Influence Zone and Its Applications in Reverse k Nearest Neighbors Processing

Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, Ying Zhang

The University of New South Wales, Australia

{macheema,lxue,zhangw,yingz}@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-1017
Jun 2010

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Given a set of objects and a query q, a point p is called the reverse k nearest neighbor (RkNN) of q if q is one of the k closest objects of p. In this paper, we introduce the concept of influence zone which is the area such that every point inside this area is the RkNN of q and every point outside this area is not the RkNN. The influence zone has several applications in location based services, marketing and decision support systems. It can also be used to efficiently process RkNN queries. First, we present efficient algorithm to compute the influence zone. Then, based on the influence zone, we present efficient algorithms to process RkNN queries that significantly outperform existing best known techniques for both the *snapshot* and *continuous* RkNN queries. We also present a detailed theoretical analysis to analyse the area of the influence zone and IO costs of our RkNN processing algorithms. Our experiments demonstrate the accuracy of our theoretical analysis.

1 Introduction

The reverse k nearest neighbors (RkNN) query [1, 2, 3, 4, 5, 6, 7, 8] has received significant research attention ever since it was introduced in [1]. A RkNN query finds every data point for which the query point q is one of its k nearest neighbors. Since q is close to such data points, q is said to have high influence on these points. Hence, the set of points that are the RkNNs of a query is called its influence set [1]. Consider the example of a gas station. The drivers for which this gas station is one of the k nearest gas stations are its potential customers. In this paper, the objects that provide a facility or service (e.g., gas stations) are called *facilities* and the objects (e.g., the drivers) that use the facility are called *users*. The influence set of a given facility q is then the set of users for which q is one of its k closest facilities.

In this paper, we first introduce a more generic concept called *influence zone* and then we show that the influence zone can be used to efficiently compute the influence set (i.e., RkNNs). Consider a set of facilities $F = \{f_1, f_2, \cdots, f_n\}$ where f_i represents a point in Euclidean space and denotes the location of i^{th} facility. Given a query $q \in F$, the influence zone Z_k is the area such that for every point $p \in Z_k$, q is one of its k closest facilities and for every point $p' \notin Z_k$, q is not one of its k closest facilities.

The influence zone has various applications in location based services, marketing and decision support systems. Consider the example of a coffee shop. Its influence zone may be used for market analysis as well as targeted marketing. For instance, the demographics of its influence zone may be used by the market researchers to analyse its business. The influence zone can also be used for marketing, e.g., advertising bill boards or posters may be placed in its influence zone because the people in this area are more likely to be influenced by the marketing. Similarly, the people in its influence zone may be sent SMS advertisements.

Note that the concept of the influence zone is more generic than the influence set, i.e., the RkNNs of q can be computed by finding the set of users that are located in its influence zone. In this paper, we show that our influence zone based RkNN algorithms significantly outperform existing best known algorithms for both the *snapshot* and *continuous* RkNN queries (formally defined in Section 2).

Existing RkNN processing techniques [4, 5, 6, 9, 8] require a *verification* phase to finalize the query results. Initially, the space is pruned by using the locations of the facility points. Then, the users that are located in the unpruned space are retrieved. These users are the possible RkNNs and are called candidates. Finally, in the verification phase, a range query is issued for every candidate to check whether it is a RkNN or not.

In contrast to the existing approaches, our influence zone based algorithm does not require the verification phase. Initially, we use our algorithm to efficiently compute the influence zone. Then, every user that is located in the influence zone is reported as RkNN. This is because a user can be the RkNN if and only if it is located in the influence zone. Similarly, to continuously monitor RkNNs, initially the influence zone is computed. Then, to update the results we only need to monitor the users that enter or leave the influence zone (i.e., the users that enter in the influence zone become the RkNNs and the users that leave the influence zone are no more the RkNNs). To further improve the performance, we present efficient methods to check whether a point lies in the influence zone or not.

It is important to note that the influence zone of a query is the same as the Voronoi cell of the query when $k = 1$ [7]. For arbitrary value of k, there does not exist an equivalent representation in literature (i.e., order k Voronoi cell is different from the influence zone). Nevertheless, we show that a precomputed order k Voronoi diagram can be used to compute the influence zone (see Section 5.1). However, using the precomputed Voronoi diagrams is not a good approach to process spatial queries as mentioned in [10]. For instance, the value of k is not known in advance and precomputing several Voronoi diagrams for different values of k is expensive and incurs high space requirement. In Section 5.1, we state several other limitations of this approach.

Below, we summarize our contributions in this paper.

- We present an efficient algorithm to compute the influence zone. Based on the influence zone, we present efficient algorithms that outperform best known techniques for both *snapshot* and *continuous* RkNN queries.
- We provide a detailed theoretical analysis to analyse the IO costs of computing the influence zone and our RkNN processing algorithms, the area of the influence zone and the number of RkNNs. Our experiment results show the accuracy of our theoretical analysis.
- Our extensive experiments on real and synthetic data demonstrate that our proposed algorithms are several times faster than the existing best known algorithms.

The rest of the paper is organized as follows. In Section 2, we define the problem and briefly overview the related work. Section 3 presents our technique to efficiently compute the influence zone. In Section 4, we present efficient techniques to answer RkNN queries by using the influence zone. Theoretical analysis is presented in Section 5 followed by the experiment results in Section 6. Section 7 concludes the paper.

2 Preliminaries

2.1 Problem Definition

First, we define a few terms and notations. Consider a set of facilities $F = \{f_1, f_2, \cdots f_n\}$ and a query $q \in F$ in a Euclidean space[1]. Given a point p, C_p denotes a circle centered at p with radius equal to $dist(p,q)$ where $dist(p,q)$ is the distance between p and q. $|C_p|$ denotes the number of facilities that lie within the circle C_p (i.e., the count of facilities such that for each facility f, $dist(p,f) < dist(p,q)$). Please note that the query q can be one of the k closest facilities of a point p iff $|C_p| < k$. Now, we define influence zone and RkNN queries.

Influence zone Z_k. Given a set of facilities F and a query $q \in F$, the influence zone Z_k is the area such that for every point $p \in Z_k$, $|C_p| < k$ and for every point $p' \notin Z_k$, $|C_{p'}| \geq k$.

[1]Although, like existing techniques [6, 9], focus of this paper is 2-d location data, we show in Appendix (Section 3.4) that the techniques can be extended to higher dimensions.

Now, we define the reverse k nearest neighbor (RkNN) queries. RkNN queries are classified [1] into *bichromatic* and *monochromatic* RkNN queries. Below, we define both.

Bichromatic RkNN queries. Given a set of facilities F, a set of users U and a query $q \in F$, a bichromatic RkNN query is to retrieve every user $u \in U$ for which $|C_u| < k$.

Consider that the supermarkets and the houses in a city correspond to the set of facilities and users, respectively. A bichromatic RkNN query may be used to find every house for which a given supermarket is one of the k closest supermarkets.

Monochromatic RkNN queries. Given a set of facilities F and a query $q \in F$, a monochromatic RkNN query is to retrieve every facility $f \in F$ for which $|C_f| < k + 1$.

Please note that for every f, C_f contains the facility f. Hence we have condition $|C_f| < k + 1$ instead of $|C_f| < k$. Consider a set of police stations. For a given police station q, its monochromatic RkNNs are the police stations for which q is one of the k nearest police stations. Such police stations may seek assistance (e.g., extra policemen) from q in case of an emergency event.

Snapshot vs continuous RkNN queries. In a snapshot query, the results of the query are computed only once. In contrast, in a continuous query, the results are to be continuously updated as the underlying datasets issue location updates. Although extensions are possible, in this paper, we focus on a special case of continuous RkNN queries where only the users issue location updates.

Given a set of facilities F, a query $q \in F$ and a set of users U that issues location updates, a continuous RkNN query is to continuously update the bichromatic RkNNs of q.

A gas station may want to continuously monitor the vehicles for which it is one of the k closest gas stations. It may issue a continuous RkNN query to do so.

Throughout this paper, we use RNN query to refer to the RkNN query for which $k = 1$. Table 2.1 defines other notations used throughout this paper.

Notation	**Definition**
q	the query point
C_p	a circle centered at p with radius $dist(p, q)$
$\|C_p\|$	the number of facilities located inside C_p
$B_{x:q}$	a perpendicular bisector between point x and q
$H_{x:q}$	a half-plane defined by $B_{x:q}$ containing point x
$H_{q:x}$	a half-plane defined by $B_{x:q}$ containing point q

Table 2.1: Notations

2.2 Related work

First, we present related work for *snapshot* RkNN queries.

Snapshot RkNN Queries: Korn *et al.* [1] were first to study RNN queries. They answer the RNN query by pre-calculating a circle for each data object p such that the nearest neighbor of p lies on the perimeter of the circle. RNN of a query q is every point that contains q in its circle. Techniques to improve their work were proposed in [2, 3].

Now, we briefly describe the existing techniques that do not require pre-computation. All these techniques can be easily extended to answer the bichromatic RkNN queries. These techniques have three phases namely *pruning*, *containment* and *verification*. In the pruning phase, the space that cannot contain any RkNN is pruned by using the set of facilities. In the containment phase, the users that lie within the unpruned space are retrieved. These are the possible RkNNs and are called the candidates. In the verification phase, a range query is issued for each candidate object to check if q is one of its k nearest facility or not.

