Noname manuscript No.
(will be inserted by the editor)

Reverse k nearest neighbors queries and spatial reverse top-k queries

Shiyu Yang · Muhammad Aamir Cheema · Xuemin Lin · Ying Zhang · Wenjie Zhang

Received: date / Accepted: date

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. Almost all of the existing techniques to answer RkNN queries 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 regions-based pruning algorithm called SLICE that utilizes the strength of regions-based pruning and overcomes its limitations. We also study *spatial reverse top-k* (SRTk) queries that return every user u for which the query facility is one of the top-k facilities according to a given linear scoring function. We first extend half-space based pruning to answer SRTk queries. Then, we propose a novel regions-based pruning algorithm following SLICE framework to solve the problem. Our extensive experimental study on synthetic and real data sets demonstrates that SLICE is significantly more efficient than all existing RkNN and SRTk algorithms.

Shiyu Yang, Xuemin Lin and Wenjie Zhang
School of Computer Science and Engineering,
The University of New South Wales, Australia
E-mail: {yangs, lxue, zhangw}@cse.unsw.edu.au

Muhammad Aamir Cheema
Faculty of Information Technology, Monash University, Australia
E-mail: aamir.cheema@monash.edu

Ying Zhang
University of Technology, Australia
E-mail: ying.zhang@uts.edu.au

1 Introduction

Consider a set of facilities (e.g., supermarkets) and a set of users (e.g., residents). The influence set of a query facility q consists of every user u for which the query facility is an *important* facility. Since q is an *important* facility for such users, these users are said to be influenced by q, e.g., the query facility can send targeted deals to these users to attract them. In this paper, we study two types of influence set namely reverse k nearest neighbors and spatial reverse top-k users. These two differ from each other due to the different definition of *important* facilities. Below, we give more details of each.

Reverse k Nearest Neighbors (RkNN) Queries. In the context of RkNN queries, a facility is considered to be important for a user u if it is among her k-closest facilities. An RkNN query q returns every user u for which q is one of its k closest facilities. Consider the example of a restaurant. The people for which this restaurant is one of the k-closest restaurants are its potential customers and may be influenced by targeted marketing or deals.

Spatial Reverse Top-k Queries (SRTk) Queries. In RkNN queries, the notion of *importance* is based only on the distance between users and facilities. However, in many real world applications, distance is not usually the only criterion desired by the users and a user may also be interested in other attributes of the facilities such as price and rating. In the context of SRTk queries, a user u considers a query to be important if q is one of the top-k facilities for the user u according to a given linear scoring function.

Consider the example of a person who is looking for restaurants. She may be interested in the restaurants that are close to her location, have good reputations and are cheap, i.e., distance, rank and price are the three criterions. She may issue a top-k query that uses a scoring function involving these three criterions to compute the score of each restaurant and returns the top-k restaurants based on their scores.

We say that the query facility q is *important* for her if q is one of her top-k restaurants. Next, we formally define spatial reverse top-k queries.

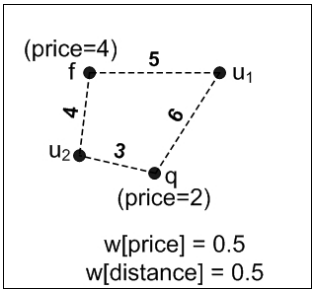


Fig. 1 RkNN vs SRTk queries ($k = 1$): RkNN = u_2, SRTk= u_1, u_2

Let W be a linear scoring function that computes the score of a facility f for a given user u. A spatial reverse top-k (SRTk) query returns every user u for which q is one of the top-k facilities according to the scoring function W.

Example 1 Consider the example of Fig. 1 that shows two facilities (q and f) and two users u_1 and u_2. The distances between the users and the facilities are shown above the broken lines. RkNN ($k = 1$) returns the user u_2 because q is the closest facility for the user u_2. The user u_1 is not the RkNN because q is not the closest facility for u_1.

Now, consider a spatial reverse top-k ($k = 1$) query and assume that the scoring function W is a linear scoring function involving two criterions (price and distance) and both criterions have equal weighting, i.e., $w[price] = w[distance] = 0.5$. Then, the score of a facility for a user is the weighted sum of its distance from the user and its price, e.g., $score(u_1, q) = 0.5 \times 6 + 0.5 \times 2 = 4$, $score(u_1, f) = 0.5 \times 5 + 0.5 \times 4 = 4.5$. Although f_1 is closer to the user u_1, the top-1 facility of u_1 is q because it has a better score due to it being cheaper. The top-1 facility of u_2 is q because $score(u_2, q) = 2.5$ and $score(u_2, f_1) = 4$. For $k = 1$, the spatial reverse top-k query returns both of the users u_1 and u_2 because for both users q is the top-1 facility.

Next, we present the contributions we make in this paper for both the RkNN queries and SRTk queries.

1.1 Reverse k Nearest Neighbors Queries

RkNN query has received significant research attention [42, 20, 26, 1, 25, 40, 27, 16, 7] ever since it was introduced in [17]. The existing techniques use a *pruning* and *verification* framework. The two most notable pruning strategies employed by the existing techniques are *regions-based pruning* [26, 39] and *half-space pruning* [28, 40, 7]. As we show later in Section 3.2, the advantage of half-space pruning is that it prunes a much larger area as compared to the area pruned by the regions-based pruning. However, its disadvantage is that the cost of pruning is significantly higher than the regions-based pruning.

It has been a general perception that the half-space pruning is superior to the regions-based pruning. Consequently, most of the proposed techniques use half-space pruning to compute RkNNs and related problems. Surprisingly, there has been no effort to utilize the strength of regions-based pruning (i.e., cheap pruning) and to address 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 existing regions-based technique (called six-regions) with almost similar computational complexity. Furthermore, it significantly improves the verification phase of the algorithm by employing the novel concept of *significant facilities* (see Section 3.2 for more details).

Below, we summarize our contributions for RkNN queries.

- 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 [28, 40, 7, 18] and its variants (e.g., continuous RkNN queries [8, 16, 11], probabilistic RkNN queries [6, 4], and incremental RkNN queries [18, 13] 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 demonstrate this by developing a regions-based technique to answer spatial reverse top-k queries.
- 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 $k = 1$).

1.2 Spatial Reverse Top-k Queries

Spatial reverse top-k query is closely related to the traditional reverse top-k query which has been extensively studied [34, 15, 37, 14, 47, 36, 38, 12, 9] in the past few years. One major difference is that the traditional reverse top-k queries do not consider the spatial distance between the locations of users and facilities as one of the criterions. Due to this important difference, the existing techniques to answer traditional reverse top-k queries cannot be applied or easily extended to answer spatial reverse top-k queries. Specifically, the traditional reverse top-k queries assume that each facility f has a set of attributes (e.g., price) and, for each attribute, the facility has the same value for all the users (e.g., price is the same for all the users). In contrast, the distance between

a user u and a facility f depends on the locations of u and f. Consequently, the distance value of a facility is different for each user and this important difference makes the existing techniques inapplicable for the spatial reverse top-k queries.

