SLICE: Reviving Regions-Based Pruning for Reverse k Nearest Neighbors Queries

Shiyu Yang[†], Muhammad Aamir Cheema[‡†], Xuemin Lin[†§], Ying Zhang[†]
[†]*School of Computer Science and Engineering, The University of New South Wales, Australia*
[‡]*Clayton School of Information Technology, Monash University, Australia*
[§]*Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China*
yangs@cse.unsw.edu.au, aamir.cheema@monash.edu, {lxue, yingz}@cse.unsw.edu.au

***Abstract*—Given a set of facilities and a set of users, a reverse k nearest neighbors (RkNN) query q returns every user for which the query facility is one of the k-closest facilities. Due to its importance, RkNN query has received significant research attention in the past few years. Almost all of the existing techniques adopt a *pruning-and-verification* framework. Regions-based pruning and half-space pruning are the two most notable pruning strategies. The half-space based approach prunes a larger area and is generally believed to be superior. Influenced by this perception, almost all existing RkNN algorithms utilize and improve the half-space pruning strategy. We observe the weaknesses and strengths of both strategies and discover that the regions-based pruning has certain strengths that have not been exploited in the past. Motivated by this, we present a new RkNN algorithm called SLICE that utilizes the strength of regions-based pruning and overcomes its limitations. Our extensive experimental study on synthetic and real data sets demonstrate that SLICE is significantly more efficient than the existing algorithms. We also provide a detailed theoretical analysis to analyze various aspects of our algorithm such as I/O cost, the unpruned area, and the cost of its verification phase etc. The experimental study validates our theoretical analysis.**

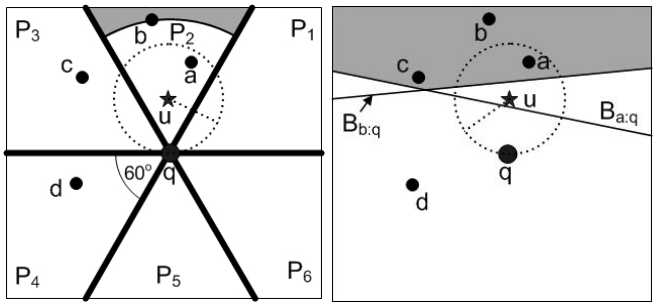
I. INTRODUCTION

A reverse k nearest neighbors (RkNN) query finds every data point p for which the query point q is one of its k-closest points. The RkNN queries for which $k = 1$ are denoted as RNN queries. Consider the example of a shopping market. The residents for which this market is one of the k-closest markets are the potential customers of this market. In this paper, the objects that provide a facility or a service (e.g., markets, fuel stations) are called *facilities* and the objects (e.g., residents, drivers) that use the facilities are called *users*. In this context, a RkNN query returns every user u for which the query facility q is one of its k-closest facilities. Since q is close to such users, a RkNN query returns the users that are likely to use the query facility q.

RkNN query has received significant research attention [1], [2], [3], [4], [5], [6], [7], [8], [9] ever since it was introduced in [10]. The existing techniques adopt a *pruning* and *verification* framework. In the pruning phase, the set of facilities is used to prune the area that cannot contain any RkNN. In the verification phase, the users that lie in the unpruned space are retrieved. These users are potential RkNNs of the query and are called candidates. Each candidate u is then verified by checking whether q is one of its k closest facilities or not. This is usually done by issuing a range query centered at u with radius $dist(u, q)$ and checking whether the range contains less than k facilities or not. Next, we briefly describe two most notable pruning strategies and then highlight the weaknesses and advantages of each strategy.

Regions-based pruning [3]. In this approach, the space around q is divided into six equally sized regions, each of angle range $60°$. In Fig. 1(a), q and four facilities a to d are shown. Six regions (P_1 to P_6) are also shown. For a partition P_i, let d_k be the distance between q and its k-th nearest facility in P_i. Assuming $k = 2$, $d_k = dist(b, q)$ for the partition P_2 in Fig. 1(a). It has been shown that a user u that lies in P_i at a distance from q greater than d_k can be pruned, i.e., the shaded area in Fig. 1(a) can be pruned.

Half-space pruning [11], [9]. Let $B_{a:q}$ denote the perpendicular bisector between a facility a and q (see Fig. 1(b)). The bisector divides the space into two halves. We use $H_{a:q}$ to denote the half-space that contains a and $H_{q:a}$ to denote the half-space that contains q. For a point p that lies in $H_{a:q}$, $dist(p, a) < dist(p, q)$. In other words, p cannot be the RNN of q and we say that the half-space $H_{a:q}$ prunes the point p. Clearly, a point that is pruned by at least k half-spaces cannot be the RkNN of q. Fig. 1(b) shows two bisectors $B_{a:q}$ and $B_{b:q}$. Assuming $k = 2$, the shaded area can be pruned because every point in the shaded area is pruned by the two half-spaces $H_{a:q}$ and $H_{b:q}$.



(a) Six-regions pruning (b) Half-space pruning

Fig. 1. Illustration of pruning strategies ($k = 2$)

The advantage of half-space pruning is that it prunes a much larger area as compared to the area pruned by six-regions (compare the shaded area in Fig. 1(a) and Fig. 1(b)). The advantage of the regions-based approach is that the cost of

Operation	Six-regions	InfZone	SLICE
Prune a facility	$O(1)$	$O(m)$	$O(t)$
Prune the space	$O(m \log k)$	$O(km^2)$	$O(tm \log m)$
Prune a user	$O(1)$	$O(\log m)$	$O(1)$
Verify candidate	range query	$O(\log m)$	$O(m)$
Expected #cand.	$\frac{6k\|U\|}{\|F\|}$	$\frac{k\|U\|}{\|F\|}$	$< \frac{3.1k\|U\|}{\|F\|}$

TABLE I
COMPARISON OF COMPUTATIONAL COMPLEXITIES

checking whether a point p can be pruned or not is $O(1)$. Specifically, to check whether a point p is pruned by six-regions pruning, we only need to compare $dist(p, q)$ with d_k. On the other hand, half-space pruning is significantly more expensive. For instance, checking whether p can be pruned requires to check whether p lies in at least k half-spaces or not. The cost is $O(m)$ where m is the number of facilities considered for pruning.

Table I compares six-regions approach [3] and the state-of-the-art half-space pruning based approach called InfZone [9], [12]. InfZone not only improves the pruning cost of a user from $O(m)$ to $O(\log m)$ but also significantly improves the verification cost. However, note that the pruning cost of InfZone is still significantly higher than that of six-regions. The pruning cost of InfZone for a facility is $O(m)$ which is higher than the pruning cost of a user $O(\log m)$. This is because InfZone utilizes a different pruning criterion for facilities and prunes only the facilities that are unable to further prune the unpruned area.

As shown in Table I, the bottleneck of the six-regions approach is its verification phase. Six-regions approach verifies a candidate u by issuing a range query centered at u with radius $dist(u, q)$ (see the dotted circle Fig. 1). This results in not only additional computational cost but also I/O cost because the index is to be accessed to verify each candidate. Table I also demonstrates the expected number of candidates (i.e., the users that cannot be pruned) where $|F|$ denotes the total number of facilities and $|U|$ denotes the total number of users. Hence, the verification phase of six-regions approach is expected to issue $\frac{6k|U|}{|F|}$ range queries that dominates the total cost of the algorithm. Note that the expected number of candidates for six-regions is 6 times the expected number of candidates for InfZone. This is due to the poor pruning power of six-regions approach.

Several techniques have been proposed to address the limitations of half-space pruning (e.g., FINCH [6], InfZone [9]). Surprisingly, there is no work that tries to utilize the strength of regions-based pruning (i.e., cheap pruning) and addresses its limitations (low pruning power and expensive verification). In this paper, we propose an algorithm called SLICE that addresses the limitations of regions-based approach and utilizes its strength. Specifically, SLICE uses a more powerful and flexible pruning approach that prunes a much larger area as compared to six-regions with almost similar computational complexity. Furthermore, it significantly improves the verification phase by computing $sigList$ for each partition P_i. The $sigList$ of a partition P_i consists of a set of facilities that is sufficient to verify every candidate $u \in P_i$. Hence, a candidate can be verified by accessing $sigList$ instead of issuing a range query.

Table I compares the cost of SLICE with six-regions and InfZone. t denotes the number of partitions used by our approach (typically 6 to 12). We remark that, in the worst case, m may be as large as $|F|$ where $|F|$ is the total number of facilities. Note that the pruning cost of our algorithm is significantly smaller than InfZone and is quite close to six-regions approach.