First technique that does not need any pre-computation was proposed by Stanoi *et al.* [4]. They solve RNN queries by partitioning the whole space centred at the query q into six equal regions of 60° each (S_1 to S_6 in Fig. 2.1). It can be proved that the nearest facility to q in each region defines the area that can be pruned. In other words, assume that f is the nearest facility to q in a region S_i. Then any user that lies in S_i and lies at a distance greater than $dist(q, f)$ from q cannot be the RNN of q. Fig. 2.1 shows nearest neighbors of q in each region and the white area can be pruned. Only the users that lie in the shaded area can be the RNNs. The RkNN queries can be solved in a similar way, i.e., in each region, the k-th nearest facility of q defines the pruned area.

Tao *et al.* [5] proposed TPL that uses the property of perpendicular bisectors to prune the search space. Consider the example of Fig. 2.2, where a bisector between q and a is shown as $B_{a:q}$ which divides the space into two half-spaces. The half-space that contains a is denoted as $H_{a:q}$ and the half-space that contains q is denoted as $H_{q:a}$. Any point that lies in the half-space $H_{a:q}$ is always closer to a than to q and cannot be the RNN for this reason. Similarly, any point p that lies in k such half-spaces cannot be the RkNN. TPL algorithm prunes the space by the bisectors drawn between q and its neighbors in the unpruned area. Fig. 2.2 shows the example where the bisectors between q and a, b and c are drawn ($B_{a:q}$, $B_{b:q}$ and $B_{c:q}$, respectively). The white area can be pruned because every point in it lies in at least two half-spaces.

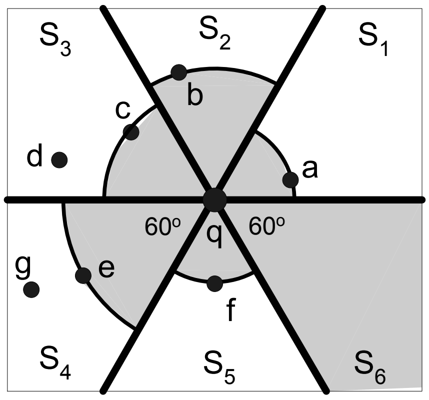


Figure 2.1: Six-regions pruning

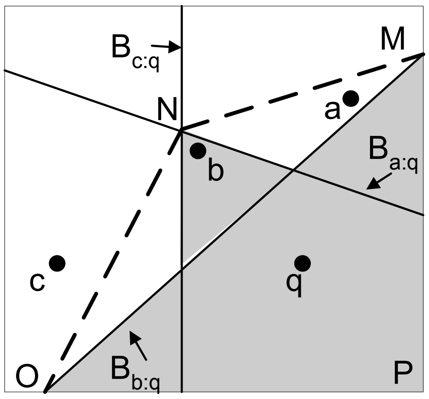


Figure 2.2: TPL and FINCH

In the containment phase, TPL retrieves the users that lie in the unpruned area by traversing an R-tree that indexes the locations of the users. Let m be the number of facility points for which the bisectors are considered. An area that is the intersection of any combination of k half-spaces can be pruned. The total pruned area corresponds to the union of pruned regions by all such possible combinations of k bisectors (a total of $m!/k!(m-k)!$ combinations). Since the number of combinations is too large, TPL uses an alternative approach which has less pruning power but is cheaper. First, TPL sorts the m facility points by their Hilbert values. Then, only the combination of k consecutive facility points are considered to prune the space (total m combinations).

As discussed above, to prune the entries, TPL uses m combinations of k bisectors which is expensive. To overcome this issue, Wu *et. al* [6] proposed an algorithm called FINCH. Instead of using bisectors to prune the objects, they use a convex polygon that approximates the unpruned area. Any object that lies outside the polygon can be pruned. Fig. 2.2 shows an example where the shaded area is the unpruned area. FINCH approximates the unpruned area by a polygon $MNOP$. Any point that lies outside this polygon can be pruned. Clearly, the containment checking is easier than TPL because containment can be done in linear time for convex polygons. However, please note that the area pruned by FINCH is smaller than the area that actually can be pruned.

It is worth mentioning that some of the existing work focus on computing Voronoi cell (or order k Voronoi cell) on the fly. More specifically, Stanoi et al. [7] compute Voronoi cell to answer RNN queries. On fly computation of order k Voronoi cell was presented in [10, 11] to monitor kNN queries. However, these approaches are not applicable for RkNN queries.

Continuous RNN Queries: Benetis *et al.* [12] presented the first continuous RNN monitoring algorithm. However, they assume that velocities of the objects are known. First work that does not assume any knowledge of objects' motion patterns was presented by Xia *et al.* [13]. Their proposed solution is based on the six 60^o regions based approach described earlier in this section. Kang *et al.* [8] proposed a continuous monitoring RNN algorithm based on the bisector based (TPL) pruning approach. Both of these algorithms continuously monitor RNN queries by monitoring the unpruned area.

Wu *et al.* [14] propose the first technique to monitor RkNNs. Their technique is based on the six-regions based RNN monitoring presented in [13]. More specifically, they issue k nearest neighbor (kNN) queries in each region instead of the single nearest neighbor queries. The users that are closer than the k-th NN in each region are the candidate objects and they are verified if q is one of their k closest facilities. To monitor the results, for each candidate object, they continuously monitor the circle around it that contains k nearest facilities.

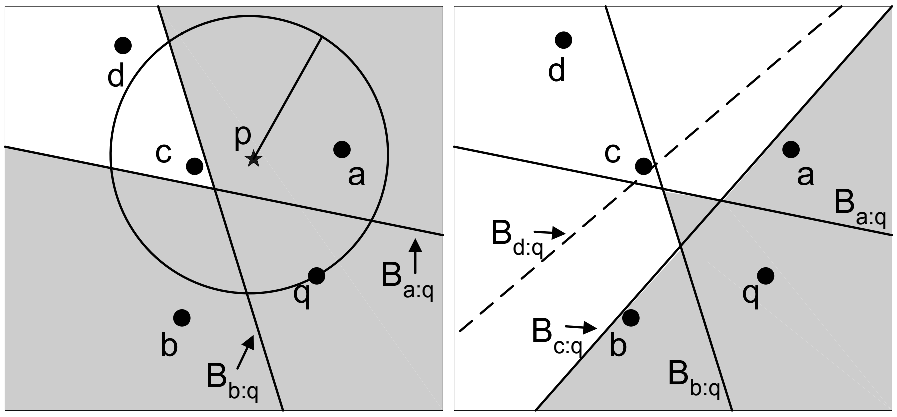
Cheema *et al.* [9] propose Lazy Updates that is the best known algorithm to continuously monitor RkNN queries. The existing approaches call the expensive pruning phase whenever the query or a candidate object changes the location. Lazy Updates saves the computation time by reducing the number of calls to the expensive pruning phase. They assign each moving object a safe region and propose the pruning techniques to prune the space based on the safe regions. The pruning phase is not needed to be called as long as the related objects remain inside their safe regions.

3 Computing Influence Zone

3.1 Problem Characteristics

Given two facility points a and q, a perpendicular bisector $B_{a:q}$ between these two points divides the space into two halves as shown in Fig 3.1(a). The half plane that contains a is denoted as $H_{a:q}$ and the half plane that contains q is denoted as $H_{q:a}$. The perpendicular bisector has the property that any point p (depicted by a star in Fig. 3.1(a)) that lies in $H_{a:q}$ is closer to a than q (i.e., $dist(p, a) \leq dist(p, q)$) and any point y that lies in $H_{q:a}$ is closer to q than a (i.e., $dist(y, q) \leq dist(y, a)$). Hence, q cannot be the closest facility of any point p that lies in $H_{a:q}$, i.e., C_p contains at least one facility a. We say that the point p is pruned by the bisector $B_{a:q}$ if p lies in $H_{a:q}$. In general, if a point p is pruned by at least k bisectors then C_p contains at least k facilities (i.e., $|C_p| \geq k$).

Existing work [5, 6, 9] use this observation to prune the space that cannot contain any RkNN of q. More specifically, an area can be pruned if at least k bisectors prune it. In Fig. 3.1, five facility points (q, a, b, c and d) are shown. In Fig. 3.1(a) the bisectors between q and two facility points a and b are drawn (see $B_{a:q}$ and $B_{b:q}$). If k is 2, then the white area can be pruned because it lies in two half-planes ($H_{a:q}$ and $H_{b:q}$) and $|C_{p'}| \geq 2$ for any point p' in it. The area that is not pruned is called unpruned area and is shown shaded.



(a) Unpruned area is not influence zone (b) Unpruned area is influence zone

Figure 3.1: Computing influence zone Z_k ($k = 2$)

Although it can be guaranteed that for every point p' in the pruned area $|C_{p'}| \geq k$, it cannot be guaranteed that for every point p in the unpruned area $|C_p| < k$. In other words, the unpruned area is not the influence zone. For example, in Fig. 3.1(a), the point p lies in the unpruned area but $|C_p| = 2$ (i.e., C_p contains a and c). Hence, the shaded area of Fig. 3.1(a) is not the influence zone.

One straight forward approach to compute the influence zone is to consider the bisectors of q with every facility point f. If the bisectors of q and all facilities are considered, then the unpruned area is the area that is pruned by less than k bisectors. Fig. 3.1(b) shows the unpruned area (the shaded polygon) after the bisectors $B_{c:q}$ and $B_{d:q}$ are also considered. It can be verified that the shaded area is the influence zone (i.e., for every p in the shaded area $|C_p| < 2$ and for every p' outside it $|C_{p'}| \geq 2$).

However, this straight forward approach is too expensive because it requires computing the bisectors between q and all facility points. We note that for some facilities, we do not need to consider their bisectors. In Fig. 3.1(b), it can be seen that the bisector $B_{d:q}$ (shown in broken line) does not affect the unpruned area (shown shaded). In other words, if the bisectors of a, b and c are considered then the bisector $B_{d:q}$ does not prune more area. Hence, even if $B_{d:q}$ is ignored, the influence zone can be computed.