To the best of our knowledge, the only existing technique that can be used to answer spatial reverse top-k queries has been recently proposed in [21]. However, the proposed technique has certain limitations. It can only handle the case when $k = 1$. Furthermore, the techniques can only be applied when the number of attributes/criterions is two (including the distance criterion). It requires non-trivial changes in the indexing and query processing algorithm to handle more attributes and $k > 1$. Our experiments demonstrate that our proposed approach significantly outperforms it even for $k = 1$ and two attributes both in terms of CPU time and I/O cost.

Below, we summarize our contributions for spatial reverse top-k queries.

- To the best of our knowledge, we are the first to propose solutions for spatial reverse top-k queries for arbitrary number of attributes/criterions and $k \geq 1$. To showcase the strength of regions-based pruning, based on several non-trivial observations, we develop regions-based pruning techniques for SRTk queries and propose a highly efficient algorithm.
- As stated earlier, the existing technique (PCK) [21] can only handle at most two attributes and $k = 1$. To demonstrate the effectiveness of our proposed algorithm for arbitrary number of attributes and $k > 1$, we carefully extend TPL [28] to efficiently answer SRTk queries - TPL is arguably the most popular half-space based pruning approach for RkNN queries. We conduct extensive experiments on real and synthetic data sets and demonstrate that SLICE is up to several orders of magnitude better than the TPL based algorithm and PCK in terms of both CPU cost and I/O cost.

The rest of the paper is organized as follows. In Section 2, we present the related work. Our techniques to answer RkNN queries are presented in Section 3. Section 4 presents the algorithms to answer spatial reverse top-k queries. Experimental evaluation is presented in Section 5 followed by conclusion in Section 6.

2 Related work

2.1 Reverse k Nearest Neighbors

RkNN queries have been extensively studied considering different settings such as continuous RkNN queries [2,41, 16,8], probabilistic RkNN queries [6,19,3,4], RkNN queries on graphs [29,46,24], metric spaces [29] and adhoc spaces [45] 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.* [17] 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 [42,20].

Next, we briefly describe some of the most related techniques that do not require pre-computation namely six-regions [26], TPL [28], FINCH [40] and InfZone [7]. A comprehensive experimental study comparing these algorithms is presented in [43]. 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. 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. Stanoi *et al.* [26] proposed the first technique that does not need any pre-computation. 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 1, 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.* [28] propose TPL that prunes the space using the half-space pruning. Let $B_{a:q}$ denote the perpendicular bisector between a facility a and q (see Fig. 3). The bisector divides the space into two halves. We use $H_{a:q}$ to denote the half-space that contains a. For a point p that lies in $H_{a:q}$, $dist(p, a) < dist(p, q)$. In other words, p cannot be the RkNN ($k = 1$) 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.

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.

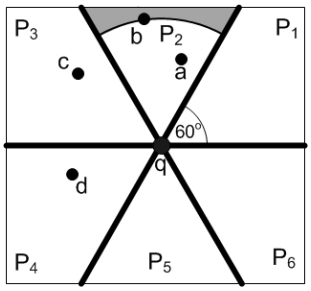


Fig. 2 Six-regions [26]

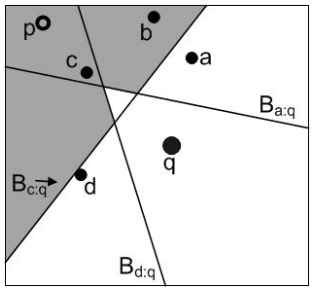


Fig. 3 TPL [28]

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 $O(km)$.

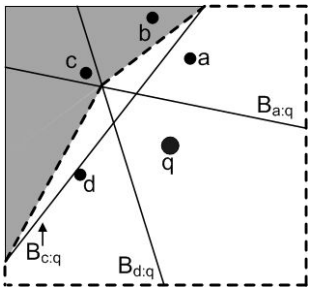


Fig. 4 FINCH [40]

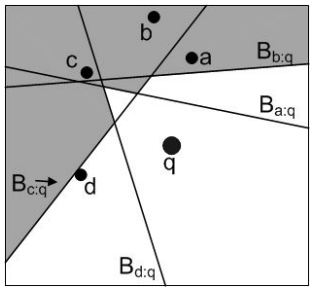


Fig. 5 InfZone [7]

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.* [40] 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, FINCH and TPL is quite expensive because it requires issuing a range query for each candidate. Cheema *et al.* [7] 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 [22] 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.* [7] 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 $O(km^2)$ [10].

2.2 Reverse Top-k Queries

Reverse top-k query has gained significant research attention since it was first introduced by Vlachou *et al.* [35]. They proposed two variants of the reverse top-k queries namely *monochromatic* and *bichromatic* reverse top-k queries. A monochromatic query returns every possible scoring function for which q is one of the top-k objects. On the other hand, in a bichromatic query, a set of scoring functions F is given and every function $f \in F$ is returned for which q is one of the top-k objects. Many variations of reverse top-k query have also been studied in the past few years such as probabilistic reverse top-k query [15], continuous reverse top-k query [37, 14] and reverse top-k query on graphs [47] etc. Next, we briefly describe the most closely related work.

Vlachou *et al.* [35] propose a threshold-based algorithm which utilizes some geometric properties to eliminate unnecessary objects based on a threshold. The threshold is maintained throughout the execution of the algorithm and is improved as new users are accessed. They also propose a space

partitioning based index structure to further improve reverse top-k query processing.

In their follow up work, Vlachou *et al.* [36,38] propose techniques to further improve their earlier algorithm. In [36], the authors discuss the geometric interpretation of the solution space for higher dimensionality. They also present a more comprehensive experimental study which evaluates the proposed algorithms for both monochromatic and bichromatic reverse top-k query. In [38], they propose an efficient algorithm which processes reverse top-k query by traversing the branch-and-bound data structures while utilizing novel lower and upper bounds.

Chester *et al.* [12] propose an index structure specific for a fixed value k such that multiple reverse top-k queries can be efficiently answered using this index. If the value of k is unknown at the query time (which is usually the case), the index is to be constructed on-the-fly which becomes the bottleneck. The proposed index pays off when numerous queries use the same value of k. The proposed techniques are only applicable to two dimensional data sets.

Cheema *et al.* [9] propose I/O and CPU efficient algorithms to answer various ranking related queries including the reverse top-k queries. They utilize the concept of k-lower envelope to efficiently answer the reverse top-k queries. Their main reverse top-k query processing algorithm is based on an I/O optimal algorithm to compute k-lower envelope. Similar to the work by Chester *et al.* [12], the proposed techniques are only applicable for two dimensional data sets.

We remark that the problem studied in this paper is different from the traditional reverse top-k queries. Specifically, in the traditional reverse top-k queries, the attributes of a facility are the same for all users, e.g., price is the same regardless of the user. In contrast, in spatial reverse top-k queries, the distance between facilities and users is also one of the attributes/criterions. Note that the distance between a facility and users may be different for each user. Hence, the existing techniques cannot be applied or extended for the spatial reverse top-k queries.

The two most closely related work to the problem studied in this paper are presented in [37,21]. In [37], the authors study a continuous version of SRTk problem where the users are continuously moving. They identify various properties of spatial reverse top-k queries that enable efficient monitoring of the result set. However, they assume that the initial results of spatial reverse top-k query are already known, and they focus on updating the results when the users move, without re-computing the results from scratch.