To verify a user, SLICE accesses the objects in $sigList$ which contains $O(m)$ facilities in the worst case. However, our theoretical analysis demonstrates that the expected size of $sigList$ is $O(k)$. Hence, the expected cost of the verification is $O(k)$. As we show later, this also implies that the expected cost of "*prune the space*" operation for SLICE is $O(tm \log k)$. Also, note that a majority of the users are pruned by "*prune a user*" operation and the number of candidates that require verification is usually small, e.g., when $|U| = |F|$, the expected number of candidates is less than $3.1k$. Hence, the dominating cost of InfZone and SLICE is the pruning phase for which SLICE is significantly more efficient.

A reader may assume that the verification phase of InfZone is more efficient than that of SLICE which is not necessarily true. The verification phase of the two algorithms consist of two major operations: 1) pruning the user entries; 2) verifying the unpruned users, i.e., candidates. As shown in Table I, SLICE is more efficient for the first operation whereas InfZone is faster for the second operation. Hence, the total verification cost depends on the ratio of the numbers of the two operations during the verification phase. Our experimental study demonstrates that the verificaiton phase of SLICE is slower than that of InfZone only for larger k. This is because the number of candidates increases as the value of k increases.

We remark that SLICE aims at reducing the overall computational cost by compromising on a slightly higher I/O cost than InfZone [9]. Later in Section V, we demonstrate that the I/O cost of each algorithm is negligible when compared with its CPU cost and the overall cost of SLICE is much lower than InfZone. Nevertheless, InfZone must be preferred if the focus is to minimize the number of I/Os.

Below, we summarize our contributions in this paper.

- Influenced by the perception that regions-based pruning is always inferior to half-space pruning, almost all existing techniques use half-space pruning to solve RkNN queries [11], [6], [9], [13] and its variants (e.g., continuous RkNN queries [14], [8], [15], probabilistic RkNN queries [16], [17], and incremental RkNN queries [13], [18] etc). In this paper, we address the limitations of six-regions approach and demonstrate that the regions-based pruning is not necessarily inferior to half-space pruning. We propose an algorithm based on regions-based pruning that significantly outperforms state-of-the-art algorithm in terms of running time. This also indicates that the improved regions-based approach may be useful to solve other variants of RkNN queries.

- We provide a detailed theoretical analysis to analyse the cost of our algorithm. Specifically, we theoretically analyse the I/O cost, number of candidates, the area pruned, and the expected computational complexity of the verification phase. Our experimental results validate our theoretical analysis.
- Our experimental study on real and synthetic data sets demonstrates that SLICE is significantly more efficient than the existing algorithms for all settings (except when $k = 1$). We have also released the source codes, data sets and scripts to assist the readers in conducting more experiments or reproducing the experimental results.

The rest of the paper is organized as follows. In Section II, we formally define the problem and introduce the related work. Our techniques are presented in Section III. Section IV provides a detailed theoretical analysis. Experimental evaluation is presented in Section V followed by conclusion in Section VI.

II. PRELIMINARIES

First, we formally define the problem in Section II-A. Then, we present the related work in Section II-B.

A. Problem Definition

RkNN queries are classified [10] into *bichromatic RkNN queries* and *monochromatic RkNN queries*.

Bichromatic RkNN Queries. Consider a set of facilities F and a set of users U. Given a query $q \in F$, a bichromatic RkNN query returns every user $u \in U$ for which q is one of its k-closest facilities.

Monochromatic RkNN Queries. Given a set of facilities F and a query $q \in F$, a monochromatic RkNN query returns every facility $f \in F$ for which q is one of its k-closest facilities.

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.

Since most of the applications of RkNN queries are in location-based services, like existing techniques [9], [6], [3], the focus of this paper is on two dimensional location data. Our proposed approach can be applied to answer both bichromatic and monochromatic RkNN queries. However, for ease of presentation, we limit our discussion to bichromatic RkNN queries unless specifically mentioned. Later, in Section III-C3, we show that our techniques can be easily applied for monochromatic RkNN queries.

B. Related work

RkNN queries have been extensively studied considering different settings such as continuous RkNN queries [19], [20], [8], [14], probabilistic RkNN queries [16], [21], [22], [17], RkNN queries on graphs [23], [24], [25], metric spaces [23] and adhoc spaces [26] etc. Since the focus of this paper is on static queries in Euclidean space, we provide a brief overview of the techniques to solve RkNN queries in Euclidean space.

Korn *et al.* [10] 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 [1], [2].

Now, we briefly describe the existing techniques that do not require pre-computation. These techniques have two phases namely *pruning* and *verification*. In the pruning phase, the space that cannot contain any RkNN is pruned by using the set of facilities. In the verification phase, the users that lie within the unpruned space are retrieved. These are the possible RkNNs and are called the candidates. Most of the existing techniques verify a candidate by issuing a range query and checking if q is one of its k nearest facilities or not.

In the past few years, RkNNs have been extensively studied under different settings (e.g., see [4], [5], [7], [8], [8], [15] and references therein). Next, we briefly describe some of the most related techniques namely six-regions [3], TPL [11], FINCH [6] and InfZone [9]. We present only the pruning strategy of these techniques. The verification phase of these techniques (except of InfZone) is similar. Specifically, for each candidate u, a range query centered at u and range set as $dist(u, q)$ is issued and the user is returned as a RkNN if the range contains less than k facilities. The verification phase of InfZone will be discussed later.

Six-regions. First technique that does not need any pre-computation was proposed by Stanoi *et al.* [3]. They solve RkNN queries by partitioning the whole space centred at the query q into six equal regions of $60°$ each (P_1 to P_6 in Fig. 2). As stated in Section I, the k-th nearest facility of q in each region defines the area that can be pruned. In other words, assume that d_k is the distance between q and its k-th nearest facility in a region P_i. Then any user u that lies in P_i and lies at a distance greater than d_k from q cannot be the RkNN of q. This is because, for such user u, $dist(u, f) < dist(u, q)$ for every f where f is one of the k-nearest facilities of q in the region P_i.

Fig. 2 shows a RkNN ($k = 2$) query q and four facilities a to d. In region P_2, b is the second nearest facility of q and the shaded area can be pruned, i.e., only the users that lie in the white area can be the RkNNs. A user u that lies in the shaded area cannot be RkNN because it is guaranteed to be closer to a and b than q.

TPL. Tao *et al.* [11] propose TPL that prunes the space using the half-space pruning (as described in Section I). Recall that a point p that is pruned by at least k half-spaces cannot be the RkNN. TPL algorithm iteratively accesses the nearest facilities in the unpruned area. Each accessed facility f is used to prune the space. The pruning phase completes when there does not exist any facility in the unpruned space. Fig. 3 shows the example where the bisectors between q and a, c and d are drawn ($B_{a:q}$, $B_{c:q}$ and $B_{d:q}$, respectively). If $k = 2$, the shaded area can be pruned because every point in it lies in at least two half-spaces. Note that the facility b lies in the pruned area so its bisector is not used for pruning.

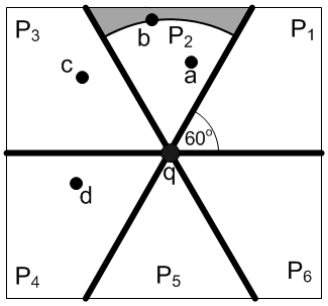


Fig. 2. Six-regions [3]

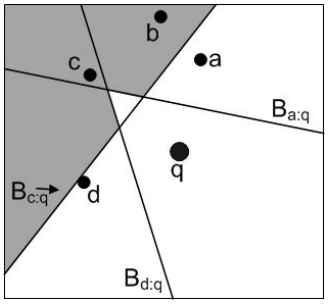


Fig. 3. TPL [11]

Let m be the number of facilities for which the bisectors are considered for pruning. 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 facilities by their Hilbert values. Then, only the combinations of k consecutive facility points are considered to prune the space (total m combinations). The cost to prune an entry using this strategy is $O(km)$.

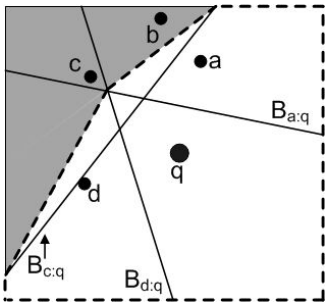


Fig. 4. FINCH [6]

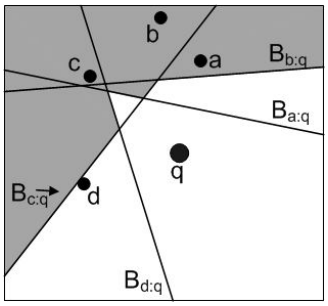


Fig. 5. InfZone [9]

FINCH. 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] propose 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. For example, in Fig. 3, the unpruned area is the white area. FINCH approximates this unpruned area by a convex polygon (the white area in Fig. 4 with boundary shown in broken lines). Any point that lies outside this polygon can be pruned, i.e., the shaded area of Fig. 4 can be pruned. Clearly, pruning of FINCH is more efficient than TPL because containment can be done in logarithmic time for convex polygons. Hence, a point can be pruned in $O(\log\ m)$. Unfortunately, the cost of computing the convex polygon that approximates the unpruned area is $O(m^3)$ where m is the number of facilities used for pruning.