Next, we present some lemmas that help us in identifying the facilities that can be ignored. Since we use bisectors to prune the space, the unpruned area is a polygon and is interchangeably called unpruned polygon hereafter. Below we present several lemmas that not only guide us to the final lemma but also help us in few other proofs in the paper.

LEMMA 1 : A facility f can be ignored if for *every* point p of the unpruned polygon, the facility f lies outside C_p.

PROOF. As described earlier, a point p can be pruned by the bisector $B_{f:q}$ iff $dist(p, f) < dist(p, q)$. In other words, the point p can be pruned iff C_p contains f. Hence, if f lies outside C_p, it cannot prune p. If f lies outside C_p for *every* point p, it cannot prune *any* point of the unpruned polygon and can be ignored for this reason.

Checking containment of f in C_p for every point p is not feasible. In next few lemmas, we simplify the procedure to check if a facility point can be ignored.

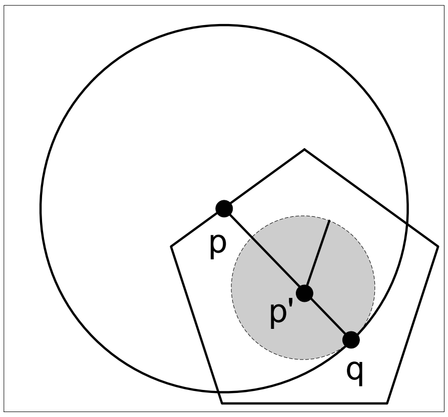


Figure 3.2: Lemma 2 and 3

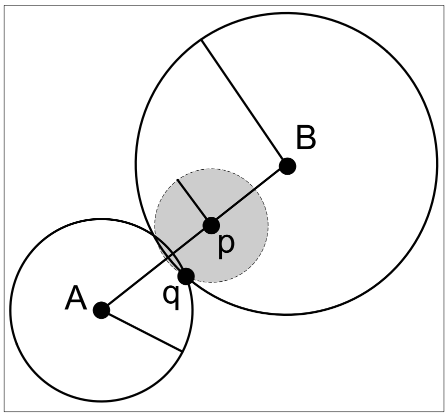


Figure 3.3: Lemma 4

LEMMA 2 : Let pq be a line segment between two points q and p. Let p' be a point on pq. The circle $C_{p'}$ is contained by the circle C_p.

Fig. 3.2 shows an example where the circle $C_{p'}$ (the shaded circle) is contained by C_p (the large circle). The proof is straight forward and is omitted. Based on this lemma, we present our next lemma.

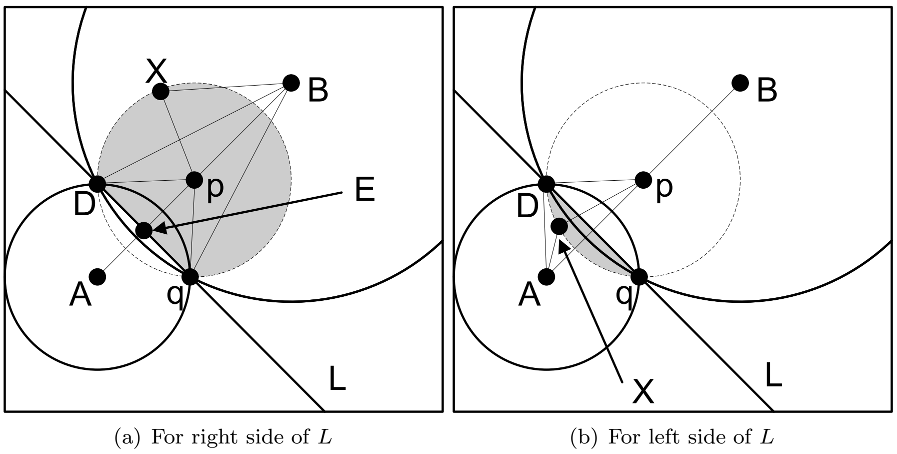
LEMMA 3 : A facility f can be ignored if, for *every* point p on the *boundary* of the unpruned polygon, f lies outside C_p.

PROOF. We prove the lemma by showing that we do not need to check containment of f in $C_{p'}$ for any point p' that lies within the polygon. Let p' be a point that lies within the polygon. We draw a line that passes through q and p' and cuts the polygon at a point p (see Fig. 3.2). From Lemma 2, we know that C_p contains $C_{p'}$. Hence, if f lies outside C_p, then it also lies outside $C_{p'}$. Hence, it suffices to check the containment of f in C_p for every point p on the boundary of the polygon.

The next two lemmas show that we can check if a facility f can be ignored or not by only checking the containment of f in C_v for every vertex v of the unpruned polygon.

LEMMA 4 : Given a line segment AB and a point p on AB. The circle C_p is contained by $C_A \cup C_B$, i.e., every point in the circle C_p is either contained by C_A or by C_B (see Fig. 3.3).

PROOF. Fig. 3.4 shows the line segment AB and the point p. It suffices to show that the boundary of C_p is contained by $C_A \cup C_B$. If q lies on AB, the lemma can be proved by Lemma 2. Otherwise, we identify a point D such that AB is a segment of the perpendicular bisector between D and q. Then, we draw a line L that passes through points D and q. First, we show that the part of the circle C_p that lies on the right side of L (i.e., the shaded part in Fig. 3.4(a)) is contained by C_B. Then, we show that the part of the circle C_p that lies on the left side of L (i.e., the shaded part in Fig. 3.4(b)) is contained by C_A.



(a) For right side of L (b) For left side of L

Figure 3.4: Illustration of Lemma 4

We can find the length of qB (denoted as $\overline{qB}$) by using the triangle $\triangle qpB$ and applying the law of cosines (see Fig. 3.4(a)).

$$\overline{qB} = \sqrt{(\overline{pB})^2 + (\overline{pq})^2 - 2 \cdot \overline{pB} \cdot \overline{pq}(Cos\angle Bpq)} \tag{3.1}$$

For any point X that lies on the boundary of C_p and is on the right side of L (i.e., the boundary of the shaded circle in Fig. 3.4(a)), consider the triangle $\triangle\ pXB$. The length of BX can be computed using the law of cosines.

$$\overline{BX} = \sqrt{(\overline{pB})^2 + (\overline{pX})^2 - 2 \cdot \overline{pB} \cdot \overline{pX}(Cos\angle BpX)} \tag{3.2}$$

Please note that the triangles $\triangle qpB$ and $\triangle DpB$ are similar because $\overline{Dp} = \overline{qp}$ and $\overline{DB} = \overline{qB}$ (any point on a perpendicular bisector $B_{u:v}$ is equi-distant from u and v). Due to similarity of triangles $\triangle qpB$ and $\triangle DpB$, $\measuredangle Bpq = \measuredangle BpD$.

It can be shown that $\overline{BX} \leq \overline{qB}$ by comparing Eq. (3.1) and Eq. (3.2). This is because $\overline{pX} = \overline{pq}$ and $\measuredangle BpX \leq (\measuredangle Bpq = \measuredangle BpD)$. Since cosine monotonically decreases as the angle increases from 0° to 180°, $\overline{BX} \leq \overline{qB}$. This means the point X lies within the circle C_B.

Similarly, for any X that lies on the part of circle C_p that is on left side of the line L (see Fig. 3.4(b)) it can be shown that $\overline{AX} \leq (\overline{AD} = \overline{Aq})$. This can be achieved by considering the triangles $\triangle pXA$ and $\triangle pDA$ and using law of cosines to obtain $\overline{AX}$ and $\overline{AD}$ (the key observation is that $\measuredangle XpA \leq \measuredangle DpA$).

LEMMA 5 : A facility f can be ignored if, for every vertex v of the unpruned polygon, the facility f lies outside C_v.

PROOF. Let AB be an edge of the polygon. From Lemma 4, we know that if a facility f lies outside C_A and C_B, then it lies outside C_p for every point p on the edge AB. This implies that if f lies outside C_v for every vertex v of the polygon then it lies outside C_p for every point p that lies on the boundary of the polygon. Such facility f can be ignored as stated in Lemma 3.

Next lemma shows that we only need to check this condition for *convex* vertices. First, we define the convex vertices.

DEFINITION 1 : Consider a polygon P where V is the set of its vertices. Let H_{con} be the convex hull of V. The vertices of H_{con} are called convex vertices of the polygon P and the set of the convex vertices is denoted as V_{con}.

Fig. 3.5 shows an example where a polygon with vertices A to J is shown in broken lines. Its convex hull is shown in solid lines which contains the vertices A, C, E, G and I and these vertices are the convex vertices. Note that $V_{con} \subseteq V$.

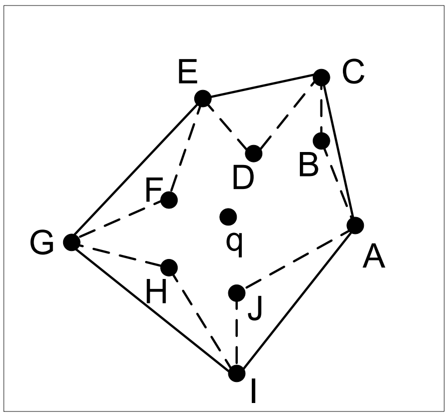


Figure 3.5: Convex Polygon

LEMMA 6 : A facility f can be ignored if it lies outside C_v for every *convex vertex* v of the unpruned polygon P.