Recently, Park *et al.* [21] proposed an algorithm (PCK) for the spatial reverse top-k ($k = 1$) queries. The PCK algorithm consists of both spatial and non-spatial pruning techniques. They also introduce Hausdorff distance to improve efficiency of query processing. However, the online Hausdorff distance computation is quite time consuming. Furthermore, the proposed techniques can only handle two attributes (including the spatial distance). Also, the proposed techniques are only applicable for $k = 1$. Extending the techniques for more than two attributes and $k > 1$ is non-trivial. Nevertheless, we demonstrate that our proposed algorithm is several orders of magnitude better than PCK in terms of both CPU and I/O cost.

3 Answering Reverse k Nearest Neighbors Queries

In Section 3.1, we formally define RkNN queries and its two variations. The motivation behind our algorithm, SLICE is presented in Section 3.2. Our pruning and verification techniques are presented in Section 3.3 and Section 3.4, respectively. The overall algorithm is presented in 3.5.

3.1 Problem Definition

RkNN queries are classified [17] 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 [7,40,26], the focus of this paper is on two dimensional location data. Similar to most of the existing algorithms, we assume that the users and facilities are indexed by two separate R*-trees called user R*-tree and facility R*-tree, respectively. 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 3.5.3, we show that our techniques can be easily applied for the case of monochromatic RkNN queries.

3.2 Motivation

As discussed earlier, there are two most notable pruning strategies used by the existing techniques: half-space pruning; and regions-based pruning. 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

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

Table 1 Comparison of computational complexities

Fig. 2 and Fig. 3). The advantage of the regions-based approach is that, once d_k is computed, the cost of checking whether a point p can be pruned or not is $O(1)$ where d_k is the distance between q and its k-th nearest facility in the region that p lies in (see Section 2). 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 1 compares six-regions approach [26] and InfZone [7, 10], the state-of-the-art half-space pruning based approach. 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 1, 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 $dist(u, q)$. 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 1 also shows 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 [40], InfZone [7]). 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). This motivates us to carefully develop a technique called SLICE that utilizes the strengths of regions-based pruning and address its limitations. 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. SLICE also 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 1 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.

We also remark 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 $3.1k$ (see theoretical analysis in [44]). 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 1, 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. A detailed theoretical analysis can be found in the conference version of this paper [44].

We remark that SLICE aims at reducing the overall computational cost by compromising on a slightly higher I/O cost than InfZone [7]. Later in Section 5, 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.

Next, we present our techniques to improve the pruning power of regions-based pruning by keeping its low computational cost.

3.3 Reviving Regions-Based Pruning

First, we define a few terms and notations (Table 2 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 $angle(x, y) \leq 180°$. Similar to six-regions approach, we divide the whole space into several partitions. Next, we define maximum and minimum subtended angle between a point and a partition.

Definition 2 Maximum (minimum) subtended angle. Given a query point q, maximum subtended angle between a point x and a partition P, denoted as $maxAngle(x, P)$, is the maximum subtended angle between x and any point p in the partition P, i.e., $argmax_{p \in P}\ angle(x, p)$. The minimum subtended angle is defined similarly and is denoted as $minAngle(x, P)$.

In Fig. 6(b), $maxAngle(f, P) = \theta_1$ and $minAngle(f, P) = \theta_2$ (the partition P is shown shaded). 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 Law of Cosines.

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

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

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

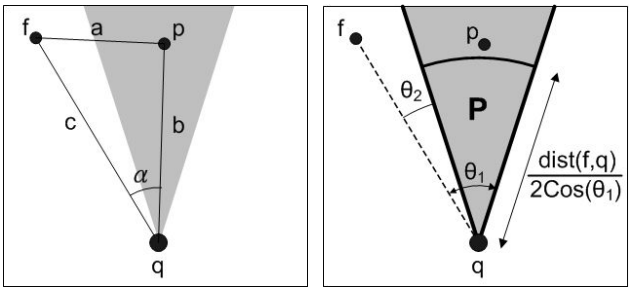
Since $\alpha \leq \theta$ and $0 \leq \theta < 90°$, $\frac{\cos(\alpha)}{\cos(\theta)} \geq 1$. Hence, $c^2 - 2bc \cdot \cos(\alpha) < 0$. □

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 $\frac{dist(f,q)}{2\cos(\theta)}$ 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 $\theta \geq 90°$, $r^U_{f:P} = \infty$.

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 2 Notations



(a) Proof of Lemma 1 (b) Illustration of terms

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

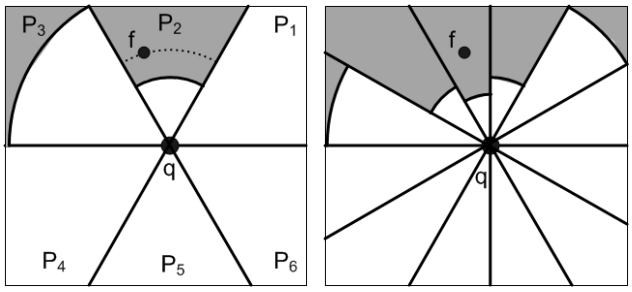
Fig. 6(b) shows the upper arc of f for the partition P. We say that a point lies outside (resp. inside) an arc of radius r if its distance from the center of the arc is greater (resp. smaller) than r. According to Lemma 1, a facility prunes area outside the upper arc of f for every partition P for which $maxAngle(f, P) < 90°$. 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 search 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 $\frac{dist(f,q)}{2\cos(60°)} = dist(f, q)$.

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 di-

[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 $O(1)$.



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

Fig. 7 Comparison with six-regions approach

vide 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 the conference version of this paper [44], we presented 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 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.

3.4 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 un-necessary 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.

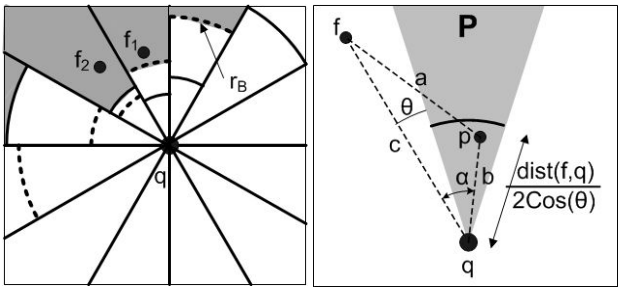


Fig. 8 The shaded area is pruned **Fig. 9** Lower arc

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). The expected size of $sigList$ is $O(k)$ [44]. 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 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 $\frac{dist(f,q)}{2\cos(\theta)}$ 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).

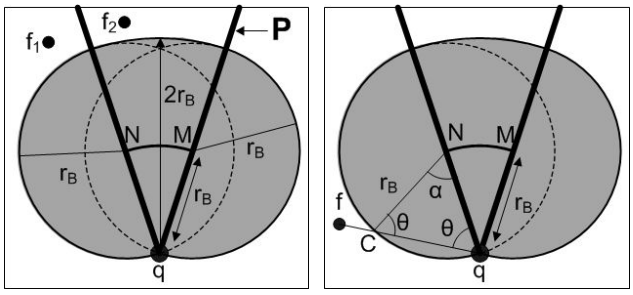
According to Lemma 2, a facility f cannot prune any point $p \in P$ that lies inside the lower arc.

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. 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 $dist(f, q) > \overline{Cq}$, we have $dist(f, q) > 2r_B\cos(\theta)$. Recall that $r^L = \frac{dist(f,q)}{2\cos(\theta)}$ which implies that $2r^L\cos(\theta) > 2r_B\cos(\theta)$ 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.

