20 Overall running time in three-dimensional-space

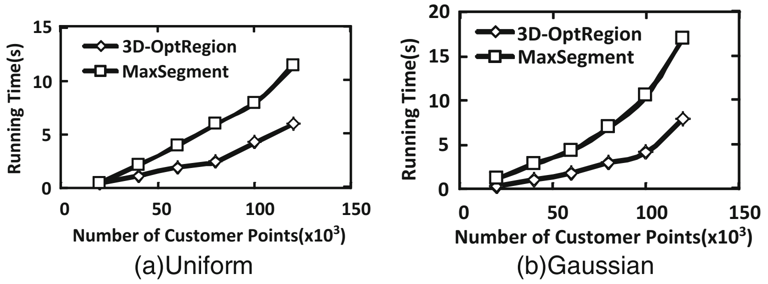


Fig. 21 Running time without preprocessing in three-dimensional-space

Zipf distribution datasets with different factor values varying from 0.2 to 5, which indicate different data distributions. We test both the running time and pruning power of our algorithm on different data sets. The data sets are described in Fig. 22, with the increasing of the factor value, the dataset skews more dramatically, thus, using the different Zipf datasets, we can tell the best and worst cases of our proposed algorithm. From Fig. 22f, we can see that the data distribution becomes irregular when factor value is larger than 5, so we only test factor value from 0.2 to 5.

Figure 23 shows the running time on different data distributions, we can see that algorithm 3D-OptRegion has the best performance when factor value varies from 0.2 to 0.8 or factor value is bigger than 4. And when factor value varies from 1 to 3 the algorithm performs worst. The reason of this efficiency change can be explained by Fig. 24.

The critical factor that impacts the algorithm's efficiency is the number of sphere-pairs and subRegions being processed, and Fig. 24a, b illustrate the effect of data distribution from these two aspects. From Fig. 24a, we can see that as factor value varies from 0.2 to 1, the number of sphere-pairs being processed decreases while the number of subRegions being processed

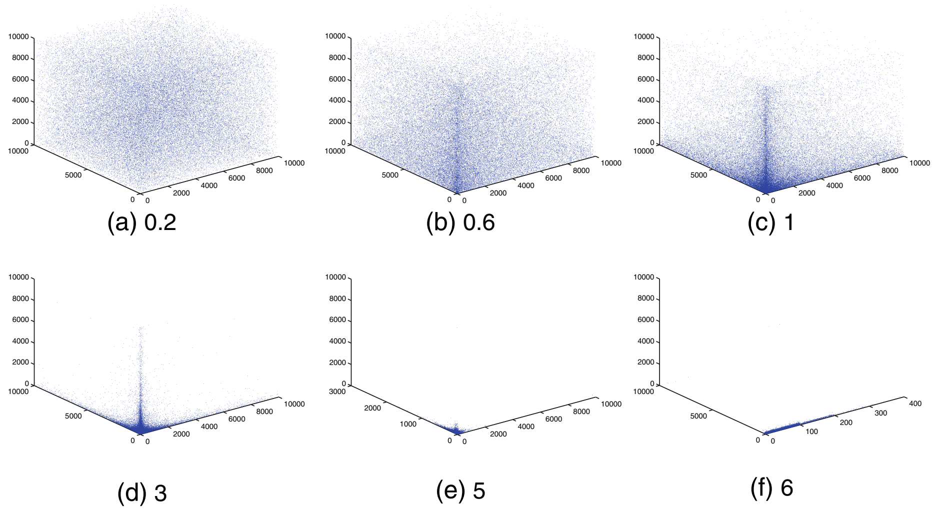


Fig. 22 Data distribution (Zipf distribution with different factor value)