Proof. By definition of a convex hull, the convex hull H_{con} contains the polygon P. If a facility point f does not prune any point of the convex polygon H_{con}, it cannot prune any point of the polygon P because $P \subseteq H_{con}$. Hence, it suffices to check if f prunes any point of H_{con} or not. From Lemma 5, we know that f does not prune any point of H_{con} if it lies outside C_v for every vertex v of H_{con}. Hence, f can be ignored if it lies outside every C_v where v is a vertex of the convex polygon (i.e., v is a convex vertex).

The above lemma identifies a condition for a facility f to be ignored. Next lemma shows that any facility that does not satisfy this condition prunes at least one point of the unpruned area. In other words, next lemma shows that the above condition is tight.

Lemma 7 : If a facility f lies in *any* C_v for any convex vertex v of the unpruned polygon P then there exists at least one point p in the polygon P that is pruned by f.

Proof. If f lies in C_v for any $v \in V_{con}$, it means that $dist(f, v) < dist(f, q)$. Hence, f prunes the vertex v. Since $V_{con} \subseteq V$, the vertex v is a point in the polygon P.

3.2 Algorithm

Based on the problem characteristics we described earlier in this section, we propose an algorithm to efficiently compute the influence zone. We assume that the facilities are indexed by an R-tree [15]. The main idea is that the facilities are iteratively retrieved and the space is iteratively pruned by considering their bisectors with q. The facilities that are close to the query q are expected to prune larger area and are given priority.

Algorithm 1 presents the details. Initially, the whole data space is considered as the influence zone and the root of the R-tree is inserted in a min-heap h. The entries are iteratively de-heaped from the heap. The entries in the heap may be rectangles (e.g., intermediate nodes) or points. If a de-heaped entry e completely lies outside C_v of all convex vertices of the current influence zone (e.g., the current unpruned area), it can be ignored. Otherwise, it is considered valid (lines 5 to 7). If the entry is valid and is an intermediate node or a leaf node, its children are inserted in the heap (lines 8 to 10). Otherwise, if the entry e is valid and is a data object (e.g., a facility point), it is used to prune the space. The current influence zone is also updated accordingly (line 12). The algorithm stops when the heap becomes empty.

The proof of correctness follows from the lemmas presented in the previous section because only the objects that do not affect the unpruned area are ignored. It is also important to note that the entries of R-tree are accessed in ascending order of their minimum distances to the query. The nearby facility points are accessed and the unpruned area keeps shrinking which results in a greater number of upcoming entries being pruned. Hence, the entries that are far from the query are never accessed.

Now, we briefly describe how to update the influence zone when a new facility point f is considered (line 12 of Algorithm 1). The idea is similar to [6]. The intersection points between all the bisectors are maintained. Each intersection point is assigned a counter that denotes the number of bisectors that prune it.

Algorithm 1 Compute Influence Zone

```
Input:   a set of objects $O$, a query $q \in O$, $k$
Output:  Influence Zone $Z_k$
 1: initialize $Z_k$ to the boundary of data universe
 2: insert root of R-tree in a min-heap $h$
 3: while $h$ is not empty do
 4:   deheap an entry $e$
 5:   for each convex vertex $v$ of $Z_k$ do
 6:     if $mindist(v, e) < dist(v, q)$ then
 7:       mark $e$ as valid; break
 8:   if $e$ is valid then
 9:     if $e$ is an intermediate node or leaf then
10:       insert every child $c$ in $h$ with key $mindist(q, c)$
11:     else if $e$ is an object then
12:       update the influence zone $Z_k$ using $e$
```

Fig. 3.6 shows an example ($k = 2$) where three bisectors $B_{a:q}$, $B_{b:q}$ and $B_{c:q}$ have been considered. The counter of intersection point v_{11} is 2 because it is pruned by $B_{b:q}$ and $B_{c:q}$. The counter of v_8 is 1 because it is pruned only by $B_{c:q}$. It can be immediately verified that the unpruned area can be defined by only the intersection points with counters less than k [6] (see the shaded area of Fig. 3.6).

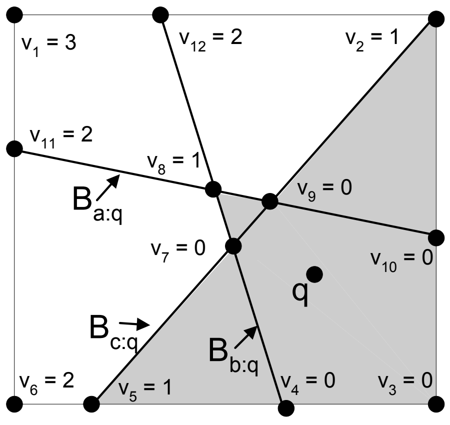


Figure 3.6: Counters

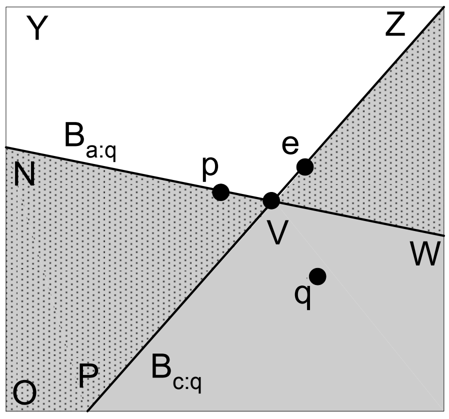


Figure 3.7: Lemma 8

Next issue is to determine the convex vertices of the unpruned area. One straight forward approach to determine the convex vertices is to compute the convex hull of all intersection points that have counters less than k. However, please note that the number of intersection points may be $O(m^2)$ where m is the number of bisectors considered so far. In [6], the authors show that the number of vertices can be reduced from $O(m^2)$ to $O(m)$. However, the computation of convex hull on these $O(m)$ vertices costs $O(m \ Log \ m)$. Following lemma shows that we do not need to compute the convex hull (in contrast to [6]) to shortlist the vertices that are possibly the convex vertices.

Lemma 8 : Among the intersection points that do not lie on the boundary of the data space, only the intersection points with counters equal to $k-1$ can be the convex vertices.

Proof. Any intersection that has a counter greater than $k-1$ is pruned by at least k objects hence cannot be on the boundary of the influence zone (hence, cannot be a convex vertex). Now, we show that the intersections that have counters less than $k-1$ cannot be the convex vertices.

Consider the example of Fig. 3.7 where a vertex V has been shown which is the intersection point of two bisectors $B_{a:q}$ and $B_{c:q}$. Suppose that the counter of the vertex V is n. Now, imagine a point p that lies on the line VN and is infinitely close to the vertex V. Clearly, the point p is pruned by at most $n+1$ bisectors[1]. This is because it is pruned by n bisectors that prune V and the bisector $B_{c:q}$. Following the similar argument, we can say that any point e that lies on the line VZ and is infinitely close to V has a counter at most $n+1$. The counter of any point that lies in the polygon $VNYZ$ (white area) and is infinitely close to V is at least $n+2$ (it is pruned by $B_{c:q}$ and $B_{a:q}$ in addition to all the bisectors that prune V).

If the counter n of the vertex V is less than or equal to $k-2$, then the line VN has at least one point p that has counter at most $k-1$ (i.e., $n+1$ as shown above). Hence, the line VN has at least one point p that lies in the influence zone. Similarly, the line VZ has at least one point e that lies in the influence zone. Clearly, the angle eVp is at least $180°$. By definition of a convex hull, no internal angle of a convex hull can be greater than 180^0. Hence, the vertex V is not a convex vertex if its counter is less than or equal to $k-2$.

In Fig. 3.6, the vertices v_7 and v_9 do not lie on the boundary and have counters less than $k-1$ (where $k=2$). Hence, they are not the convex vertices. Among the points that lie on the boundary points and have counters less than k, only the two extreme points for each boundary line can be the convex vertices. For example, in Fig. 3.6, the lower horizontal boundary line contains 4 vertices (v_3, v_4, v_5 and v_6). The vertex v_6 has counter not less than k and can be ignored. Among the remaining vertices, we consider the extreme vertices (v_3 and v_5) as the convex vertices. Following the above strategy, the convex vertices in Fig. 3.6 are v_3, v_2, v_8 and v_5.

It can be shown that the number of possible vertices with counters equal to $k-1$ are $O(m)$ where m is the number of bisectors considered so far [6]. Hence, checking whether an entry of the R-tree is valid or not requires $O(m)$ distance computations (see lines 5- 7 of Algorithm 1). Next, we present few observations and show that we can determine the validity of some entries by a single distance computation.

Lemma 9 : Let r_{min} be the minimum distance of q to the boundary of the influence zone. Then, an entry e is a valid entry if $mindist(q, e) < 2r_{min}$ (Fig. 3.8 shows r_{min}).

Proof. To prove that e is a valid entry, we show that there exists at least one point p in the influence zone such that C_p contains e. If e lies inside the

[1]In this proof, we assume that only two bisectors pass through the intersection point V. For the special case, when more than two bisectors pass through a vertex V, we may choose to treat V as a convex vertex. Note that this does not affect the correctness of the algorithm because checking containment in a vertex that is not a convex vertex does not affect the correctness.