3.5 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.

3.5.1 Pruning Algorithm

Algorithm 1 describes the pruning phase. The space around q is divided into t equally sized partitions ($t = 12$ in our experiments). In the conference version of this paper [44], we provided 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 facility R-tree. 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). To check whether an entry e may or may not contain a significant facility, we apply 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: Filtering

```
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 a larger 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
3         r^U_{f:P} = ∞;
4     else
5         r^U_{f:P} = dist(f,q) / (2 cos(maxAngle(f,P)));
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). The expected size of $sigList$ is $O(k)$ [44]. 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 $O(tm \log\ k)$).

3.5.2 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 user R-tree. 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).

3.5.3 Answering Monochromatic Queries

Since a facility f cannot prune 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.

4 Answering Spatial Reverse Top-k Queries

In this section, we present our algorithms to answer spatial reverse top-k queries. First, we formally define the problem, and introduce terms and notations in Section 4.1. Then, we present a half-space based algorithm in Section 4.2 followed by our techniques to answer spatial reverse top-k queries using SLICE in Section 4.3.

4.1 Preliminaries

4.1.1 Problem Definition

Consider a set of users U and a set of facilities F. In addition to location coordinates, each facility $f \in F$ has d attributes (e.g., price, rating) and the value of the i-th attribute is denoted as $f[i]$. The distance between a user u and a facility f is denoted as $dist(u,f)$. Since $f[i]$ of a facility f remains the same regardless of the user u, each $f[i]$ is called a static attribute of the facility. On the other hand, $dist(u,f)$ may be different for each user u. Therefore, the distance is called the dynamic attribute of a facility. We assume that each attribute is normalized such that all the values are between 0 to 1.

Consider a $(d+1)$-dimensional linear scoring function W where each $w[i] \geq 0$ and $\sum_{i=1}^{d+1} w[i] = 1$. Here, $w[d+1]$ is the weighting for the dynamic attribute (i.e., $dist(u,f)$) and $w[i]$ (for $1 \leq i \leq d$) is the weight for each static attribute $f[i]$. The score of a facility f w.r.t. a user u is denoted as $score(u,f)$ and is computed as follows.

$$score(u,f) = w[d+1] \cdot dist(u,f) + \sum_{i=1}^{d} w[i] \cdot f[i] \quad (1)$$

Top-k facilities. Given a scoring function W, the top-k facilities of a user u are the k facilities having the smallest scores w.r.t. the user u. In other words, a facility f is one of the top-k facilities for a user u if there are less than k facilities that have a score smaller than $score(u,f)$.

Spatial reverse top-k (SRTk) query. Given a query facility q and a scoring function W, a spatial reverse top-k query returns every user u for which the query facility is one of its top-k facilities.

Remark. This definition differs from traditional reverse top-k queries (e.g., [34]) that assume that the users may have different scoring functions. Our definition is inspired by the existing work on spatial reverse top-k query [21] where the query defines a scoring function that is assumed to be the same for all users. Although we intend to investigate it in future, in this paper, we do not focus on the definition that assumes unique scoring functions for the users mainly due to the following reason.

While the locations of the users are easily obtainable, their preferences (i.e., scoring functions) are not well defined and are usually unknown to the system. Even if all users are able to define suitable scoring functions and the system manages to obtain these, it cannot be guaranteed that the users will use the same scoring functions in future, e.g., a user who usually gives higher weighting to price compared to the distance may give a higher weighting to distance when he is in hurry. Since the essence of reverse queries is to analyze the potential influence of a query facility as compared to the other facilities in the system, our definition allows the query facility to analyze its impact without requiring the scoring functions of all the users. If required, the query facility may further analyze the influence by issuing different queries each using a different scoring function.

4.1.2 Terms and notations

Static score of a facility. The static score of a facility f (denoted as f_s) is $\sum_{i=1}^{d} w[i] \cdot f[i]$. It is called static score because it remains the same for every user and does not depend on the distance between f and users.

Note that $score(u,f)$ can be computed by rewriting Eq. (1) as follows.

$$score(u,f) = f_s + w[d+1] \cdot dist(u,f) \quad (2)$$

Although our proposed techniques can be applied on any branch-and-bound data structure, we assume that both the sets of facilities and users are indexed by two different R*-trees. The tree that indexes the users is called the user R*-tree and the tree that stores the facilities is called the facility

Notation	Definition
$f[i]$	i-th attribute of a facility
$w[i]$	weight of i-th dimension
f_s	$\sum_{i=1}^{d} w[i] \cdot f[i]$ (static score of f)
$score(u, f)$	$f_s + w[d+1] \cdot dist(u, f)$
Δ_f	$(q_s - f_s)/w[d+1]$
e	an entry of the facility R*-tree
$e.min[i]$	minimum value of e in i-th dimension
$e.max[i]$	maximum value of e in i-th dimension
e_s^{min}	$\sum_{i=1}^{d} w[i] \cdot e.min[i]$
e_s^{max}	$\sum_{i=1}^{d} w[i] \cdot e.max[i]$
Δ_e^{min}	$(q_s - e_s^{max})/w[d+1]$
Δ_e^{max}	$(q_s - e_s^{min})/w[d+1]$

Table 3 Notations used for reverse top-k queries algorithms

R*-tree. The user R*-tree is a 2-dimensional R*-tree that indexes the location coordinates of the users. The facility R*-tree is a $(d+2)$-dimensional R*-tree that indexes d static attributes and two location coordinates for each facility f. We remark that converting the d static attributes of each facility to a one dimensional static score using the scoring function W and indexing in a 3-dimensional R*-tree (two location coordinates and one static score) is not a feasible solution because such an index would not support queries involving a different scoring function W'.

Static scores of a facility entry. Let e be an entry (intermediate or leaf node) of the facility R*-tree. We denote $e.min[i]$ (resp. $e.max[i]$) to denote the minimum (resp. maximum) value of e for the i-th static attribute. The minimum (resp. maximum) static score of an entry e is denoted as e_s^{min} (resp. e_s^{max}) and $e_s^{min} = \sum_{i=1}^{d} w[i] \cdot e.min[i]$ and $e_s^{max} = \sum_{i=1}^{d} w[i] \cdot e.max[i]$.

Given a point p, $maxdist(p, e)$ (resp. $mindist(p, e)$) corresponds to the maximum (resp. minimum) possible Euclidean distance between the entry e and the point p (considering only the location coordinates). Table 3 summarizes the terms and notations used throughout this section.

4.2 Half-space based solution