InfZone. The verification phase of six-regions, TPL and FINCH is quite expensive because it requires issuing a range query for each candidate. Cheema *et al.* [9] propose InfZone which uses the concept of *influence zone* to significantly improve the verification phase. Influence zone is the area such that a point p is a RkNN of q if and only if p lies inside this area. It was shown that the influence zone is a star-shaped polygon [27] and the point containment can be done in logarithmic time to the number of edges of the star-shaped polygons, i.e., the cost to prune a point is $O(\log\ m)$. Note that InfZone does not require the verification of a candidate because every user u that lies in the influence zone is guaranteed to be RkNN. In other words, the verification cost is $O(\log\ m)$.

The influence zone corresponds to the unpruned area when the bisectors of *all* the facilities have been considered for pruning. For instance, in Fig. 5, the bisectors between q and all facilities are drawn ($B_{a:q}$, $B_{b:q}$, $B_{c:q}$, and $B_{d:q}$). The shaded area can be pruned and the white area is the influence zone. Recall that FINCH and TPL did not consider $B_{b:q}$ because when the bisectors of a, c and d are considered the facility b lies in the pruned area and is ignored. Cheema *et al.* [9] present several properties to reduce the number of facilities that must be considered in order to correctly compute the influence zone. The cost of computing the influence zone using m facilities is $O(km^2)$ [12].

III. Techniques

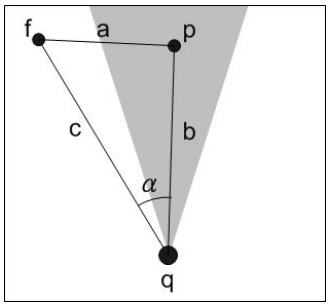
In Section III-A, we present observations to improve the area pruned by regions-based pruning. In Section III-B, we present techniques to be used in the verification phase of our algorithm. Our algorithm is described in Section III-C.

A. Reviving Regions-Based Pruning

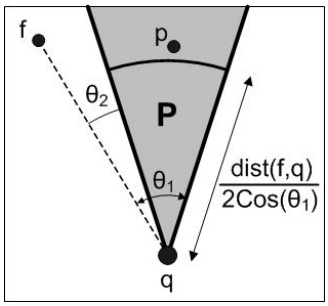
First, we define a few terms and notations (Table II contains a summary).

Definition 1 : **Subtended angle**. Given a query point q, the subtended angle between two points x and y is the angle $\angle xqy$ in the triangle $\triangle xqy$. It is denoted as $angle(x, y)$. If x, q and y lie on the same line then $angle(x, y) = 0°$ or $angle(x, y) = 180°$ depending on the relative positions of x, q and y.

In Fig. 6(a), $angle(f, p) = \alpha$. Note that $angle(x, y) \leq 180°$.



(a) Proof of Lemma 1



(b) Illustration of terms

Fig. 6. Pruning space using a facility f in a partition P

Definition 2 : **Maximum (minimum) subtended angle**. Given a query point q, maximum subtended angle between a point x and a partition P is the maximum subtended

Notation	Definition
q	the query point
P	a partition
$angle(x, y)$	the subtended angle between x and y
$maxAngle(x, P)$	$argmax_{p \in P} angle(x, P)$
$minAngle(x, P)$	$argmin_{p \in P} angle(x, P)$
$r^U_{f:P}$ or r^U	The upper arc of facility f for partition P
$r^L_{f:P}$ or r^L	The lower arc of facility f for partition P
$r_{B:P}$ or r_B	The bounding arc for a partition P

TABLE II
NOTATIONS

angle between x and any point p in the partition P, i.e., $argmax_{p \in P}\ angle(x, p)$. It is denoted as $maxAngle(x, P)$. The minimum subtended angle is defined similarly and is denoted as $minAngle(x, P)$.

In Fig. 6(b), the partition P is shown shaded and $maxAngle(f, P) = \theta_1$ and $minAngle(f, P) = \theta_2$. Since $angle(x, y) \leq 180°$, $minAngle(f, P) < maxAngle(f, P) \leq 180°$. Also, note that $minAngle(f, P) = 0°$ if f lies inside the partition P.

Next, we present a lemma that identifies the area that can be pruned by a facility f. We say a facility f prunes a point p if $dist(f, p) < dist(p, q)$. Note that a point that is pruned by at least k facilities cannot be a RkNN of q.

LEMMA 1 : A facility f prunes every point $p \in P$ for which $dist(p, q) > \frac{dist(f,q)}{2\cos(\theta)}$ where $\theta = maxAngle(f, P)$ and $0° \leq \theta < 90°$.

Proof: Fig. 6(b) shows a point $p \in P$ for which $dist(p, q) > \frac{dist(f,q)}{2\cos(\theta)}$. Consider the triangle obtained by joining f, p and q as shown in Fig 6(a). Let the side lengths of the triangle be denoted as a, b and c and $angle(f, p)$ be denoted as α. To prove the lemma, we need to show that $dist(f, p) < dist(p, q)$, i.e., $a < b$. The side length a can be calculated by using the following equation.

$$a^2 = b^2 + c^2 - 2bc \cdot \cos(\alpha)$$

To prove $a < b$, we need to show that $c^2 - 2bc \cdot \cos(\alpha) < 0$. Since $b > (\frac{dist(f,q)}{2\cos(\theta)} = \frac{c}{2\cos(\theta)})$, the following inequality holds.

$$c^2 - 2bc \cdot \cos(\alpha) < c^2 - 2\frac{c}{2\cos(\theta)} c \cdot \cos(\alpha) = c^2(1 - \frac{\cos(\alpha)}{\cos(\theta)})$$

Since $\alpha \leq \theta$ and $0 \leq \theta < 90°$, $\frac{\cos(\alpha)}{\cos(\theta)} \geq 1$. Hence, $c^2 - 2bc \cdot \cos(\alpha) < 0$. ■

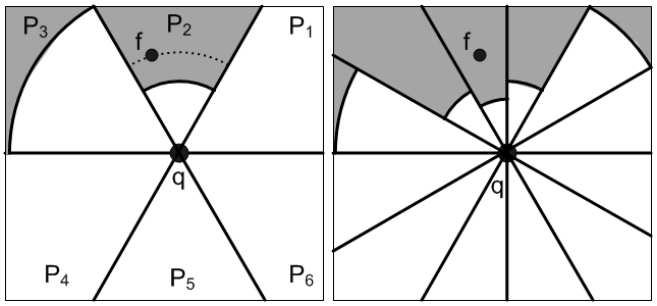
For the ease of presentation, we define upper arc (the lower arc will be defined later).

DEFINITION 3 : **Upper arc**. Upper arc of a facility f w.r.t. a partition P is the arc centered at q with radius $dist(f, q)/2\cos(\theta)$ where $\theta = maxAngle(f, P)$ and $0° \leq \theta < 90°$. The radius of the upper arc is denoted as $r^U_{f:P}$ (or simply r^U when the facility f and the partition P are clear by context). If $\theta \geq 90°$, $r^U_{f:P} = \infty$.

Fig. 6(b) shows the upper arc of f for the partition P. We say that a point lies outside (resp. inside) an arc of radius r if its distance from the center of the arc is greater (resp. smaller) than r. According to Lemma 1, a facility prunes area outside the upper arc of f for every partition P for which $maxAngle(f, P) < 90°$. Fig. 7 shows the area pruned (shown shaded) by a facility f for different partitions. This pruning approach is superior to six-regions approach in the following ways.

1. In six-regions approach, a facility f prunes space in only the partition in which f lies. In contrast, our proposed approach prunes space in every partition P for which $maxAngle(f, P) < 90°$. For instance, in Fig. 7(a), our proposed approach prunes space in partitions P_2 and P_3 (the shaded area) whereas the six-regions approach prunes only the space in the partition P_2.

2. Even for the partition P_i that contains f, our approach is superior because it prunes at least as much area of P_i as pruned by six-regions approach. In Fig. 7(a), the area of P_2 pruned by our approach is shown shaded whereas the area of P_2 pruned by six-regions approach is bounded by the dotted arc. Note that when $maxAngle(f, P) = 60°$, the area pruned by our approach for the partition that contains f is the same as the area pruned by six-regions approach because $dist(f, q)/2\cos(60°) = dist(f, q)$.