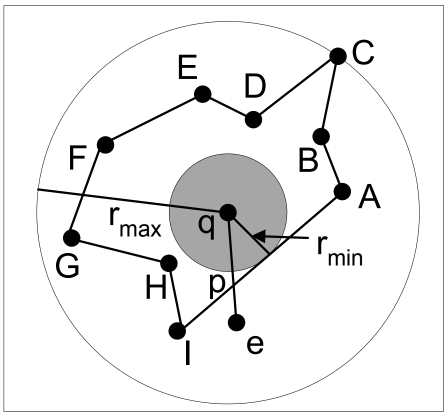


Figure 3.8: r_{max} and r_{min}

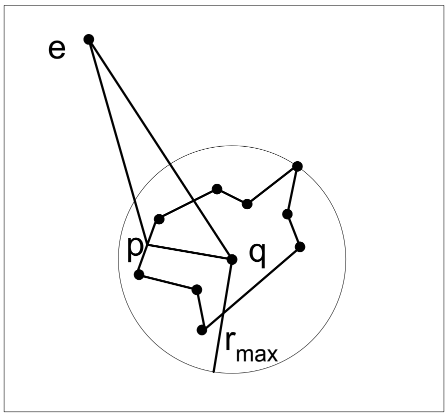


Figure 3.9: Lemma 10

influence zone then e is a valid entry because C_e contains e and e is a point in the influence zone. Now, we prove the lemma for the case when e lies outside the influence zone. Fig. 3.8 shows an entry e for which $dist(q,e) < 2r_{min}$. We draw a line that passes through e and q and intersects the boundary of the influence zone at a point p. Clearly, $dist(p,e) = dist(q,e) - dist(p,q)$. We know that $dist(q,e) < 2r_{min}$ and $dist(p,q) \geq r_{min}$. Hence, $dist(p,e) \leq r_{min}$ which implies that $dist(p,e) \leq dist(p,q)$. Hence, e lies in C_p.

Lemma 10 : Let r_{max} be the distance of q to the furthest vertex of the influence zone. Then, an entry e of the R-tree is an invalid entry if $mindist(e,q) > 2r_{max}$.

Proof. Fig. 3.9 shows r_{max} and a point e such that $dist(e,q) > 2r_{max}$. Consider a point p on the boundary of the influence zone. By the definition of r_{max}, $dist(p,q) \leq r_{max}$. Clearly, $dist(p,q) + dist(p,e) \geq dist(q,e)$ (this covers both the cases when p lies on the line qe and when $\triangle qpe$ is a triangle). Since, $dist(p,q) \leq r_{max}$ and $dist(e,q) > 2r_{max}$, $dist(p,e)$ must be greater than r_{max}. Hence, $dist(p,e) > dist(p,q)$ which means e lies outside C_p. This holds true for every point p on the boundary of the influence zone. Hence, e can be ignored (i.e., e is invalid).

If an entry of the R-tree satisfies one of the above two lemmas, we can determine its validity without computing its distances from the convex vertices. Note that r_{max} and r_{min} can be computed in linear time to the number of edges of the influence zone and are only computed when the influence zone is updated at line 12 of Algorithm 1.

3.3 Checking containment in the influence zone

The applications that use influence zone may require to frequently check if a point or a shape lies within the influence zone or not. Although the suitability of a method to check the containment depends on the nature of the application, we briefly describe few approaches.

One simple approach is to record all the objects that were accessed during the construction of the influence zone (the objects for which the bisectors were considered). If a shape is pruned by less than k of these bisectors then the shape lies inside the influence zone otherwise it lies outside the influence zone. This approach takes linear time in number of the accessed objects. Moreover, checking whether a point is pruned by a bisector $B_{a:q}$ is easy (e.g., if $dist(p, a) < dist(p, q)$ then the point p is pruned otherwise not). Hence, a point containment check requires $O(m)$ distance computations where m is the number of the accessed objects.

Before we show that the point containment can be done in logarithmic time, we define a *star-shaped polygon* [16]. A polygon is a star-shaped polygon if there exists a point z in it such that for each point p in the polygon the segment zp lies entirely in the polygon. The point z is called a kernel point. The polygon shown in Fig. 3.8 is a star-shaped polygon and q is its kernel point. Fig. 4.1 shows a polygon that is not star-shaped (the segment qp does not lie entirely in the polygon). Let n be the number of vertices of a star-shaped polygon. After a linear time pre-processing, every point containment check can be done in $O(Log\ n)$ if a kernel point of the polygon is known [16].

LEMMA 11 : The influence zone is always a star-shaped polygon and q is its kernel point.

PROOF. We prove this by contradiction. Assume that there is a point p in the influence zone such that the segment pq does not lie completely within the influence zone. Fig. 4.1 shows an example, where a point p' lies on the segment pq but does not lie within the influence zone. From Lemma 2, we know that C_p contains $C_{p'}$. Since p is a point inside the influence zone, $|C_p| < k$. As $C_{p'}$ is contained by C_p, $|C_{p'}|$ must also be less than k. Hence, p' cannot be a point outside the influence zone.

Following similar argument as in [6], it can be shown that the number of vertices of the influence zone is $O(m)$ where m is the number of the objects accessed during the construction of the influence zone. Hence, the point containment check can be done in $O(Log\ m)$.

Although we showed that the point containment in influence zone can be checked in logarithmic time, we present two simple checks to reduce the cost of containment check in certain cases by using r_{max} and r_{min} we introduced earlier.

Let r_{min} and r_{max} be as defined in Lemma 9 and 10, respectively. Then the circle centered at q with radius r_{max} (the big circle in Fig. 3.8) completely contains the influence zone. Similarly, the circle centered at q with radius r_{min} (the shaded circle in Fig. 3.8) is completely contained by the influence zone. Hence, any point p that has a distance greater than r_{max} from q is not contained by the influence zone and any point p' that lies within distance r_{min} of q is contained by the influence zone.

For the applications that allow relatively expensive pre-processing, the influence zone can be indexed (e.g., by a grid or a quad-tree) to efficiently check the containment. For example, for the continuous monitoring of RkNN queries, we use a grid to index the influence zone. The details are presented in next section.

3.4 Extension to higher dimensions

In this sectiom, we show that the proposed approach can be extended to high-dimensional data. In higher dimensions, the bisectors are called half-spaces and the unpruned region is a polyhedron instead of a polygon [17]. The circle C_p centered at p with radius $dist(p, q)$ is called hypersphere. It can be shown that Lemma 4 holds for higher dimensions. This can be proved, for each point of the hypersphere, by projection on a two dimensional space.

The space is pruned in a similar way as by the bisectors. i.e., the space that is pruned by at least k half-spaces is pruned. The following lemma holds for the unpruned area which is a polyhedron.

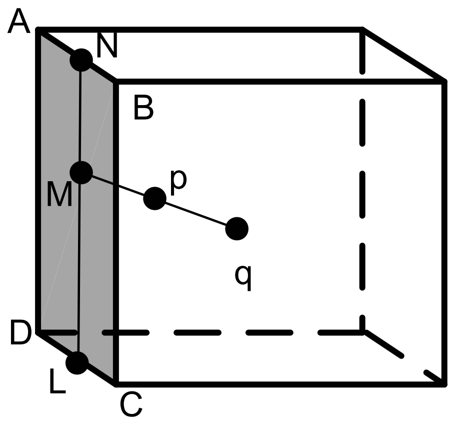


Figure 3.10: Lemma 12

LEMMA 12 : A facility point f can be ignored if for every vertex v of the unpruned polyhedron, f lies outside C_v.

PROOF. We prove the lemma for a 3-dimensional polyhedron and the proof for the arbitrary dimensionality is similar. Let p be any point inside the polyhedron as shown in Fig. 3.10. We draw a line that passes through p and q and crosses a face (the shaded face $ABCD$) of the polyhedron at a point M. For such point M, we can always draw a line on this face of the polyhedron such that it passes through M and intersects the edges of the face at points L and N as shown in Fig 3.10. From Lemma 4, C_A and C_B contain C_N. Similarly, C_C and C_D contain C_L. Again, from Lemma 4, C_N and C_L contain C_M. Lastly, C_M contains C_p (Lemma 2). Hence, C_p is contained by the hyperspheres of the vertices of the face $ABCD$ (C_A, C_B, C_C and C_D). This holds for any arbitrary point p inside the polyhedron. Hence, we only need to check the containment in C_v for every vertex v of the polyhedron.

4 Applications in RkNN Processing

4.1 Snapshot Bichromatic RkNN Queries

Our algorithm consists of two phases namely *pruning* phase and *containment* phase.

Pruning Phase. In this phase, the influence zone Z_k is computed using the given set of facilities.

Containment Phase. By the definition of influence zone Z_k, a user u can be the bichromatic RkNN if and only if it lies within the influence zone Z_k. We assume that the set of users are indexed by a R-tree. The R-tree is traversed and the entries that lie outside the influence zone are pruned. The objects that lie in the influence zone are RkNNs.

4.2 Snapshot Monochromatic RkNN Queries

By definition of a monochromatic RkNN query (see Section 2.1), a facility f is the RkNN iff $|C_f| < k+1$. Hence, a facility that lies in Z_{k+1} is the monochromatic RkNN of q where Z_{k+1} is the influence zone computed by setting k to $k+1$. Below, we highlight our technique.
Pruning Phase. In this phase, we compute the influence zone Z_{k+1} using the given set of facilities F. We also record the facility points that are accessed during the construction of the influence zone and call them the candidate objects.
Containment Phase. Please note that every facility point that is contained in the influence zone Z_{k+1} will be accessed during the pruning phase. This is because every facility that lies in the influence zone cannot be ignored during the construction of the influence zone (inferred from Lemma 1). Hence, the set of candidate object contains all possible RkNNs. For each of the candidate object, we report it as RkNN if it lies within the influence zone Z_{k+1}.

4.3 Continuous monitoring of RkNNs

In this section, we present our technique to continuously monitor bichromatic RkNN queries (see the problem definition in Section 2.1). The basic idea is to index the influence zone by a grid. Then, the RkNNs can be monitored by tracking the users that enter or leave the influence zone.