In this section, we present a solution based on the half-space based approach used to answer RkNN queries. The perpendicular bisector based pruning used for RkNN queries is not applicable for the reverse top-k query. However, we note that the half-space based pruning can be still applied using a hyperbola instead of perpendicular bisector. The details of hyperbola based pruning are presented in Section 4.2.1. We also note that certain new optimizations are possible for the spatial reverse top-k queries and we present the details of these optimizations in Section 4.2.2 and Section 4.2.3.

4.2.1 Hyperbola based pruning

In this section, we show how to extend the half-space based pruning for spatial reverse top-k queries. We say that a user u is pruned by a facility f if $score(u, f) < score(u, q)$. Note that a user u cannot be the reverse top-k of a query q if it is pruned by least k facilities. We say that a user u is *filtered* if it is pruned by at least k facilities.

Let f_s and q_s denote the static scores of two facilities f and q, respectively. Since a user u is pruned by the facility f if $score(u, f) < score(u, q)$, we write this inequality as follows.

$$f_s + w[d+1] \cdot dist(f, u) < q_s + w[d+1] \cdot dist(q, u)$$

This inequality can be re-written as below.

$$dist(f, u) - dist(q, u) < \frac{q_s - f_s}{w[d+1]}$$

Let $\Delta_f = \frac{q_s - f_s}{w[d+1]}$ be called the *static score gap* between q and f. A facility f prunes a user u (i.e., $score(u, f) < score(u, q)$) if the following inequality holds.

$$dist(f, u) - dist(q, u) < \Delta_f \tag{3}$$

When the facility f is clear by context, we denote Δ_f as Δ. The above inequality shows that $score(u, f) < score(u, q)$ if the difference between $dist(f, u)$ and $dist(q, u)$ is smaller than Δ, the static score gap between q and f. In other words, q can only have a better score than f if the difference between $dist(f, u)$ and $dist(q, u)$ is larger than Δ.

Corollary 1 *A facility f prunes a user u (i.e., $score(u, f) < score(u, q)$) if and only if $dist(f, u) - dist(q, u) < \Delta$.*

Example 2 Consider the example of Fig. 11 that shows two facilities f and q and a user u. The distances between the facilities and user are shown above the broken lines. Assume that the facilities have only one static attribute, e.g., price. Let $f[1] = 2$, and $q[1] = 8$. Assume that both the price and distance have the same weighting, i.e., $w[1] = w[2] = 0.5$. The static scores of the facilities are $f_s = 2 \times 0.5 = 1$ and $q_s = 8 \times 0.5 = 4$. $\Delta = \frac{4-1}{0.5} = 6$. Since $dist(f, u) - dist(q, u) = 4 - 7 = -3$ is less than Δ, the user u is pruned by the facility f. It can be confirmed that $score(u, f) < score(u, q)$, i.e. $score(u, f) = 3$ and $score(u, q) = 7.5$.

Note that the inequality of Eq. (3) defines a space that is bounded by a hyperbola $dist(f, u) - dist(q, u) = \Delta_f$. This hyperbola can be used for pruning, i.e., the space where $dist(f, u) - dist(q, u) < \Delta$ is bounded by the hyperbola and *every* user u that lies in the hyperbola can be pruned. The half-space that can be pruned using a facility f is denoted as $H_{f:q}$.

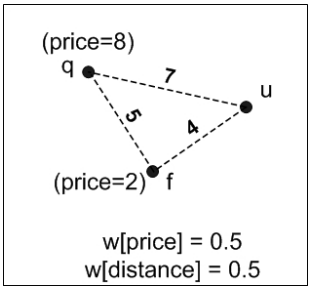


Fig. 11 u is pruned by f

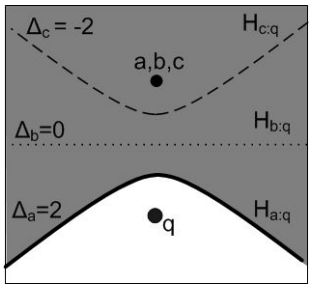


Fig. 12 Hyperbola based pruning

Example 3 Fig. 12 shows four facilities q, a, b and c. To illustrate the properties of hyperbola based pruning, the three facilities (a to c) are assumed to have the same location. First, we show the space pruned using the facility a. Assume that $q[price] = 4$, $a[price] = 2$, and $w[price] = w[distance] = 0.5$. Hence, $\Delta_a = \frac{2-1}{0.5} = 2$. Fig. 12 shows the area that can be pruned using the hyperbola for the facility a (see the shaded area).

Now consider the facilities b and c that have the same location as the facility a and assume that $b[price] = 4$ and $c[price] = 6$ (i.e., $\Delta_b = 0$ and $\Delta_c = -2$). The hyperbola $H_{b:q}$ in this case is defined by the perpendicular bisector similar to the traditional traditional RkNN queries (see the half-space that lies above the dotted line in Fig. 12). This is because the facility b and the query have equal static scores (i.e., $\Delta_b = 0$) and their final scores are decided solely based on distances. The hyperbola $H_{c:q}$ prunes the half-space that lies above the hyperbola shown in the broken line. Note that this example shows that the facility that has smaller static score (i.e., larger Δ_f) prunes larger area, e.g., the facility a prunes a larger area than the facilities b and c. This is intuitive because the facilities that have better static attributes are expected to prune more users.

Extending TPL to answer reverse top-k queries. Extending InfZone [7] and FINCH [40] using the hyperbola based pruning is non-trivial mainly because computing the unpruned polygon (or influence zone) using the hyperbolas is non-trivial. However, TPL [28] can be easily extended to answer reverse top-k queries by using hyperbolas for pruning instead of the perpendicular bisectors. As shown in the example above, the facilities that have smaller static scores prune larger area. Also, the facilities that are closer to the query point are expected to prune larger area. Hence, the TPL algorithm accesses the facilities in increasing order of $w[d+1] \cdot dist(q, f) + f_s$. The rest of the details are similar to the original TPL algorithm except two possible optimizations briefly described below.

We note that there are some problem characteristics specific to reverse top-k queries that can be exploited to further improve the performance. Firstly, we observe that there may be some queries that cannot have any reverse top-k user regardless of the users' locations. Such queries are called *futile* queries. For the futile queries, the algorithm can return empty results without even accessing the user R*-tree. Secondly, there may be some un-necessary facilities that can be completely ignored to correctly compute the results. In Section 4.2.2, we present observations to identify if a query is futile or not. If a query is futile, the algorithm terminates and returns empty results. In Section 4.2.3, we show how to identify the un-necessary facilities that can be ignored by the algorithm to improve the performance.

4.2.2 Identifying a futile query

We say that a facility f prunes the whole data space if it prunes every point in the data space. The next lemma identifies the facilities that prune the whole data space.

Lemma 6 *A facility f prunes the whole data space if $dist(f, q) < \Delta$.*

Proof Let u be an arbitrary user located *anywhere* in the data space. Due to the triangular inequality, $dist(f, u) - dist(q, u) \leq dist(f, q)$. Since $dist(f, q) < \Delta$, we have $dist(f, u) - dist(q, u) < \Delta$. Hence, u is pruned by f (Corollary 1). □

Example 4 Consider the example of Fig. 11 where $dist(q, f) = 5$ and $\Delta = 6$. Regardless of the location of u, every user u can be pruned by f. This is intuitive because if two facilities are *reasonably* close to each other and one facility (say f) has a *significantly* better static score than the other (say q) then every user will prefer f over q.