(a) Prunes more area (b) Effect of number of partitions

Fig. 7. Comparison with six-regions approach

3. Six-regions approach restricts the division of the space into strictly six regions each of equal size. In contrast, our approach allows arbitrary number of partitions where each partition may have a different size[1]. In Fig. 7(b), we divide the space into 12 equally sized partitions and the area pruned by the facility f is shown shaded. Note that the area pruned by our approach becomes larger as the number of partitions increases (compare the shaded area in Fig. 7(a) and Fig. 7(b)). Although increasing the number of partitions increases the pruned area, it results in computational overhead because more partitions are to be processed for each facility. In Section IV, we present a detailed theoretical analysis to study the effect of the number of partitions.

DEFINITION 4 : **Bounding arc**. The k-th smallest upper arc of a partition P is called its bounding arc. The radius of the

[1]Although the partitions with different sizes can be used, in this paper, we use equally sized partitions so that the partition that contains a point p can be identified in $O(1)$.

bounding arc of a partition P is denoted as $r_{B:P}$ (or simply r_B when clear by context). If the partition contains less than k upper arcs, $r_{B:P} = \infty$.

Note that a point p that is pruned by at least k facilities cannot be a RkNN. In other words, the area that is pruned by at least k upper arcs cannot contain any RkNN. Hence, a point $p \in P$ cannot be a RkNN if it lies outside the bounding arc, i.e., $dist(p, q) > r_{B:P}$.

Fig. 8 shows the area pruned by two facilities f_1 and f_2. The upper arcs of f_1 are shown using solid lines and the upper arcs of f_2 are shown using broken lines. The bounding arc r_B for one of the partitions is also shown (assuming $k = 2$). Clearly, the shaded area cannot contain a RkNN ($k = 2$) and can be pruned.

B. Improving Verification Phase

As stated earlier, most of the existing techniques issue a range query to verify a candidate u and check whether the range contains less than k facilities or not. This requires accessing the facility R-tree for each such user and incurs unnecessary I/O and CPU cost. In this section, we present several observations that help to significantly improve the verification phase. First, we define *significant* facilities.

DEFINITION 5 : **Significant facility**. A facility f is called a significant facility of a partition P if it prunes at least one point $p \in P$ lying inside the bounding arc of P. We remark that p is an arbitrary point in P and is not necessarily a data object.

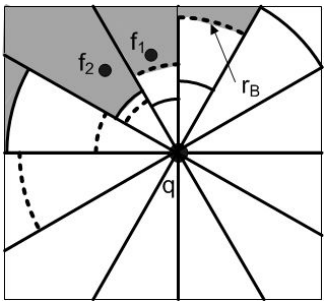


Fig. 8. The shaded area is pruned

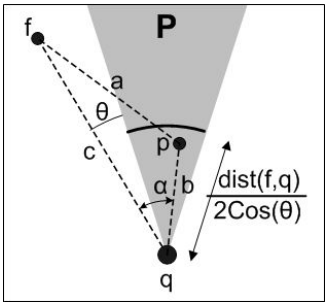


Fig. 9. Lower arc

A facility that is not significant for a partition P is called an insignificant facility for P. During the pruning phase, for each partition P, we identify the set of its *significant* facilities called $sigList$ of P. Note that a user $u \in P$ that lies outside the bounding arc of P is pruned as stated in the previous section. Every other user $u \in P$ can be verified by accessing only the facilities in $sigList$ of P because $sigList$ contains every facility f that can possibly prune u. This not only reduces the I/O cost because accessing the R-tree is not required but it also improves the computation cost because the $sigList$ is kept sorted in a specific order to speed up the verification (as we describe later). We also demonstrate that the expected size of $sigList$ is $O(k)$. Hence, the verification cost of a candidate is expected to be $O(k)$.

Lemma 4 and Lemma 5 demonstrate how to identify the insignificant facilities for a partition P. Before we formally present the lemmas, we present a few other lemmas that do not only lead to Lemma 4 and Lemma 5 but also help in other aspects of the verification phase.

LEMMA 2 : A facility f cannot prune a point $p \in P$ for which $dist(p, q) \leq \frac{dist(f,q)}{2\cos(\theta)}$ where $\theta = minAngle(f, P)$ and $0° \leq \theta < 90°$.

Proof: Fig. 9 shows a point $p \in P$ for which $dist(p, q) < \frac{dist(f,q)}{2\cos(\theta)}$. Consider the triangle obtained by joining f, p and q. Let the side lengths of the triangle be denoted as $dist(f, p) = a$, $dist(p, q) = b$ and $dist(f, q) = c$ and $angle(f, p)$ be denoted as α as shown in Fig. 9. To prove that f does not prune p, we show that $dist(f, p) \geq dist(p, q)$, i.e., $a \geq b$. The side length a can be computed by the following equation.

$$a^2 = b^2 + c^2 - 2bc \cdot \cos(\alpha)$$

To prove $a \geq b$, we need to show that $c^2 - 2bc \cdot \cos(\alpha) \geq 0$. Since $dist(p, q)$ (i.e., b) is at most $\frac{c}{2\cos(\theta)}$, $c^2 - 2bc \cdot \cos(\alpha)$ is at least $c^2 - \frac{2c^2\cos(\alpha)}{2\cos(\theta)} = c^2(1 - \frac{\cos(\alpha)}{\cos(\theta)})$. Since $\alpha \geq \theta$ and $0° \leq \theta < 90°$, $\frac{\cos(\alpha)}{\cos(\theta)} \leq 1$. Hence, $c^2 - 2bc \cdot \cos(\alpha) \geq 0$ which completes the proof. ■

LEMMA 3 : A facility f cannot prune any point $p \in P$ if $minAngle(f, P) \geq 90°$.

Proof: Consider the example of Fig. 9 and assume that $minAngle(f, P) \geq 90°$. Since $minAngle(f, P) \geq 90°$, $\alpha \geq 90°$. The side opposite to α is the largest side of the triangle $\triangle fpq$ because α is the largest angle of the triangle. This implies that $dist(f, p) > dist(p, q)$. Hence, f cannot prune the point p. ■

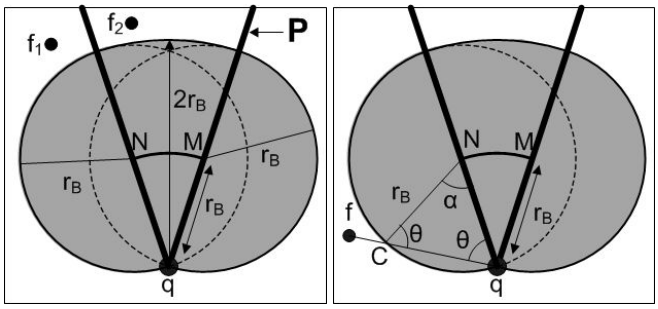
DEFINITION 6 : **Lower arc**. Lower arc of a facility f w.r.t. a partition P is the arc centered at q with radius $dist(f, q)/2\cos(\theta)$ where $\theta = minAngle(f, P)$. The lower arc is denoted as $r^L_{f:P}$ (or simply r^L when the facility f and the partition P are clear by context).

According to Lemma 2, a facility f cannot prune any point $p \in P$ that lies inside the lower arc. Next, we show how to identify insignificant facilities. Consider a partition P as shown in Fig. 10(a) (shown with thick boundaries). Its bounding arc is also shown with radius r_B. Let M and N be the points where this arc intersects the boundary of the partition P (see Fig. 10). The next two lemmas show that a facility that lies outside the shaded area cannot be a significant facility.

LEMMA 4 : A facility $f \in P$ cannot be a significant facility of P if $dist(f, q) > 2r_B$.

Proof: Since f lies in P, $minAngle(f, P) = 0$. The radius of its lower arc is $r^L = dist(f, q)/2\cos(0) = dist(f, q)/2$. Since $dist(f, q) > 2r_B$, $r^L > r_B$. According to Lemma 2, f cannot prune any point in $p \in P$ that lies inside its lower arc r^L. Since $r^L > r_B$, f cannot prune any point that lies inside the bounding arc. Hence, f is not a significant facility. ■

In Fig. 10(a), f_2 is not a significant facility. Next, we show that f_1 is also an insignificant facility.



(a) Facilities outside the shaded area are insignificant

(b) Proof of Lemma 5

Fig. 10. Identifying insignificant facilities

LEMMA 5 : A facility $f \notin P$ cannot be a significant facility if $dist(M, f) > r_B$ and $dist(N, f) > r_B$ where M and N are the points where the bounding arc of P intersects the boundary of P (see Fig. 10(b)).