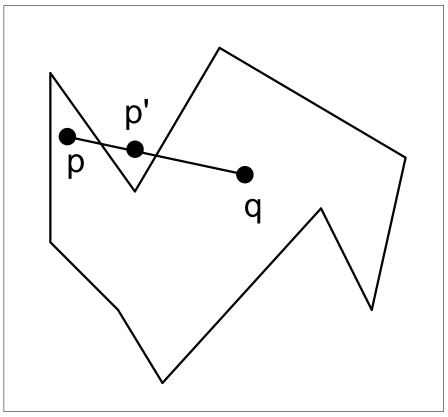


Figure 4.1: Lemma 11

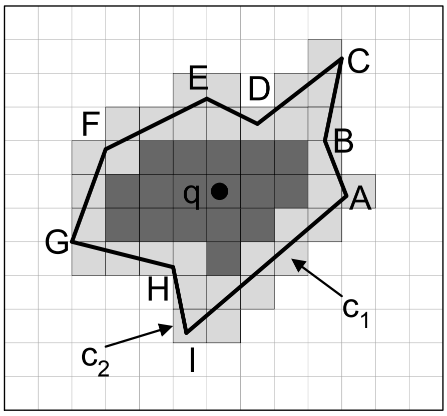


Figure 4.2: Continuous Monitoring

Initially, the influence zone Z_k of a query q is computed by using the set of facility points. We use a grid based data structure to index the influence zone. More specifically, a cell c of the grid is marked as an *interior* cell if it is completely contained by the influence zone. A cell c' is marked as a *border* cell if it overlaps with the boundary of the influence zone. Fig. 4.2 shows an example where the influence zone is the polygon $ABCDFEGHI$, interior cells are shown in dark shade and the border cells are the light shaded cells.

For each border cell, we record the edges of the polygon that intersect it. For example, in c_1, we record the edge AI and in c_2 we record the edges AI and HI. If a user $u \in U$ is in an interior cell, we report it as RkNN of the query. If a user lies in a border cell, we check if it lies outside the polygon by checking the edges stored in this cell. For example, if a user lies in c_1 and it lies inside AI, we report it as RkNN.

5 Theoretical Analysis

We assume that the facilities and the users are uniformly distributed in a unit space. The number of facilities is $|F|$. For bichromatic queries, the number of users is $|U|$.

5.1 Area of Influence Zone

Before we analyse the area of the influence zone, we show the relationship between an order k Voronoi cell and the influence zone. We utilize this relationship to analyse the area of the influence zone.

Relationship with order k Voronoi cell: An order k Vornoi diagram divides the space into cells and we refer to each cell as a k-Voronoi cell. Each k-Voronoi cell is related to a set of k facility points (denoted as F_k) such that for any point p in this cell the k closest facilities are F_k. Fig. 5.1 shows an order 2 Voronoi diagram computed on the facility points a to i. Each cell c is related to two facility points (shown as $\{f_i, f_j\}$ in Fig. 5.1) and these are the two closest facilities for any point p in c. For example, for any point p in the cell marked as $\{a, e\}$ the two closest facilities are a and e.

Clearly, when $k = 1$ the k-Vornoi cell related to q is exactly same as the influence zone. For $k > 1$, the influence zone corresponds to the union of all k-Voronoi cells that are related to q (i.e., have q in their F_k). For example, in Fig. 5.1, the influence zone of the facility a is shown in bold boundary and it corresponds to the union of the cells that are related to a.

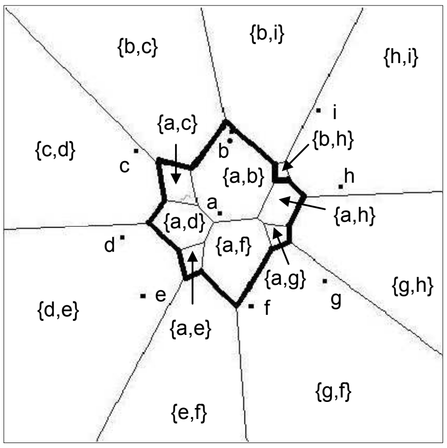


Figure 5.1: order 2 Voronoi diagram

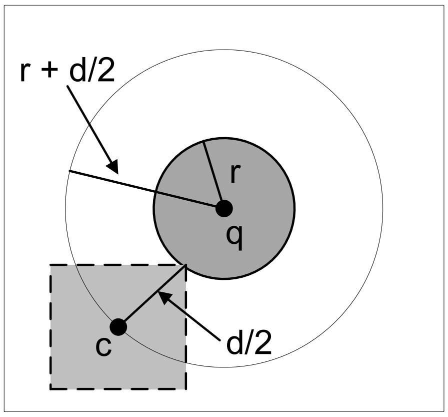


Figure 5.2: Range query

Now, we analyse the area of the influence zone.

As described earlier, the influence zone Z_k is the union of k-Voronoi cells that are related to q. The expected size of one k-Voronoi cell is $\frac{1}{(2k-1)|F|}$ [17]. Since we assume a unit space, the total number of the cells in the Voronoi diagram is $(2k-1)|F|$. Each cell is related to k facilities, this means that the expected number of cells related to q is $k(2k-1)$. Hence, the area of the influence zone Z_k is $k/|F|$ which is computed by multiplying the number of cells related to q with the area of one k-Voronoi cell.

Remark: The above discussion shows that the influence zone can be computed by using a pre-computed order k Voronoi diagram. However, as mentioned in [10], a technique that uses a pre-computed order k Voronoi diagram may not be practical for the following reasons : i) the value of k may not be known in advance; ii) even if k is known in advance, order k Voronoi diagrams are very expensive to compute and incur high space requirement; iii) spatial indexes are useful for all query types and pre-computed Voronoi diagrams may not be used for all queries. In contrast, R-tree based indexes used by our algorithm are used for many important queries.

5.2 Number of RkNNs

First, we evaluate the number of bichromatic RkNNs. We assume that the users are uniformly distributed in the space. The number of users that lie in the influence zone is the number of bichromatic RkNNs. Hence, the number of bichromatic RkNNs is $|U|.k/|F|$.

The area of the influence zone Z_{k+1} for a monochromatic RkNN query is $(k+1)/|F|$. The number of facilities in this area is $(k+1)$ which includes the query. Hence the expected number of monochromatic RkNNs is k.

5.3 IO cost of our algorithms

Before we analyse the IO costs of our proposed algorithms, we analyse the cost of a *circular range* query. Then, we analyse the costs of our algorithms by using the IO cost of the circular range queries.

IO cost of a circular range query: A circular range query [18] finds the objects that lie within distance r of the query location. We assume that the objects are indexed by an R-tree and analyse the number of nodes that lie within the range of the query. Fig. 5.2 shows a circular range query where the search area is the circle centered at q with radius r (the shaded circle). The approach to analyse the IO cost of the circular range query is similar to the IO cost analysis of window queries presented in [19]. Let R_l be the number of rectangles at level l of the R-tree. Let s_l be the side length of each rectangle at level l (the rectangles of a good R-tree have similar sizes [20]). We assume that the centers of rectangles at each level follow a uniform distribution. Let d_l be the diagonal length of each rectangle at level l. As shown in Fig. 5.2, any rectangle that has its center c at a distance at most $r + d_l/2$ intersects the range query and should be accessed. Hence, the number of rectangles (nodes) accessed at level l is $\pi(r + d_l/2)^2 R_l$ which is the number of center points c that lie in the circle of radius $r + d_l/2$ (the large circle in Fig. 5.2).

Now, we need to compute d_l and R_l for each level l. Let S be the number of objects indexed by the R-tree. Let f be the fanout of the tree. The number of rectangles R_l at level l of the R-tree is S/f^l (e.g., leaf nodes are at level 1 and the number of leaf level rectangles is S/f). Since we assume uniform distribution of points, each rectangle at level l contains f^l points. In other words, the area of each rectangle is f^l/S. Assuming that the both sides of a rectangle are of same size, the side length s_l is $\sqrt{f^l/S}$. Given s_l, half of the diagonal length $d_l/2$ can be computed easily which is $\sqrt{f^l/2S}$.

The total IO cost (the total number of nodes accessed) is obtained by applying the formula for each level l. The total number of levels excluding the root is $\lfloor log_f S \rfloor$. The root is accessed anyway, so one is added to this cost. Hence, the total IO cost is obtained by the following equation.

$$Range\ query\ cost = 1 + \sum_{l=1}^{\lfloor log_f S \rfloor} \pi(r + \sqrt{f^l/2S})^2 S/f^l \tag{5.1}$$

Based on this, first we analyse the cost of computing the influence zone and then we analyse the costs of our RkNN algorithms.

IO cost of computing the influence zone: We approximate the influence zone to a circular shape having the same area (we noted that as k gets larger the shape of influence zone has more resemblance with a circle). Since the area of the influence zone Z_k is $k/|F|$, the radius of the circle can be computed as $r_k = \sqrt{\frac{k}{\pi|F|}}$. From Lemma 5, an object can be ignored if it lies at a distance greater than $dist(q, v)$ from every vertex v of the unpruned area. Since we assume that each vertex is at same distance r_k from the query (i.e., influence zone is a circle), an object can be ignored if it lies at a distance greater than $2r_k$ from q. Hence, the objects within the range $2r_k$ of the query are accessed during the computation of the influence zone. The IO cost can be found by replacing r in Eq. (5.1) with $2r_k = 2\sqrt{\frac{k}{\pi|F|}}$ and S with $|F|$ (the number of the facility points).

IO cost of a monochromatic RkNN query: The IO cost for monochromatic RkNN query is same as computing the IO cost of the influence zone Z_{k+1}. This is because the R-tree is traversed only during the construction of the influence zone (i.e., the containment phase does not access R-tree). Hence, IO cost can be found by replacing r in Eq. (5.1) with $2r_{k+1} = 2\sqrt{\frac{k+1}{\pi|F|}}$ and S with $|F|$.