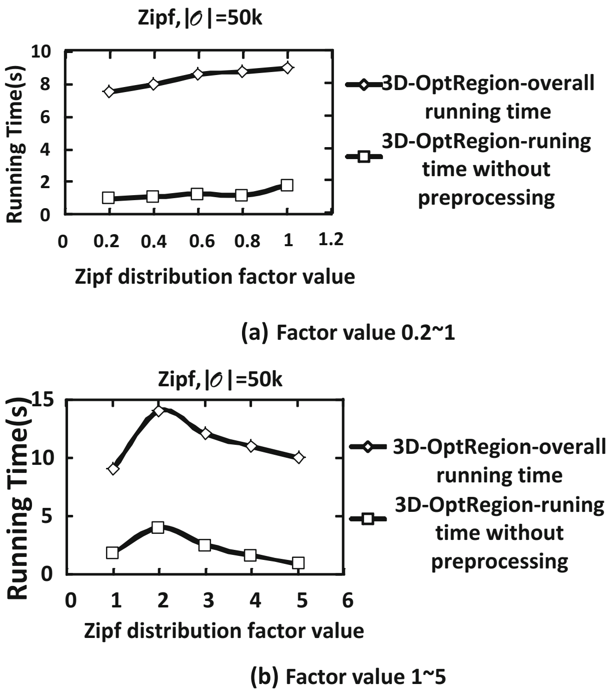


Fig. 23 Running time on different data distributions

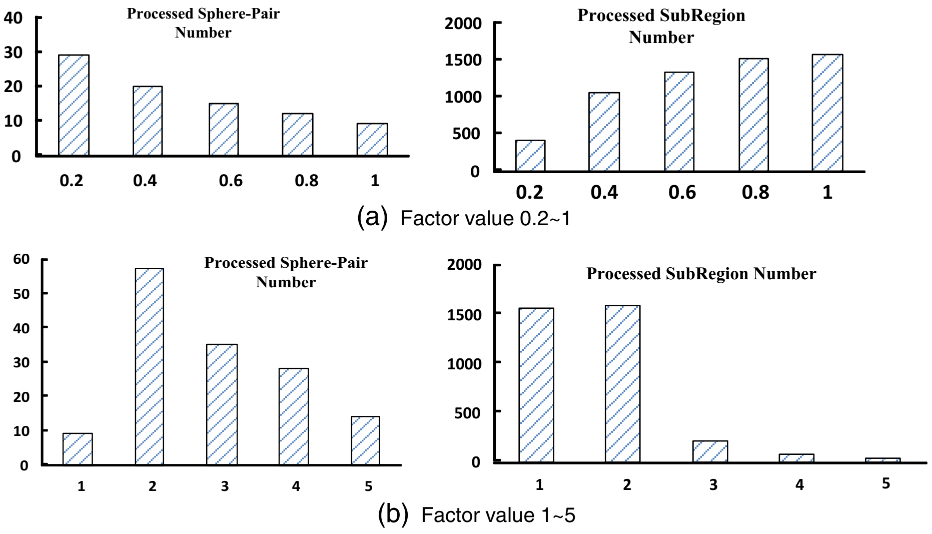


Fig. 24 Pruning effect on different data distributions

increases, as a result of this, the running time remains stable as shown in Fig. 23a. When factor value varies from 2 to 5, both of the sphere pair and subRegion's number decreases (as shown in Fig. 24b), consequently, the running time decreases (Fig. 23b).

In general, the 3D-OptRegion algorithm performs better when the dataset is uniform or skews dramatically.

8 Conclusion and future work

In this paper, we propose two efficient algorithms, namely, OptRegion, and 3D-OptRegion to tackle the MaxBRNN, Max-***3D***-BRNN and MaxBR*k*NN problems, which have many applications in real life. The algorithms employ several efficient techniques, such as sweep line (plane), upper bound estimation, and fast influence value computation. Extensive experiments using both real and synthetic data sets verify the effectiveness and efficiency of our proposed algorithms. The experimental results show that OptRegion and 3D-OptRegion outperform the state-of-the-art algorithms significantly in two- and three-dimensional spaces.

There are several directions in our future work. First, the current MaxBRNN problem only returns one optimal region. However, a decision maker may want to choose a group of optimal regions to set up new service facilities. It is necessary to consider the problem under the combinatorial context. Second, we will explore the MaxBRNN problem in the road network.

Acknowledgments This research was partly supported by The National Science and Technology Supporting Plan-Project no. 2012BAH70F02, 2013BAH62F01 and 2013BAH62F02.

References

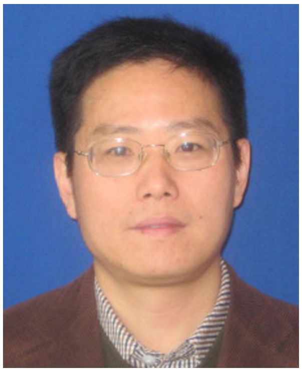
1. Arya S, Mount D-M, Netanyahu N-S, Silverman R, Wu A-Y (1998) An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. JACM 45(6):891–923
2. Berg M, Cheong O, Kreveld M, Overmars M (2009) Computational geometry: algorithms and applications. [M]. 3rd edn. Springer, p 32–55
3. Bernecker T, Emrich T, Kriegel H-P, Renz M, Zankl S, Züfle A (2011) Efficient probabilistic reverse nearest neighbor query processing on uncertain data. VLDB:669–680
4. Cabello S, Díaz-Báñez JM, Langerman S, Seara C, Ventura I (2005) Reverse facility location problems. CCCG
5. Cabello S, Díaz-Báñez JM, Langerman S, Seara C, Ventura I (2010) Facility location problems in the plane based on reverse nearest neighbor queries. Eur J Oper Res 202(1):99–106
6. Cheema M-A, Zhang W, Lin X, Zhang Y, Li X (2012) Continuous reverse k nearest neighbors queries in Euclidean space and in spatial networks. VLDB 21(1):69–95
7. Chen Z, Wang L, Liu W (2012) Method for maximizing bichromatic reverse nearest neighbor in road networks. J Converg Inf Technol 7(4):125–133
8. Chen F, Lin H, Gao Y, Lu D (2015) Capacity constrained maximizing bichromatic reverse nearest neighbor search. Expert Syst Appl. doi:10.1016/j.eswa.2015.08.051
9. Choi D-W, Chung C-W, Tao Y (2012) A scalable algorithm for maximizing range sum in spatial databases. VLDB 5:1088–1099
10. Du Y, Zhang D, Xia T (2005) The optimal-location query. SSTD:163–180
11. Friedman J-H, Bentley J-L, Finkel R-A (1977) An algorithm for finding best matches in logarithmic expected time. ACM TOMS 3:209–226