Proof: Fig. 10(b) shows a facility f for which $dist(M, f) > r_B$ and $dist(N, f) > r_B$ (i.e., f lies outside the circles centered at M and N with radius r_B). We prove that the radius of the lower arc of f is not less than the radius of the bounding arc of P, i.e., $r^L \geq r_B$. This implies that f cannot prune any point that lies inside the bounding arc r_B and is an insignificant facility. Let $\theta = minAngle(f, P)$. If $\theta \geq 90°$, the facility is not a significant facility because it cannot prune any point $p \in P$ (according to Lemma 3).

If $\theta < 90°$, the line that joins f and q intersects at least one of the circles of radius r_B centered at M and N (this is because q lies at the boundary of these two circles). Without loss of generality, assume that the line $\overline{fq}$ intersects the circle centered at N at a point C (as shown in Fig. 10(b)). Now consider the triangle $\triangle NCq$. Since $\overline{NC} = \overline{Nq} = r_B$, $\angle NCq = \angle NqC = \theta$. The length of $\overline{Cq}$ can be obtained by the following equation.

$$\overline{Cq}^2 = r_B^2 + r_B^2 - 2r_B^2 \cdot \cos(\alpha) = 2r_B^2(1 - \cos(\alpha))$$

Since $\alpha = 180° - 2\theta$, the following is obtained.

$$\overline{Cq}^2 = 2r_B^2(1 - \cos(180° - \theta)) = 2r_B^2(1 + \cos(2\theta))$$

$$\overline{Cq}^2 = 4r_B^2 \cos^2(\theta)$$

Hence, $\overline{Cq} = 2r_B \cos(\theta)$. Since $dist(f, q) > \overline{Cq}$, $dist(f, q) > 2r_B \cos(\theta)$. Recall that $r^L = dist(f, q)/2\cos(\theta)$ which implies that $r^L > 2r_B Cos(\theta)/2\cos(\theta)$ or $r^L > r_B$. ■

Lemma 4 and Lemma 5 demonstrate that the facilities that lie outside the shaded area of Fig. 10 are not significant facilities and are not required to verify any user $u \in P$. In fact, it can be proved that a facility is significant if and only if it lies in the shaded area of Fig. 10. We omit the proof due to space limitations but it can be obtained using the similar arguments.

C. Algorithm

Our algorithm has two phases: i) pruning; and ii) verification. In the pruning phase, the algorithm prunes the search space using the set of facilities. It also identifies the set of significant facilities that are used later in the verification phase. In the verification phase, the set of users that lie in the unpruned area are identified. These users are then verified as RkNN if there are at most $k - 1$ facilities closer to it than q.

1) Pruning Algorithm: Algorithm 1 describes the pruning phase. The space around q is divided into t equally sized partitions ($t = 12$ in our experiments). Later in Section IV, we provide a detailed theoretical analysis to study the effect of t. The algorithm utilizes a min-heap h which is initialized by inserting the root of the R-tree that indexes the set of facilities. The entries are de-heaped iteratively from the heap. If the de-heaped entry cannot contain any significant facility for *any* partition, we ignore it because it cannot prune space in *any* partition (line 5). Whether an entry e may or may not contain a significant facility can be checked by applying Lemma 4 and 5 for each partition P. Specifically, if e lies completely outside the shaded area shown in Fig. 10, e cannot contain a significant facility for the partition P.

Algorithm 1: Pruning

```
1  Divide the space around q in t equally sized partitions;
2  Insert root of facility R-tree in a min-heap h;
3  while h is not empty do
4      deheap an entry e;
5      if e may contain a significant facility for at least one partition
       then // apply Lemma 4 & 5 for each partition P
6          if e is an intermediate node or leaf then
7              insert every child c in h with key mindist(q, c);
8          else                                  // e is a data object
9              pruneSpace(e)                     //Algorithm 2
```

If e may contain a significant facility, it is processed as follows. If e is an intermediate or leaf node, every child c of e is inserted in the heap with its key set to $mindist(q, c)$ (line 7). The algorithm accesses the entries in ascending order of $mindist(q, c)$ because the facilities that are closer to the query are expected to prune more area. If e is a data object (i.e., a facility), it is used to prune the space by calling Algorithm 2. The algorithm terminates when the heap becomes empty.

Algorithm 2: pruneSpace(f)

```
1  for each partition P for which minAngle(f, P) < 90° do
2      if maxAngle(f, P) ≥ 90° then r^U_{f:P} = ∞;
3      ;
4      else r^U_{f:P} = dist(f,q) / (2 cos(maxAngle(f,P)));
5      ;
6      Set r_{B:P} to the radius of k-th smallest upper arc of P;
7      if f is a significant facility of P then // use Lemma 4&5
8          insert f in sigList of P in sorted order of
           r^L_{f:P} = dist(f,q) / (2 cos(minAngle(f,P)));
```

Algorithm 2 describes how the space is pruned using a facility f. According to Lemma 3, a facility f cannot prune any point $p \in P$ if $minAngle(f, P) \geq 90°$. Therefore, the algorithm considers only the partitions that have minimum subtended angle from f less than $90°$ (line 1). For each such partition P, the algorithm computes $r^U_{f:P}$, the upper arc of f, as described in the previous section. Then, the bounding arc $r_{B:P}$ of the partition P is updated (line 6). Recall that the bounding arc corresponds to the k-th smallest upper arc of P. The algorithm uses a heap of size k to maintain k smallest upper arcs. Hence, updating $r_{B:P}$ takes $O(\log\ k)$.

Finally, the facility is inserted in the $sigList$ if it is a significant facility of the partition P (by applying Lemma 4 or 5 depending on whether f lies in P or outside it). $sigList$ is maintained in sorted order of $r^L_{f:P}$ (line 8). As we show in Section IV, the expected size of $sigList$ is $O(k)$. Hence, the expected cost of line 8 is $O(\log\ k)$. The worst case size of $sigList$ is $O(m)$ where m is the number of facilities considered for pruning. Hence, the worst case insertion cost is $O(\log\ m)$. Since t partitions are considered, the total expected cost of Algorithm 2 is $O(t \log\ k)$ and the total worst case cost is $O(t \log\ m)$.

Since m is the total number of times Algorithm 2 is called (at line 9 of Algorithm 1), the total time the algorithm spends in pruning the space is $O(tm \log\ m)$ in the worst case (while the expected cost is $O(tm \log\ k)$).

2) Verification Algorithm: In the verification phase, the users that do not lie in the pruned area are shortlisted and are called candidate users. The verification algorithm iteratively accesses the nodes of the R-tree that indexes the users. If an entry e (the intermediate node, leaf node or the user object) lies completely in the pruned area, it is ignored. Otherwise, if e is an intermediate or leaf node, its children are accessed iteratively. If e is a data object and does not lie in the pruned area, it is called a candidate object and is verified by calling isRkNN(u) (Algorithm 3).

Algorithm 3: isRkNN(u)

```
Output : Returns true if u is RkNN. Otherwise, returns false.
1  Let P be the partition in which u lies;
2  counter=0;
3  for each f ∈ sigList of P in ascending order of r^L_{f:P} do
4      if dist(u, q) ≤ r^L_{f:P} then
5          return true;
6      if dist(u, f) < dist(u, q) then
7          counter ++;
8          if counter ≥ k then
9              return false;
10 return true;
```

Algorithm 3 verifies a user u as follows. The algorithm accesses the $sigList$ of the partition P in which the user u lies. The facilities in $sigList$ are accessed in ascending order of the radii of their lower arcs (i.e., $r^L_{f:P}$) (line 3). A counter is initialized to zero. This counter records the number of facilities that prune u and is incremented whenever the accessed facility f is found to prune u, i.e., $dist(u, f) < dist(u, q)$ (line 7). If the counter is at least equal to k, the algorithm returns *false* because u is not RkNN of q (line 9). At any stage, if $dist(u, q) \leq r^L_{f:P}$ for the accessed facility f, the user is confirmed as RkNN and the algorithm returns *true* (line 5). This is because counter is less than k and none of the remaining facilities can prune u (as implied by Lemma 2). Also, if the counter remains less than k after processing all facilities in $sigList$, the algorithm confirms u as RkNN and returns *true* (line 10).

3) Answering Monochromatic Queries: Since a facility f cannot pruned itself, we cannot prune a facility f if it lies outside the upper arcs of k facilities. However, we can safely prune a facility that lies outside $k + 1$ upper arcs, i.e., the bounding arc r_B corresponds to the radius of $(k + 1)$-th smallest upper arc of a partition. Hence, the pruning phase is called by setting k to $k + 1$. In the verification phase, the facilities that lie inside r_B are considered the candidates. Each candidate f is verified by calling Algorithm 3 with a minor change that the candidate facility f is skipped from $sigList$ (at line 3) because f cannot prune itself.