IO cost of a bichromatic RkNN query: The cost of the pruning phase is same as the cost of computing the influence zone Z_k which we have computed earlier. The cost of the containment phase is the cost of accessing the users that lie within the influence zone which can be computed in a similar way. More specifically, only the users that lie within distance r_k (the radius of the influence zone) of q are accessed. Hence, the cost of the containment phase can be computed by replacing r in Eq. (5.1) with $r_k = \sqrt{\frac{k}{\pi|F|}}$ and S with $|U|$ where $|U|$ is the number of users indexed by the R-tree.

6 Experiments

In Section 6.1, we evaluate the performance of our algorithms for snapshot RkNN queries. Since computation of the influence zone is a sub-task of the snapshot RkNN queries, we evaluate the cost of computing influence zone while evaluating the performance of RkNN algorithms. In Section 6.2, we evaluate the performance of our algorithm for continuous monitoring of RkNN queries.

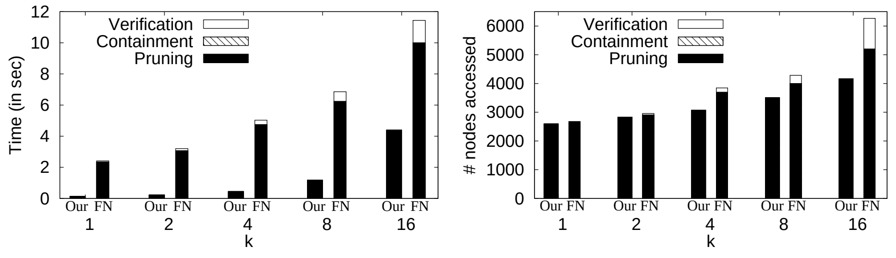
6.1 Snapshot RkNN queries

For monochromatic and bichromatic RkNN queries, we compare our algorithm with the best known existing algorithm called FINCH [6]. We use both synthetic and real datasets. Each synthetic dataset consists of 50000, 100000, 150000 or 200000 points following either Uniform or Normal distribution. The real dataset consists of $175,812$ extracted locations in North America[1] and we randomly divide these points into two sets of almost equal sizes. One of the sets corresponds to the set of facilities and the other to the set of users. In accordance with FINCH [6], the page size is set to 4096 bytes and the buffer size is set to 10 pages which uses random eviction strategy. We use the two real datasets to evaluate the performance unless mentioned otherwise. We vary k from 1 to 16 and the default value is 8. From the set of facilities, we randomly choose 500 points as the query points. The experiment results correspond to the total cost of processing these 500 queries.

As stated in Section 2.2, FINCH has three phases namely pruning, containment and verification. Our algorithm has only pruning and containment phases. We show the CPU and IO cost of each phase for the both algorithms. Experiment results demonstrate that our algorithm outperforms FINCH in terms of both CPU time and the number of nodes accessed. FINCH is denoted as FN in the experiment figures.

Monochromatic RkNN queries

In Fig. 6.1, we vary the value of k and study the effect on the both algorithms. The cost of containment phase is negligible for both of the algorithms. Note that the pruning phase corresponds to the cost of computing the influence zone for our algorithm. The cost of computing the influence zone is even smaller than the pruning cost of FINCH which prunes less area than our algorithm. CPU cost of our algorithm is lower mainly because we use efficient checks to prune the entries of the R-tree and because we do not need to compute the convex hull (in contrast to FINCH that computes a convex polygon to approximate the unpruned area).

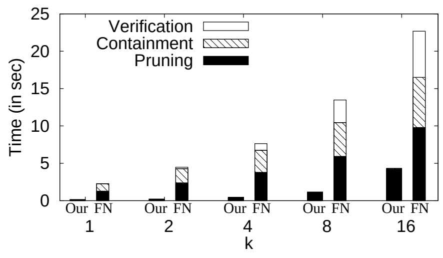


(a) CPU time (b) Nodes accesses

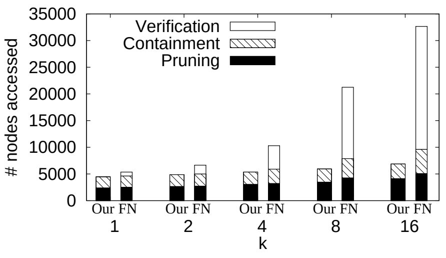
Figure 6.1: Effect of k (monochromatic RkNN)

[1]http://www.cs.fsu.edu/ lifeifei/SpatialDataset.htm

Although we access more facility points to prune the space, the IO cost of computing the influence zone is slightly lower than the pruning cost of FINCH. This is mainly because these facility points are usually found in 1 or 2 leaf nodes which are accessed by FINCH anyway because they are too close to the query. The unpruned area of our algorithm is smaller as compared to FINCH which results in pruning more nodes of the R-tree.



(a) CPU time

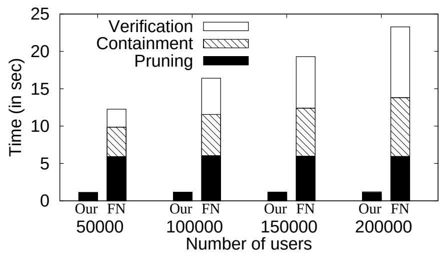


(b) Nodes accesses

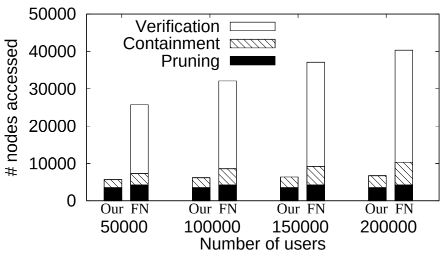
Figure 6.2: Effect of k (bichromatic RkNN)

Bichromatic RkNN queries

Fig. 6.2 studies the effect of k on the cost of bichromatic RkNN queries. The CPU time taken by containment phase of our algorithm is much smaller as compared to FINCH. This is mainly because i) the unpruned area of our algorithm is smaller and ii) we use efficient containment checking to prune the entries and the objects. IO cost of the containment phase is also smaller for our algorithm because the unpruned area of our algorithm is smaller. Our algorithm does not require the verification. On the other hand, FINCH consumes significant amount of CPU time and IOs in the verification phase.



(a) CPU time



(b) Nodes accesses

Figure 6.3: Effect of number of users

Fig. 6.3 studies the effect of the number of the users on both of algorithms. The set of facilities corresponds to the real dataset and the locations of the users follow normal distribution. Our algorithm scales much better. On the other hand, the cost of FINCH degrades with the increase in the number of users because a larger number of users are within the unpruned area and require verification.

In Fig. 6.4, we study the effect of the number of the facilities. The set of the users correspond to the real dataset and the locations of the facilities follow normal distribution. Both of the algorithms are not significantly affected by the increase in the number of the facilities and our algorithm performs significantly better than FINCH.

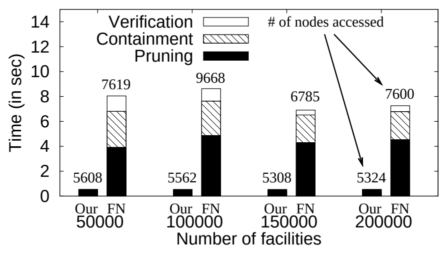


Figure 6.4: number of facilities

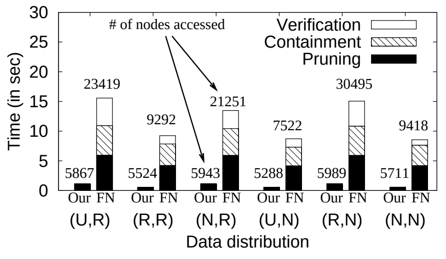


Figure 6.5: Data distribution

Fig. 6.5 studies the effect of the data distribution on both of the algorithms. The data distributions of the facilities and the users are shown in the form ($Dist_1$,$Dist_2$) where $Dist_1$ and $Dist_2$ correspond to the data distribution of the facilities and the users, respectively. U, R and N correspond to Uniform, Real and Normal distributions, respectively. For example, (U,R) corresponds to the case where the facilities follow uniform distribution and the users correspond to the real dataset. Each dataset contains around 88,000 objects. Our algorithm outperforms FINCH both in terms of CPU time and the number of nodes accessed for all of the data distributions.

Fig. 6.6 studies the effect of the buffer size on both of the algorithms. As the pruning and the containment phases do not visit a node twice, our algorithm is not affected by the buffer size. FINCH issues multiple range queries to verify the candidate objects. For this reason, the cost of its verification phase depends on the buffer size. Note that FINCH performs worse than our algorithm even when it uses large buffer size. Number of nodes accessed by FINCH is around 194,000 and 61,000 when the buffer size is 2 and 5, respectively.

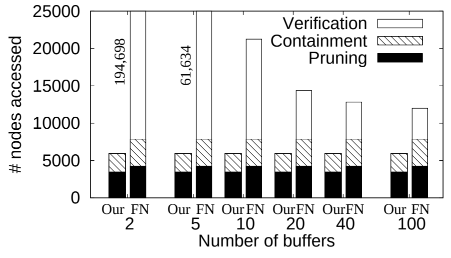


Figure 6.6: Buffer size

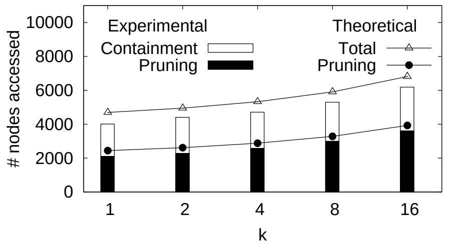


Figure 6.7: IO cost

Verification of theoretical analysis

In Fig. 6.7 and Fig. 6.8, we vary k and verify the threoretical analysis presented in Section 5. In all three experiments, we run bichromatic RkNN queries on uniform datasets consisting of 100,000 facilities and the same number of users.