Note that a query q cannot have any reverse top-k users if there are at least k facilities that satisfy Lemma 6 (i.e., $dist(f, q) < \Delta$ for each such facility). The algorithm maintains a counter that records the number of facilities that prune the whole data space. The algorithm terminates and returns empty results if the counter reaches k.

Next, we extend Lemma 6 to an entry of the facility R*-tree. First, we extend the notion of static score gap Δ for an entry of the facility R*-tree. Let e be an entry of the facility R*-tree and e_s^{max} and e_s^{min} be the maximum and minimum static scores (as defined in Section 4.1), respectively. The following two equations define the maximum static score gap Δ_e^{max} and minimum static score gap Δ_e^{min} between q and e.

$$\Delta_e^{max} = \frac{q_s - e_s^{min}}{w[d+1]} \tag{4}$$

$$\Delta_e^{min} = \frac{q_s - e_s^{max}}{w[d+1]} \tag{5}$$

It is easy to show that for every facility $f \in e$, $\Delta_e^{min} \leq \Delta_f \leq \Delta_e^{max}$. The next two lemmas extend Lemma 6 for an entry e of the facility R*-tree.

Lemma 7 *Every facility $f \in e$ prunes the whole data space if $maxdist(q,e) < \Delta_e^{min}$.*

Proof Since $f \in e$, $dist(f,q) \leq maxdist(q,e)$ and $\Delta_f \geq \Delta_e^{min}$. Hence, $dist(f,q) < \Delta_f$ which implies that f prunes every user u (Lemma 6). □

Let $|e|$ denote the number of facilities indexed in the subtree rooted at e. If e satisfies the condition in Lemma 7 then at least $|e|$ facilities prune the whole data space. In this case, the algorithm increments the counter by $|e|$ and ignores the entry e (i.e., its children are not accessed).

Lemma 8 *There exists at least one facility $f \in e$ that prunes the whole data space if $MinMaxDist(q,e) < \Delta_e^{min}$ where $MinMaxDist(q,e)$ is the upper bound on the minimum $dist(f,q)$ for every $f \in e$ as defined in [23].*

Proof By definition of $MinMaxDist(q,e)$, there exists a facility f for which $dist(q,f) \leq MinMaxDist(q,e)$. Since $\Delta_f \geq \Delta_e^{min}$ for every facility $f \in e$, we have $dist(q,f) < \Delta_f$ which implies that f prunes the whole data space. □

If there is an entry e that satisfies the condition in Lemma 8 the algorithm increments the counter by 1 and its children are inserted in the heap to be processed later.

Note that applying Lemma 8 and Lemma 6/7 together may incorrectly update the counter. For example, assume that an entry e has only two facilities f_1 and f_2 and e satisfies Lemma 8, f_1 satisfies Lemma 6 and f_2 does not satisfy the condition of Lemma 6. The correct counter after processing the entry e is one because there is only one facility that prunes the whole data space. However, when e is processed the counter is incremented by one. When its child f_1 is accessed the counter is incremented again which is incorrect. We avoid this issue as follows.

We maintain a list that records the entries that satisfy Lemma 8 and the list is kept sorted on IDs of the entries for logarithmic search. If a facility f or an entry e satisfies one of the Lemmas 6, 7 and 8, we check whether its parent e' is present in the list or not. If its parent e' is not present in the list, the algorithm increments the counter as described above (depending on which of the three lemmas are applicable). However, if its parent e' is found in the list then the counter is first decremented by one and then incremented accordingly (depending on the applicable lemma). The counter is decremented by one to avoid incorrectly incrementing the counter twice for the same facility. The parent e' is then deleted from the list because the counter has been correctly updated (and this should not be repeated for the other children of e').

4.2.3 Discarding un-necessary facilities

We observe that there may be some facilities (and the entries of facility R*-tree) that are not required to compute the results, i.e., the results can be correctly computed without considering these facilities. In this section, we present observations to identify such facilities. A facility f, for which $score(u,q) \leq score(u,f)$ for every user u (regardless of the location of u), can be safely discarded because f cannot prune *any* user u. Such a facility f is called an *un-necessary* facility. The next lemma identifies un-necessary facilities.

Lemma 9 *A facility f is an un-necessary facility if $dist(f,q) \leq -\Delta$.*

Proof Let u be an arbitrary user. Due to the triangular inequality, $dist(q,u) - dist(f,u) \leq dist(f,q)$. Since $dist(q,f) \leq -\Delta$, we have $dist(q,u) - dist(f,u) \leq -\Delta$. In other words, $dist(q,u) - dist(f,u) \leq -\frac{(q_s - f_s)}{w[d+1]}$. Hence, $w[d+1] \cdot dist(q,u) + q_s \leq w[d+1] \cdot dist(f,u) + f_s$. Hence, $score(u,q) \leq score(u,f)$, i.e., f is an un-necessary facility. □

Next, we extend this lemma for an entry e of the facility R*-tree.

Lemma 10 *Every facility $f \in e$ is an un-necessary facility if $maxdist(q,e) \leq -\Delta_e^{max}$.*

Proof For every $f \in e$, $dist(q,f) \leq maxdist(q,e)$ and $\Delta_f \leq \Delta_e^{max}$. Hence, $dist(q,f) \leq -\Delta_f$ for every $f \in e$. Hence, f is an un-necessary facility (as per Lemma 9). □

During the execution of the algorithm, any facility entry e that satisfies the condition in Lemma 10 is pruned.

4.3 SLICE: A Regions-based Solution

In this section, we extend SLICE to efficiently answer the spatial reverse top-k queries. Our proposed solution follows the same framework as SLICE for RkNN queries. However, the underlying techniques (e.g., how to compute upper arc and significant facilities etc.) are different. In Section 4.3.1, we present our regions-based pruning techniques to prune the search space using a facility f. Section 4.3.2 presents the techniques to identify significant facilities. Finally, we present the overall algorithm in Section 4.3.3.

4.3.1 Computing Upper, Lower and Bounding Arcs

As described in the previous section, a facility f prunes the whole data space if $dist(f,q) < \Delta$ (Lemma 6) and f does not prune any point if $dist(f,q) \leq -\Delta$ (Lemma 9). Next, we present regions-based pruning rules for the facilities for which $dist(q,f) > |\Delta|$ where $|\Delta|$ is the absolute value of Δ. Throughout this section, we limit our discussion to the facilities for which $dist(q,f) > |\Delta|$.

We divide the search space into t equally sized partitions (just like SLICE does for RkNN queries). In this section,

we define the area of a partition P_i that can be pruned by a given facility f. First, we present an observation to prune the search space on a *ray*. Then, we extend it to prune the space in a partition.

Consider a ray originating from q (see the arrow in Fig. 13). Let $\theta = \angle fqx$ for any point x on this ray. We denote this ray as L^θ. Next, we introduce the concept of critical point and critical distance for a ray.

Definition 7 Critical point/distance. We say that a point p^θ is a critical point on L^θ if the score of f and q are the same for a user that is located at p^θ, i.e., $score(f, p^\theta) = score(q, p^\theta)$. The distance $dist(q, p^\theta)$ is called the critical distance and is denoted as d^θ.

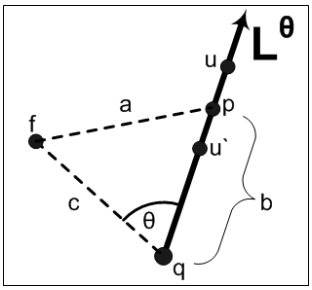