IV. Theoretical Analysis

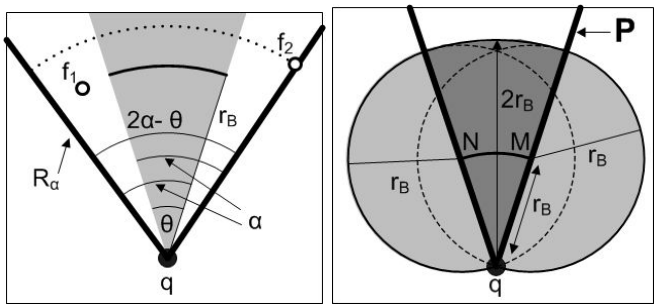
We assume that the facilities and the users are uniformly distributed in a unit space. The number of facilities is $|F|$ and the number of users is $|U|$. In this section, we use radian as the unit of angle. If t is the total number of partitions, the angle range of each partition is $2\pi/t$. We denote the angle range of each partition using θ and assume that θ is at most $\pi/3$ (the maximum angle of a partition in our experiments). For larger θ, the bounds can be obtained following the similar ideas.

A. Radius of bounding arc (r_B)

Let P be a partition with angle range θ as shown in Fig. 11(a) (the shaded area). Let α be an angle such that $\frac{\theta}{2} < \alpha < \frac{\pi}{2}$. We use R_α to denote a region such that for any point $x \in R_\alpha$, $maxAngle(x, P) \leq \alpha$. Fig. 11(a) shows R_α with thick boundaries and for any facility $f \in R_\alpha$, $maxAngle(f, P) \leq \alpha$. Let f_k be the k-th nearest facility of q in region R_α. Let $d_k = dist(f_k, q)$. Assume that $maxAngle(f_k, P) = \alpha$ (see the facility f_2 in Fig. 11(a) assuming $k = 2$). Note that, under this assumption, $r_B \leq \frac{d_k}{2\cos(\alpha)}$ because the upper arc of f_k is not among $k - 1$ smallest upper arcs (i.e., there are at least $k - 1$ facilities such that for each such facility f, $dist(f, q) \leq d_k$ and $maxAngle(f, P) \leq \alpha$). We overestimate r_B by assuming[2] that $r_B = \frac{d_k}{2\cos(\alpha)}$.

$$r_B = \frac{d_k}{2\cos(\alpha)} \tag{1}$$

[2]Overestimation is due to our assumption that $maxAngle(f_k, P) = \alpha$. Note that the best scenario (i.e., minimum r_B) is when $maxAngle(f_k, P)$ is minimum (i.e., $\theta/2$). Therefore, even when θ approaches zero, the overestimation is by a factor of $1/\cos(\alpha)$. As described later, α is chosen to be at most $7\pi/24$. Hence, the overestimation is at most by a factor 1.64.



(a) Analysing r_B (b) Expected size of $sigList$

Fig. 11. Theoretical Analysis

Now, we estimate d_k. Note that the angle range for the region R_α is $(2\alpha - \theta)$ as shown in Fig. 11(a). Since d_k is the distance of k-th nearest facility, the area of R_α that contains k facilities is $\frac{1}{2}d_k^2(2\alpha-\theta)$ (i.e., the area covered by the dotted arc in Fig. 11(a)). Since we assume that $|F|$ facilities are uniformly distributed in a unit space, k facilities are expected to be found in a space with area $k/|F|$. So d_k can be estimated by the following equation.

$$d_k = \sqrt{\frac{2k}{|F|(2\alpha - \theta)}} \tag{2}$$

Hence, r_B can be estimated by the following equation.

$$r_B = \frac{d_k}{2\cos(\alpha)} = \frac{1}{\cos(\alpha)}\sqrt{\frac{k}{2|F|(2\alpha - \theta)}} \tag{3}$$

Note that our analysis depends on the selection of α, i.e., the region R_α. To eliminate α from Eq. (3) and get a reasonable approximation of r_B, we choose α such that the angle range of the region R_α is $\pi/4$, i.e., $(2\alpha - \theta) = \pi/4$. We remark that other reasonable values of the angle range of R_α can be chosen to obtain similar results.

$$r_B = \frac{d_k}{2\cos(\frac{\theta+\pi/4}{2})} = \frac{1}{\cos(\frac{\theta+\pi/4}{2})}\sqrt{\frac{2k}{\pi|F|}} \tag{4}$$

B. Size of $sigList$

In this section, we estimate the expected size of $sigList$ of a partition P denoted as $|sigList|$. Recall that a facility f is added to $sigList$ of p only if it is not pruned using Lemma 4 and Lemma 5 (i.e., it lies in the shaded area of Fig. 11(b)). Let the shaded area be called significant area and be denoted as A_s. Since we have estimated r_B, A_s can be obtained. Specifically, A_s is the sum of the area of P covered by the arc of radius $2r_B$ (the dark shaded area in Fig. 11(b)) and the area of two half circles centered at M and N with radius r_B (the light shaded area in Fig. 11(b)).

$$A_s = \frac{1}{2}(2r_B)^2\theta + \pi r_B^2 = r_B^2(2\theta + \pi) \tag{5}$$

The expected number of facilities in the significant area is $A_s \times |F|$. Thus the expected size of $sigList$ (denoted as $|sigList|$) can be obtained by the following equation.

$$|sigList| = r_B^2(2\theta + \pi)|F| \tag{6}$$

Replacing the value of r_B from Eq. (4) in Eq. (6) gives the expected size of $sigList$.

$$|sigList| = \frac{2k(2\theta + \pi)}{\pi\cos^2(\frac{\theta+\pi/4}{2})} \tag{7}$$

Note that the expected size of $sigList$ does not depend on the total number of facilities. The minimum value of $|sigList|$ is $2.34k$ (when θ approaches zero) and the maximum value of $|sigList|$ is axround $9k$ (when $\theta = \pi/3$).

C. Expected number of candidates

Recall that every user u that lies in the unpruned area is the candidate. Since we know r_B, the unpruned area A_u can be estimated easily, i.e., $A_u = \pi r_B^2$. The expected number of candidates C is $A_u \times |U|$ where $|U|$ is the total number of users.

$$C = \pi r_B^2 \times |U| = \frac{2k \times |U|}{|F|\cos^2(\frac{\theta+\pi/4}{2})} \tag{8}$$

For the default setting in our experiments (i.e., $\theta = \pi/6$), $C = 3.1k|U|/|F|$. Note that, in the theoretical analysis, we use an overestimated value of r_B. This implies that the actual number of candidates is expected to be smaller.

D. I/O cost

We analyse the I/O cost of pruning phase and verification phase separately.

1) I/O cost of pruning phase: Recall that in the pruning phase we access the nodes of the facility R-tree in ascending order of their minimum distances from q. Furthermore, we do not access an entry e (a node or an object) for which $mindist(e, q) > 2r_B$, i.e., the entry is not (or does not contain) a significant facility. This implies that our algorithm accesses only the nodes or objects of the facility R-tree that lie within the distance range $2r_B$ from the query q. Hence, the I/O cost of the pruning phase of our algorithm is the same as the cost of a circular range query [28] centered at q and range $2r_B$. The I/O cost of a range query having a radius r can be estimated by the following equation [9].

$$Range\ query\ cost = 1 + \sum_{L=1}^{\lfloor log_f S \rfloor} \frac{\pi(\sqrt{\frac{f^L}{2S}} + r)^2 S}{f^L} \tag{9}$$

Here f is the fan-out of the R-tree and S is the number of objects indexed by the R-tree. The I/O cost of the pruning phase of our algorithm can be estimated by replacing S in Eq. (9) with $|F|$ and r with $2r_B$ where r_B can be estimated by Eq. (4).

2) I/O cost of verification phase: Recall that in the verification phase, an entry e is pruned if $mindist(q, e) > r_B$. Hence, the expected I/O cost of the verification phase is the same as the I/O cost of a range query with range r_B. This cost can be obtained by replacing S in Eq. (9) with $|U|$ and r with r_B.

V. Experimental Evaluation

A. Experimental Setup

We compare our algorithm with six-regions approach [3] and InfZone [9] which is the state-of-the-art RkNN algorithm. All algorithms are implemented[3] in C++ and the experiments are run on a 32-bit PC with Intel Xeon 2.40GHz dual CPU and 4GB memory running Debian Linux.

The experimental settings are similar to those used in [9] by our main competitor InfZone. Specifically, we use both the synthetic and real data sets. The real data set consists of $175,812$ points in North America[4] and we randomly divide these points into two sets of almost equal sizes. One of these sets corresponds to the facilities and the other to the users. Each synthetic data set consists of 50000, 100000, 150000 or 200000 points following either Uniform or Normal distribution. The default synthetic data set contains 100000 points and follows Normal distribution unless mentioned otherwise. We vary k from 1 to 25 and the default value of k is 10. For bichromatic RkNN queries, the number of users is the same as the number of facilities unless specifically mentioned.