In Fig. 6.7, we compare the experimental value of total number of nodes accessed with the theroetical value. Recall that the pruning phase of our algorithm corresponds to the computation of the influence zone. Fig. 6.7 shows the accuracy of our theoretical analysis of the IO cost of computing the influence zone and the total cost of our RkNN algorithm.

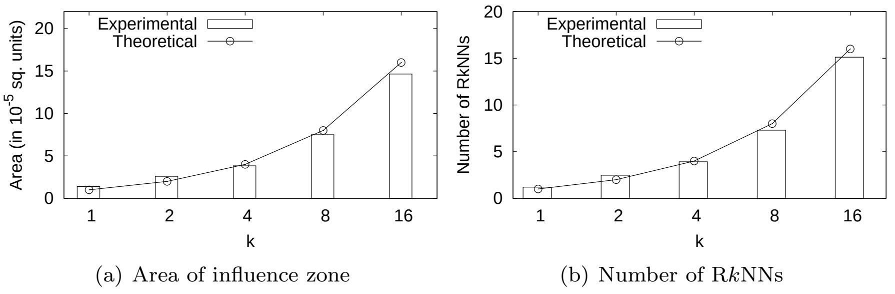


Figure 6.8: Theoretical Analysis

In Fig. 6.8(a) and Fig. 6.8(b), we vary k and verify our theoretical analysis of the area of the influence zone and the number of RkNNs, respectively. It can be seen that the theoretical results are close to the experimental results and follow the trend.

6.2 Continuous Monitoring of RkNN

As mentioned earlier, the problem addressed by the influence zone based algorithm is a special case of the continuous RkNN queries. Hence, it is not fair to use the existing best known algorithms without making any obvious changes that improve the performance. As stated earlier in Section 2.2, Lazy Updates [9] is the best known algorithm for continuous monitoring of RkNN queries (even for this special case, we find that it outperforms other algorithms after necessary changes are made to all the existing algorithms). Hence, we compare our algorithm with Lazy Updates.

To conduct a fair evaluation, we set the size of the *safe region* for the Lazy Updates algorithm to zero. This is because the facilities do not move and the safe regions will not be useful in this case. We tested different possible sizes of the safe region and confirmed that this is the best possible setting for Lazy Updates for this special case of the continuous RkNN query.

Parameter	**Range**
Number of users ($\times$1000)	40, 60, 80, **100**, 120
Number of facilities ($\times$1000)	40, 60, 80, **100**, 120
Number of queries	100, 300, **500**, 700, 1000
k	1, 2, 4, **8**, 16
Speed of objects (users) in km/hr	40, 60, **80**, 100, 120
Mobility of objects (users) in %	5, 20, 40, 60, **80**, 100

Table 6.1: System Parameters

Our experiment settings are similar to the settings used in [9] by Lazy Updates. More specifically, we use Brinkhoff generator [21] to generate the users moving on the road map of Texas (data universe is appx. 1000Km$\times$1000Km). The facilities are randomly generated points in the same data universe. Table 6.1 shows the parameters used in our experiments and the default values are shown in bold.

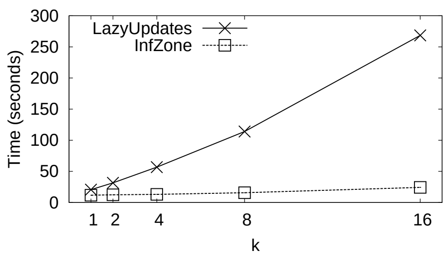


Figure 6.9: Effect of k

The locations of the users are reported to the server after every one second (i.e., timestamp length is one second). The mobility of the objects refers to the percentage of the objects that report location updates at a given timestamp. In accordance with [9], the grid cardinality of both of the algorithms is set to 64×64. Each query is monitored for 5 minutes (300 timestamps) and the total time taken by all the queries is reported.

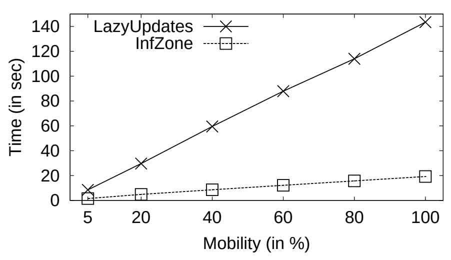


Figure 6.10: Mobility

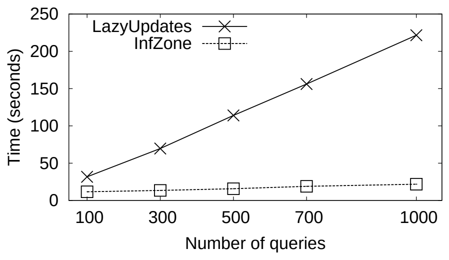


Figure 6.11: # of queries

In Fig. 6.9, 6.10, 6.11, 6.12 and 6.13, we study the effect of k, the data mobility, the number of the queries, the number of the users and the number of the facilities, respectively. Influence zone based algorithm is shown as *InfZone*. Clearly, the influence zone based algorithm outperforms Lazy Updates for all the settings and scales better. In Fig. 6.13, both of the algorithms perform better as the number of facilities increases. This is because the unpruned area becomes smaller when the number of facilities is large. Hence, a smaller area is to be monitored by both the algorithms and it results in lower cost.

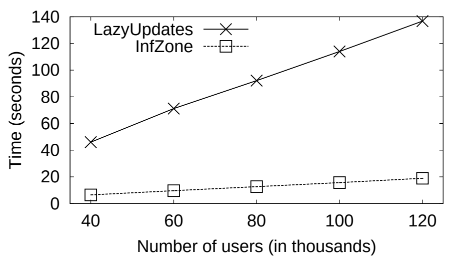


Figure 6.12: # of users

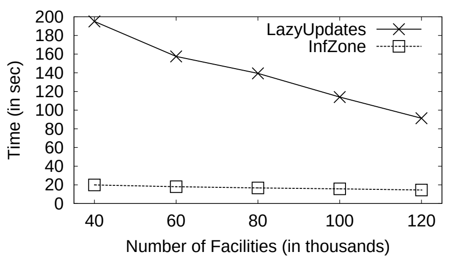


Figure 6.13: # of facilities

7 Conclusion

In this paper, we introduce the concept of an influence zone which does not only have applications in target marketing and market analysis but can also be used to answer snapshot and continuous RkNN queries. We present detailed theoretical analysis to study different aspects of the problem. Extensive experiment results verify the theoretical analysis and demonstrate that influence zone based algorithm outperforms existing algorithms.

Bibliography

[1] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *SIGMOD*, 2000.

[2] C. Yang and K.-I. Lin, "An index structure for efficient reverse nearest neighbor queries," in *ICDE*, 2001.

[3] K.-I. Lin, M. Nolen, and C. Yang, "Applying bulk insertion techniques for dynamic reverse nearest neighbor problems," *IDEAS*, 2003.

[4] I. Stanoi, D. Agrawal, and A. E. Abbadi, "Reverse nearest neighbor queries for dynamic databases," in *ACM SIGMOD Workshop*, 2000, pp. 44–53.

[5] Y. Tao, D. Papadias, and X. Lian, "Reverse knn search in arbitrary dimensionality," in *VLDB*, 2004.

[6] W. Wu, F. Yang, C. Y. Chan, and K.-L. Tan, "Finch: Evaluating reverse k-nearest-neighbor queries on location data," in *VLDB*, 2008.

[7] I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi, "Discovery of influence sets in frequently updated databases," in *VLDB*, 2001, pp. 99–108.

[8] 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.

[9] M. A. Cheema, X. Lin, Y. Zhang, W. Wang, and W. Zhang, "Lazy updates: An efficient technique to continuously monitoring reverse knn," *PVLDB*, 2009.

[10] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries," in *SIGMOD Conference*, 2003, pp. 443–454.

[11] M. Hasan, M. A. Cheema, X. Lin, and Y. Zhang, "Efficient construction of safe regions for moving knn queries over dynamic datasets," in *SSTD*, 2009.

[12] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis, "Nearest neighbor and reverse nearest neighbor queries for moving objects," in *IDEAS*, 2002.

[13] T. Xia and D. Zhang, "Continuous reverse nearest neighbor monitoring," in *ICDE*, 2006, p. 77.

[14] W. Wu, F. Yang, C. Y. Chan, and K.-L. Tan, “Continuous reverse k-nearest-neighbor monitoring,” in *MDM*, 2008.

[15] A. Guttman, “R-trees: A dynamic index structure for spatial searching,” in *SIGMOD Conference*, 1984.

[16] F. P. Preparata and M. I. Shamos, *Computational Geometry An Introduction.* Springer, 1985.

[17] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Wiley, 1999.

[18] M. A. Cheema, L. Brankovic, X. Lin, W. Zhang, and W. Wang, “Multi-guarded safe zone: An effective technique to monitor moving circular range queries,” in *ICDE*, 2010, pp. 189–200.

[19] Y. Theodoridis, E. Stefanakis, and T. K. Sellis, “Efficient cost models for spatial queries using r-trees,” *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 1, pp. 19–32, 2000.

[20] I. Kamel and C. Faloutsos, “On packing r-trees,” in *CIKM*, 1993, pp. 490–499.

[21] T. Brinkhoff, “A framework for generating network-based moving objects,” *GeoInformatica*, 2002.