12. Gao Y, Zheng B, Chen G, Li Q, Guo X (2011) Continuous visible nearest neighbor query processing in spatial databases. VLDB 20(3):371–396
13. Ghaemi P, Shahabi K, Wilson J-P, Banaei-Kashani F (2014) A comparative study of two approaches for supporting optimal network location queries. GeoInformatica 18(2):229–251
14. Kang J-M, Mokbel MF, Shekhar S, Xia T, Zhang D (2007) Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. ICDE:781–790
15. Korn F, Ukrishnan S-M (2000) Influence sets based on reverse nearest neighbor queries. SIGMOD:201–212
16. Lin H, Chen F, Gao Y, Lu D (2013) OptRegion: finding optimal region for bichromatic reverse nearest neighbors. DASFAA
17. Liu Y, Wong R-C, Wang K, Li Z-J, Chen C (2012) A new approach for maximizing bichromatic reverse nearest neighbor search. KAIS
18. Nghiem T-P, Maulana K, Nguyen K, Green D et al (2014) Peer-to-peer bichromatic reverse nearest neighbours in mobile ad-hoc networks. J Parallel Distrib Comput 74(11):3128–3140
19. Sedgewick R, Brown MH (1983) Data structures and algorithems. [M]. 1st edn. Addison-Wesley, Balanced Trees
20. Shang S, Yuan B, Deng K, Xie K, Zhou X (2011) Find the most accessible locations: reverse path nearest neighbor query in road networks. ACM GIS:181–190
21. Singh A, Ferhatosmanoglu H, Tosun A (2003) High dimensional reverse nearest neighbor queries. CIKM: 91-98
22. Stanoi I, Agrawald D (2000) Reverse nearest neighbor queries for dynamic databases. ACM SIGMOD DMKD:44–53
23. Tao Y, Papadias D, Lian X (2004) Reverse KNN search in arbitrary dimensionality. VLDB:744-755
24. TaoY, Hu X, Choi D-W, Chung C-W (2013) Approximate MaxRS in spatial databases. PVLDB:1546–1557
25. Tran Q, Taniar D, Safar M (2010) Bichromatic reverse nearest-neighbor search in mobile systems. IEEE Syst J 4(2):230–242
26. Wong R-C, Özsu MT, Yu P-S, Fu A-W, Liu L (2009) Efficient method for maximizing bichromatic reverse nearest neighbor. VLDB:1126–1137
27. Wong R-C, Özsu MT, Fu A-W, Yu P-S, Liu L, Liu Y (2011) Maximizing bichromatic reverse nearest neighbor for *Lp*-norm in two- and three-dimensional space. VLDB 20:893–919
28. Yan D, Zhao Z, Ng W (2012) Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces. CIKM:942–951
29. Zhou Z, Wu W, Li X, Lee M-L, Hsu W (2011) MaxFirst for MaxBRkNN. ICDE:828–839

Huaizhong Lin is currently an associate professor of computer science in Zhejiang University. He received a Ph.D. degree in Computer Science from Zhejiang University in 2002. His research interests are database and data mining, spatial database, information retrieval etc., he has published over 40 research papers in journals and conferences, and has been granted 6 patents.

Fangshu Chen received the BS degree in computer science from DaLian Unversity of Technology, China, in 2011. She is currently working toward the PHD degree in the College of Computer Science, Zhejiang University, China. Her research fields mainly focus on spatial databases.

Yunjun Gao received the PhD degree in computer science from Zhejiang University, China, in 2008. He is currently an associate professor in the College of Computer Science, Zhejiang University, China. Prior to joining the faculty, he was a Postdoctoral Fellow at the Singapore Management University during 2008–2010, and a Visiting Scholar or Research Assistant at the Nanyang Technological University, Simon Fraser University, and City University of Hong Kong, respectively. His research interests include spatial and spatiotemporal databases, uncertain and incomplete databases, and spatio-textual data management. He has published papers in Journals and conferences including TODS, VLDBJ, TKDE, SIGMOD, ICDE, and SIGIR. He is a member of the ACM and the IEEE, and a senior



Dongming Lu His research fields mainly focus on the digital media network technology and system, Heritage digital protection and culture passing technology, Wireless and next generation Internet technology and Network information security technology. He has publised more than 60 articles and 11 invention patents in the past five years. His monograph "digital cultural relics protection and development of technology" was published by Zhejiang University Press and Springer. He is currently the executive director of the Computer Society, Computer Society of network technical, director and members of the special committee of the China Digital Museum of Zhejiang Province.