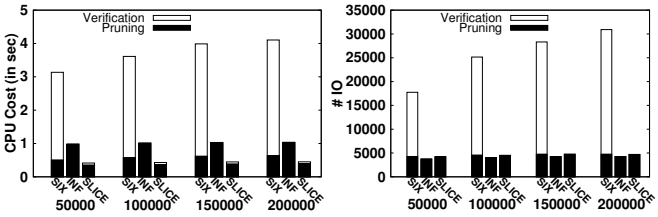
The page size of each R-tree is set to 4096 bytes. For six-regions approach, a buffer of 10 pages is used which uses random eviction strategy. We remark that InfZone and SLICE do not require buffer because each node is accessed only once by these algorithms. Thus, the buffer favors the six-regions approach (see Fig. 18 for more details).

B. Evaluating Performance

In this section, we compare the performance of the three algorithms for both the monochromatic and bichromatic RkNN queries. Six-regions [3] is shown as **SIX** and InfZone [9] is shown as **INF** in the figures. The default number of partitions for SLICE is 12 which is chosen based on the theoretical analysis and the preliminary experiments (see Fig. 19(a)).

1) Effect of data size: In Fig. 12 and 13, we study the effect of data size for monochromatic and bichromatic RkNN queries, respectively. For bichromatic queries, the number of users in each data set is the same as the number of facilities. Fig. 12(a) and 13(a) show the CPU cost of the three algorithms. As stated in Section I, the dominant cost for six-regions is the cost of verification phase whereas the major cost for SLICE and InfZone is due to the pruning phase.

Fig. 12(b) and 13(b) show the number of I/Os for each algorithm. As expected, the I/O cost of SLICE is slightly larger than the I/O cost of InfZone because InfZone prunes a larger area (hence, prunes more entries of the R-tree). The I/O cost

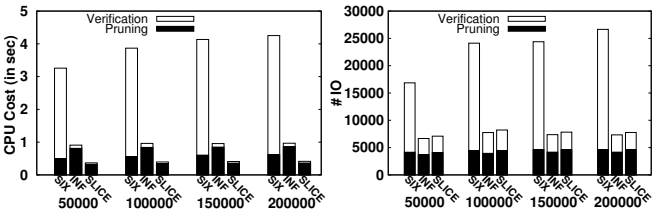


(a) Number of facilities (b) Number of facilities

Fig. 12. Monochromatic Queries: Effect of data set size (Normal Distribution)

of six-regions approach is much higher because it calls range queries to verify the candidates.

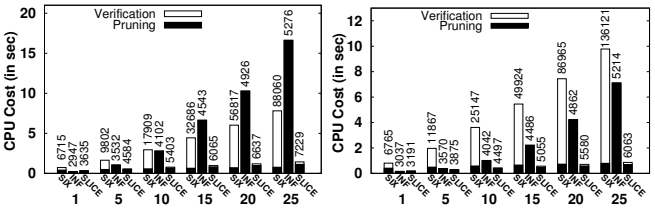
Due to space limitations, in the rest of the experiments, we focus on the CPU cost of the algorithms. *The numbers dislayed above the bars correspond to the number of I/Os unless mentioned otherwise*. Several existing works show the total cost of the algorithms by penalizing each algorithm for each I/O (e.g., average I/O cost for SSD disks is less than 0.1ms [29] so each algorithm may be charged 0.1ms per I/O). We do not follow this approach mainly because the I/O cost is highly system specific (e.g., type of disk drive used, workload etc.). Nevertheless, the interested readers can estimate the I/O cost by charging say 0.1ms for each I/O. We remark that under our system settings, the I/O cost for each algorithm is negligible as compared to its CPU cost.



(a) Number of facilities/users (b) Number of facilities/users

Fig. 13. Bichromatic Queries: Effect of data set size (Normal Distribution)

2) Effect of k: In Fig. 14 and 15, we study the effect of k for monochromatic and bichromatic RkNN queries, respectively. The performance of InfZone rapidly deteriorates as the value of k increases. Recall that the cost of pruning the space for InfZone is $O(km^2)$. The value of m increases as k increases. Hence, the pruning phase becomes quite expensive as can be observed in Fig. 14 and 15. The number of I/Os for our algorithm is larger than that of InfZone but is much lower as compared to the number of I/Os for six-regions.



(a) Varying k (Real Data) (b) Varying k (Normal Distribution)

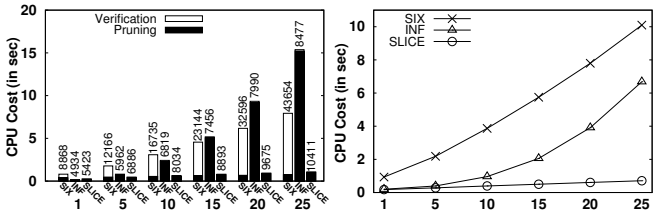
Fig. 14. Monochromatic Queries: Effect of k

The CPU cost of SLICE is lower than InfZone except when $k = 1$. This is because when $k = 1$, m is also small and the pruning cost of InfZone is low. However, note that SLICE

[3]All codes, data sets and scripts used for generating the figures are available at http://users.monash.edu.au/%7Eaamirc/reproducibility/slice/

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

scales much better than the other two algorithms and is up to an order of magnitude more efficient. In Fig. 15(b), we show the results for Normal distribution using lines (instead of bars) to clearly demonstrate how the three algorithms scale with the increase in k.



(a) Varying k (Real Data) (b) Varying k (Normal Distribution)

Fig. 15. Bichromatic Queries: Effect of k

3) Effect of data distribution: In Fig. 16, we study the effect of data distribution on each algorithm. The data distribution of the facilities and the users is shown as (D_f,D_u) where D_f and D_u correspond to the distribution of facilities and users, respectively. U, R and N correspond to Uniform, Real and Normal distribution, respectively. For instance, (U,N) correspond to the data set where the facilities follow Uniform distribution and the users follow Normal distribution. Since Real data set consists of two sets each containing almost $87,900$ points, in this experiment, the synthetic data sets also contain the same number of points. Fig. 16 demonstrates that SLICE is significantly faster than the other two algorithms for all different combinations of data distribution.

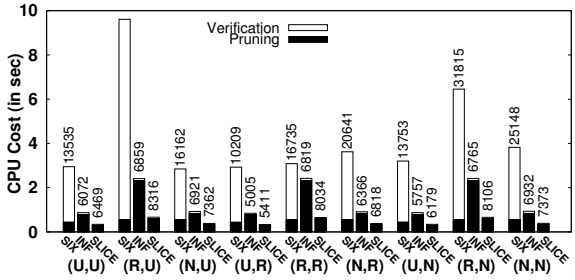


Fig. 16. Effect of Data Distribution

4) Effect of number of users relative to number of facilities: In this experiment, we fix the number of facilities to $100,000$ and change the number of users to see the effect of change in the relative size of the two data sets. The data set shown as $x\%$ correspond to the case when the number of users is $x\%$ of the number of facilities, i.e., the number of users is $\frac{100000\times x}{100}$. Fig. 17 shows that the cost of six-regions approach increases significantly with the increase in the number of users. On the other hand, the cost of InfZone and SLICE is not significantly affected. This is because the number of candidates increases as the number of users increases. Since the verification phase for the six-regions approach is significantly more expensive, its cost is severely affected. In contrast, InfZone and SLICE have much more efficient verification techniques. Therefore, the cost does not increase substantially.

5) Effect of buffer size: As stated earlier, InfZone and SLICE do not require buffer because these algorithms access each node at most once. On the other hand, six-regions approach issues multiple range queries and its I/O cost is significantly affected by buffer size. In Fig. 18, we demonstrate the effect of buffer size on six-regions. The I/O cost of six-regions decreases significantly with the increase in the buffer size. However, the cost remains unchanged when buffer size is 20 or larger. In all cases, the I/O cost of six-regions is much higher than our algorithm which does not require buffer.

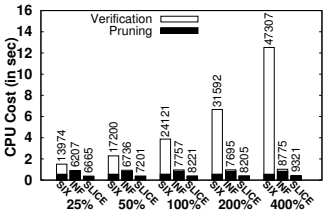


Fig. 17. % of users

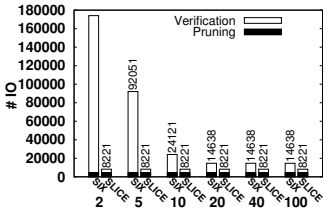
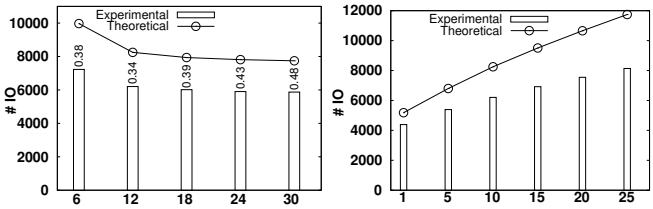


Fig. 18. Effect of buffer size

C. Evaluating Theoretical Analysis

The experimental settings and the default values are the same as used in the previous section except that we use Uniform data sets to verify the theoretical analysis.