Fig. 13 Critical point p

The next lemma identifies the critical point/distance on a ray L^θ.

Lemma 11 *Consider a facility f and a query q such that $dist(q, f) \geq |\Delta|$. For a ray L^θ, the critical distance is $d^\theta = \frac{dist(q,f)^2 - \Delta^2}{2(\Delta + dist(q,f)\cos\theta)}$.*

Proof We abuse the notation and use p to denote p^θ, the critical point on L^θ. Considering the triangle $\triangle pqf$ (with side lengths a, b and c) in Fig. 13, we compute $a = dist(p, f)$.

$$a^2 = b^2 + c^2 - 2bc\cos\theta \tag{6}$$

Recall that the scores of q and f are equal for a user u if $dist(f, u) - dist(q, u) = \Delta$. Since p is the critical point, we have $dist(f, p) - dist(q, p) = \Delta$, i.e., $a - b = \Delta$. In Eq. (6), we replace a with $\Delta + b$

$$(\Delta + b)^2 = b^2 + c^2 - 2bc\cos\theta$$

$$\Delta^2 + b^2 + 2b\Delta = b^2 + c^2 - 2bc\cos\theta$$

$$2b\Delta + 2bc\cos\theta = c^2 - \Delta^2$$

$$b(2\Delta + 2c\cos\theta) = c^2 - \Delta^2$$

$$b = d^\theta = \frac{c^2 - \Delta^2}{2(\Delta + c\cos\theta)} = \frac{dist(q,f)^2 - \Delta^2}{2(\Delta + dist(q,f)\cos\theta)} \tag{7}$$

□

Eq. (7) can be used to to compute the critical distance d^θ for a facility f for which $dist(q, f) > |\Delta|$. We remark that it is not necessary that every ray L^θ has a critical point on it, i.e., it is possible that, for a ray L^θ, there does not exist any point for which f and q have the same scores. Specifically, it can be shown that a ray L^θ, for which $\Delta + dist(q, f)\cos\theta \leq 0$, does not contain any critical point. Such rays are called invalid rays and we assume $d^\theta = \infty$ for every invalid ray. Next, we formally define a valid ray.

Valid ray. A ray L^θ is called valid if $\Delta + dist(q, f)\cos\theta > 0$. This implies that a ray L^θ is valid if $\theta < cos^{-1}(\frac{-\Delta}{dist(q,f)})$. The next lemma shows that a user u lying on a valid ray L^θ can be pruned if $dist(u, q) > d^\theta$ (e.g., u in Fig. 13 can be pruned).

Lemma 12 *A user u that lies on a valid ray L^θ and has $dist(u, q) > d^\theta$ is pruned by f, i.e. $score(u, f) < score(u, q)$.*

Proof For the critical point p on L^θ, we know that $score(p, f) = score(p, q)$ which implies that $dist(p, f) - dist(p, q) = \Delta$. We add $dist(u, p)$ to both of the terms on the left hand side of this equation which gives

$$dist(p, f) + dist(u, p) - (dist(p, q) + dist(u, p)) = \Delta$$

Note that $dist(p, q) + dist(u, p) = dist(u, q)$ (see Fig. 13). On the other hand, $dist(p, f) + dist(u, p) > dist(u, f)$ (due to the triangular inequality[2]). Hence, $dist(u, f) - dist(u, q) < \Delta$. Thus, according to the inequality (3), $score(u, f) < score(u, q)$. □

Lemma 13 *A user u' that lies on a ray L^θ and has $dist(u', q) \leq d^\theta$ cannot be pruned by f, i.e., $score(u', f) \geq score(u', q)$.*

Proof For the critical point p on L^θ, we know that $dist(p, f) - dist(p, q) = \Delta$. We subtract $dist(u', p)$ from both terms in the above equation which yields

$$(dist(p, f) - dist(u', p)) - (dist(p, q) - dist(u', p)) = \Delta$$

Note that $dist(p, q) - dist(u', p) = dist(u', q)$ (see Fig. 13). On the other hand, $dist(p, f) - dist(u', p) \leq dist(u', f)$ (due to triangular inequality). Hence, $dist(u', f) - dist(u', q) \geq \Delta$. Hence, $score(u', f) \geq score(u', q)$. □

Next, we present a property of the critical distance that is utilized later in our pruning techniques.

[2] Lemma 20 in appendix shows that $dist(u, f)$ can never be equal to $dist(f, p) + dist(u, p))$ even when f lies on the ray L^θ.

Property 1 d^θ **increases as** θ **increases.** Note that $\theta \leq 180°$ for any ray originating from q. Since $\cos\theta$ monotonically decreases as θ increases from 0 to $180°$ and $dist(q,f)\cos\theta + \Delta > 0$ for valid rays, it is easy to see from Eq. (7) that d^θ increases as θ increases.

Based on Property 1, the next lemma defines the area of a partition P that can be pruned by a facility f.

Lemma 14 *Let* $\theta_{max} = maxAngle(f,P)$ *for a partition* P *and* f *(see Definition 2) and* $\theta_{max} < cos^{-1}(-\frac{\Delta}{dist(q,f)})$. *Every user* $u \in P$ *can be pruned if* $dist(u,q) > d^{\theta_{max}}$, *i.e.,* $dist(u,q) > \frac{dist(q,f)^2-\Delta^2}{2(\Delta+dist(q,f)\cos\theta_{max})}$.

Proof Fig. 14 shows a user u that lies in the partition P and $dist(u,q) > d^{\theta_{max}}$. Let α be the subtended angle of u, i.e., $\alpha = \angle fqu$. Since $\alpha \leq \theta_{max}$, $d^\alpha \leq d^\theta_{max}$ (Property 1). Hence, $dist(u,q) > d^\alpha$. Furthermore, since $\alpha \leq \theta_{max} < cos^{-1}(-\frac{\Delta}{dist(q,f)})$, the ray L^α is a valid ray. According to Lemma 12, u can be pruned by f. □

Based on Lemma 14, we describe how to compute the upper arc for reverse top-k queries.

Definition 8 Upper arc. Upper arc of a facility f w.r.t. a partition P is the arc centered at q with radius $d^{\theta_{max}}$ where $\theta_{max} = maxAngle(f,P)$ and $\theta_{max} < \cos^{-1}(-\frac{\Delta}{dist(q,f)})$. 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 $\theta_{max} \geq \cos^{-1}(-\frac{\Delta}{dist(q,f)})$, $r^U_{f:P} = \infty$.

Fig. 14 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 14, a facility prunes area outside the upper arc of f for every partition P for which $\theta_{max} < \cos^{-1}(-\frac{\Delta}{dist(q,f)})$. In Fig. 14, every user that lies in the shaded area can be pruned by f.

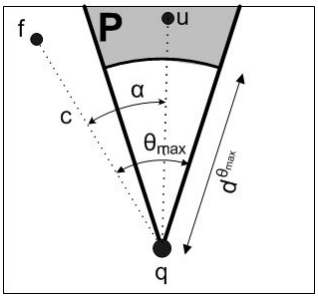


Fig. 14 Pruning using upper arc

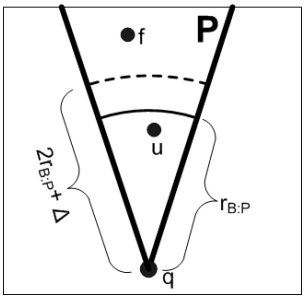


Fig. 15 f is not significant

Recall that a user u cannot be the reverse top-k (i.e., it can be filtered), if it is pruned by at least k facilities. In other words, a user u can be filtered if it lies outside the upper arcs of at least k facilities. To efficiently employ this filtering rule, we maintain k-th smallest upper arc in each partition which is called the bounding arc and is defined below.

Definition 9 Bounding arc. The k-th smallest upper-arc of a partition P is called its bounding arc. Its radius is denoted as $r_{B:P}$ (or simply r_B when the partition is clear by context). If there are less than k upper arcs in a partition P then $r_{B:P} = \infty$.

Next, we describe how to compute lower arc of a partition P w.r.t. a facility f.

Lemma 15 *Let* $\theta_{min} = minAngle(f,P)$ *for a partition* P *and* f *(see Definition 2) and* $\theta_{min} < cos^{-1}(-\frac{\Delta}{dist(q,f)})$. *The facility* f *cannot prune any user* $u \in P$ *for which* $dist(u,q) < d^{\theta_{min}}$.

Proof Let α be the subtended angle of u. Since $u \in P$, $\alpha \geq \theta_{min}$. Hence, $d^\alpha \geq d^\theta_{min}$ (Property 1). Since $dist(u,q) < d^\theta_{min}$, we have $dist(u,q) < d^\alpha$. According to Lemma 13, u cannot be pruned by f. □

Definition 10 Lower arc. Lower arc of a facility f w.r.t. a partition P is the arc centered at q with radius $d^{\theta_{min}}$ where $\theta_{min} = minAngle(f,P)$ and $\theta_{min} < \cos^{-1}(-\frac{\Delta}{dist(q,f)})$. The radius of 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). If $\theta_{min} \geq \cos^{-1}(-\frac{\Delta}{dist(q,f)})$, $r^L_{f:P} = \infty$.

Algorithm 4: pruneSpace(f)