1) I/O Cost: In Fig. 19(a), we change the number of partitions used by our approach and see the effect on I/O cost (the number above bars correspond to the CPU cost in seconds). The I/O cost estimated by the theoretical analysis is higher because we use an overestimation of r_B in the analysis. Nevertheless, the experimental results follow the trend exhibited by our theoretical analysis. As expected, the I/O cost decreases with the increase in the number of partitions. However, with the increase in number of partition, the CPU cost also increases. This is because more partitions are to be processed for each facility used for pruning. We choose default number of partitions in our experiments to be 12 because the I/O cost does not significantly decrease when the number of partitions is further increased. On the other hand, the CPU time increases as the number of partitions increases. Fig. 19(b) demonstrates that the actual I/O cost follows the trend predicted by the theoretical analysis for varying k.

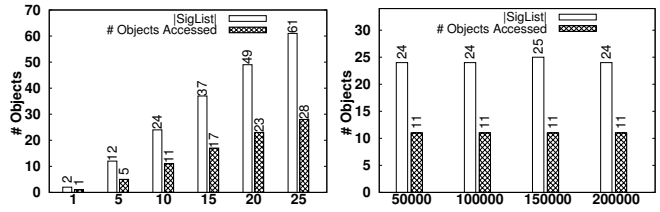


(a) Number of partitions (b) k

Fig. 19. Effect of number of partitions and verifying theoretical analysis

2) Size of sigList and verification cost: In Fig. 20, we show the size of $sigList$ for varying k and varying number of facilities. Recall that our theoretical analysis indicated that the expected size of $sigList$ is $O(k)$ and it does not depend on the number of facilities. Fig. 20(a) confirms that the size of $sigList$ increases linearly with k. Fig. 20(b) confirms that the number of facilities does not affect the size of $sigList$. Hence, the expected cost to verify a candidate is $O(k)$.

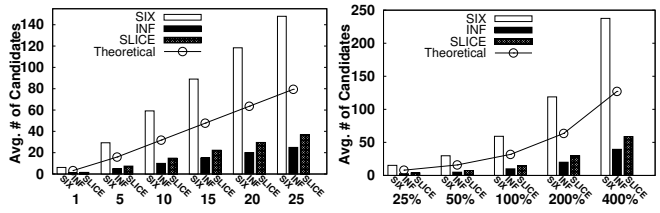
Recall that, in order to verify a user u, the algorithm accesses $sigList$ in a specific order until u is verified. Fig. 20



(a) Varying k (b) Number of facilities

Fig. 20. The size of $sigList$ and average number of objects accessed to verify a candidate

also shows the average number of objects accessed from the $sigList$ before a user u is verified. Note that the average number of objects accessed for the verification is close to k.



(a) k (b) % of users

Fig. 21. Number of candidates and verifying theoretical analysis

3) Number of candidate objects: In Fig. 21, we compare the number of candidates (i.e., the unpruned users) for each algorithm. Specifically, we study the effect of k in Fig. 21(a) and the effect of relative size of user data set to the facility data set in Fig. 21(b). Fig. 21 also shows the number of candidates we estimated in Section IV. Note that the number of candidates for SLICE is quite close to the number candidates for InfZone. This demonstrates that the area pruned by SLICE is similar to the area pruned by InfZone. Also, as stated in Section I, six-regions approach has significantly larger number of candidates.

VI. Conclusion

In this paper, we propose an efficient algorithm for reverse k nearest neighbors queries. The research in the past has mainly focused on half-space pruning approach which is generally believed to be superior to regions-based pruning. In this paper, we demonstrate that the regions-based pruning has certain advantages and it may be quite effective if its limitations are addressed appropriately. Based on several interesting observations, we rectify the weaknesses of regions-based approach and present an efficient algorithm to compute RkNN queries. We also provide a detailed theoretical analysis to analyse the cost of our algorithm. The extensive experimental study demonstrates that our approach is several times more efficient than the state-of-the-art technique and scales much better when k increases.

Acknowledgment

Muhammad Aamir Cheema is supported by ARC DE130101002 and DP130103405. Xuemin Lin is supported by NSFC61232006, NSFC61021004, ARC DP120104168 and DP110102937. The research of Ying Zhang is supported by ARC DP130103245 and DP110104880. This research was partially conducted while Muhammad Aamir Cheema was employed by The University of New South Wales.

References

[1] C. Yang and K.-I. Lin, "An index structure for efficient reverse nearest neighbor queries," in *ICDE*, 2001.
[2] K.-I. Lin, M. Nolen, and C. Yang, "Applying bulk insertion techniques for dynamic reverse nearest neighbor problems," *IDEAS*, 2003.
[3] I. Stanoi, D. Agrawal, and A. E. Abbadi, "Reverse nearest neighbor queries for dynamic databases," in *ACM SIGMOD Workshop*, 2000.
[4] E. Achtert, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle, "Reverse k-nearest neighbor search in dynamic and general metric databases," in *EDBT*, 2009, pp. 886–897.
[5] M. Sharifzadeh and C. Shahabi, "Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries," *PVLDB*, vol. 3, no. 1, pp. 1231–1242, 2010.
[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.
[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, W. Zhang, and Y. Zhang, "Influence zone: Efficiently processing reverse k nearest neighbors queries," in *ICDE*, 2011, pp. 577–588.
[10] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *SIGMOD*, 2000.
[11] Y. Tao, D. Papadias, and X. Lian, "Reverse knn search in arbitrary dimensionality," in *VLDB*, 2004.
[12] M. A. Cheema, W. Zhang, X. Lin, and Y. Zhang, "Efficiently processing snapshot and continuous reverse k nearest neighbors queries," *VLDB J.*, vol. 21, no. 5, pp. 703–728, 2012.
[13] H.-P. Kriegel, P. Kröger, M. Renz, A. Züfle, and A. Katzdobler, "Incremental reverse nearest neighbor ranking," in *ICDE*, 2009.
[14] M. A. Cheema, X. Lin, Y. Zhang, W. Wang, and W. Zhang, "Lazy updates: An efficient technique to continuously monitoring reverse knn," *PVLDB*, vol. 2, no. 1, pp. 1138–1149, 2009.
[15] M. A. Cheema, W. Zhang, X. Lin, Y. Zhang, and X. Li, "Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks," *VLDB J.*, vol. 21, no. 1, pp. 69–95, 2012.
[16] M. A. Cheema, X. Lin, W. Wang, W. Zhang, and J. Pei, "Probabilistic reverse nearest neighbor queries on uncertain data," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 4, pp. 550–564, 2010.
[17] T. Bernecker, T. Emrich, H.-P. Kriegel, M. Renz, , and S. Z. A. Züfle, "Efficient probabilistic reverse nearest neighbor query processing on uncertain data," in *PVLDB*, 2011.
[18] T. Emrich, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle, "Incremental reverse nearest neighbor ranking in vector spaces," in *SSTD*, 2009.
[19] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis, "Nearest neighbor and reverse nearest neighbor queries for moving objects," in *IDEAS*, 2002.
[20] T. Xia and D. Zhang, "Continuous reverse nearest neighbor monitoring," in *ICDE*, 2006, p. 77.
[21] X. Lian and L. Chen, "Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data," *VLDB J.*, 2009.
[22] T. Bernecker, T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle, "A novel probabilistic pruning approach to speed up similarity queries in uncertain databases," in *ICDE*, 2011, pp. 339–350.
[23] Y. Tao, M. L. Yiu, and N. Mamoulis, "Reverse nearest neighbor search in metric spaces," *TKDE*, vol. 18, no. 9, 2006.
[24] M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao, "Reverse nearest neighbors in large graphs," *IEEE Trans. Knowl. Data Eng.*, 2006.
[25] M. Safar, D. Ebrahimi, and D. Taniar, "Voronoi-based reverse nearest neighbor query processing on spatial networks," *Multimedia Syst.*, vol. 15, no. 5, pp. 295–308, 2009.
[26] M. L. Yiu and N. Mamoulis, "Reverse nearest neighbors search in ad hoc subspaces," *TKDE*, vol. 19, no. 3, pp. 412–426, 2007.
[27] F. P. Preparata and M. I. Shamos, *Computational Geometry An Introduction*. Springer, 1985.
[28] 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.
[29] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, J. L. Wiener, and G. Graefe, "Query processing techniques for solid state drives," in *SIGMOD Conference*, 2009, pp. 59–72.