```
1 if dist(f,q) > |Δ| then
2   θ ← cos^{-1}(−Δ / dist(f,q));
3   for each partition P for which minAngle(f,P) < θ do
4     if maxAngle(f,P) < θ then   // otherwise r^U_{f:P} is ∞
5       r^U_{f:P} ← d^{maxAngle(f,P)};
6       Set r_{B:P} to the radius of k-th smallest upper arc of P;
7     if f is a significant facility then
8       r^L_{f:P} ← d^{minAngle(f,P)};
9       insert f in sigList of P in sorted order of r^L_{f:P};
```

Whenever we access a facility f, we prune the search space by computing its upper arc for the relevant partitions and updating the bounding arcs if required. Algorithm 4 shows the details of pruning the search space using a facility f. Since upper-arc can only be computed if $dist(q,f) > |\Delta|$, the algorithm first ensures that $dist(q,f) > |\Delta|$ (line 1). Then, for each partition P for which $minAngle(f,P) < cos^{-1}(-\frac{\Delta}{dist(q,f)})$, the algorithm computes the upper-arc $r^U_{f:P}$ and updates the bounding arc if required (lines 3 to 6).

The lines 7 to 9 are used to insert *lower arcs* of significant facilities that are identified as described in the next section. As we will show later, this significantly improves the verification phase of the algorithm. For now, the readers can ignore these lines.

4.3.2 Identifying Significant Facilities

Recall that a facility f is called a significant facility of a partition P if it prunes at least one point $p \in P$ that lies in the bounding arc of P, i.e., $score(p, f) < score(p, q)$ for at least one $p \in P$ for which $dist(p, q) \leq r_{B:P}$. In this section, we describe techniques on how to determine whether an entry e may contain a significant facility or not. The next lemma identifies a facility that is not a significant facility for a partition P.

Lemma 16 *A facility f is an insignificant facility for a partition P if f lies inside the partition P and $dist(q, f) > 2r_{B:P} + \Delta$.*

Proof Consider the example of Fig. 15 that shows a partition P, a facility f, the bounding arc $r_{B:P}$ (shown in solid line), and an arc with radius $2r_{B:P}+\Delta$ (shown in broken line). We prove that f is not significant by showing that $score(u, f) \geq score(u, q)$ for *every* user in P that lies inside the bounding arc, i.e., for which $dist(u, q) \leq r_{B:P}$.

Fig. 15 shows a user u that lies inside the bounding arc. Due to the triangular inequality, $dist(u, f) \geq dist(q, f) - dist(u, q)$. Since $dist(q, f) > 2r_{B:P} + \Delta$ and $dist(u, q) \leq r_{B:P}$, we have $dist(u, f) \geq (2r_{B:P} + \Delta) - r_{B:P}$. This can be rewritten as $dist(u, f) - r_{B:P} \geq \Delta$. Since $dist(u, q) \leq r_{B:P}$, we have $dist(u, f) - dist(u, q) \geq \Delta$. Hence, f cannot prune the user u (Corollary 1). □

Next, we extend this lemma for an entry e of the facility R*-tree.

Lemma 17 *Every facility $f \in e$ is an insignificant facility for a partition P if f lies inside the partition P and $mindist(q, e) > 2r_{B:P} + \Delta_e^{max}$.*

Proof For every $f \in e$, $\Delta_f \leq \Delta_e^{max}$ and $dist(q, f) \geq mindist(q, e)$. Hence, we have $dist(q, f) > 2r_{B:P} + \Delta_f$ for every $f \in e$. Hence, according to Lemma 16, f cannot be a significant facility. □

The above two lemmas define the conditions to identify whether a facility f that lies inside a partition P is a significant facility or not. The next lemma defines the condition to determine whether a facility that lies outside a partition P is a significant facility of the partition P or not.

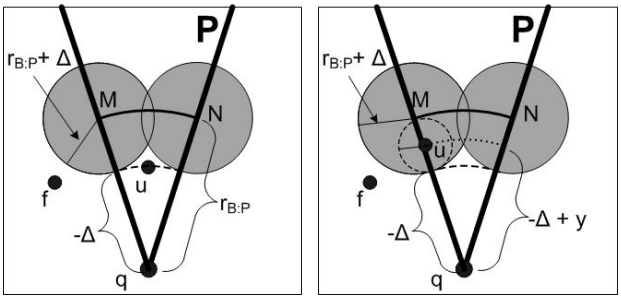
Lemma 18 *Let M and N be the points where the bounding arc intersects with the partition P (as shown in Fig. 16). A facility $f \notin P$ cannot be a significant facility if $dist(f, M) > r_{B:P} + \Delta$ **and** $dist(f, N) > r_{B:P} + \Delta$, i.e., if f lies outside the two shaded circles of Fig. 16.*

Proof To prove that f is not a significant facility, we show that, for every user $u \in P$ for which $dist(u, q) \leq r_{B:P}$, $score(u, f) \geq score(u, q)$. We prove this for the following two cases: **case 1)** $\Delta \leq 0$ (see Fig. 16); **case 2)** $\Delta > 0$ (see Fig. 17).

Case 1: $\Delta \leq 0$. We divide this further into two cases.

Case 1a: $dist(u, q) \leq -\Delta$ (see Fig. 16(a)). If $dist(u, q) \leq -\Delta$, the maximum (i.e., worst) possible score of q is when u is furthest from q, i.e., $dist(u, q) = -\Delta$. In other words, $score(u, q) \leq q_s - w[d+1] \cdot \Delta$ or $score(u, q) \leq q_s - w[d+1]\frac{q_s - f_s}{w[d+1]}$. Hence, $score(u, q) \leq f_s$. Since $score(u, f) \geq f_s$, $score(u, q) \leq score(u, f)$ which completes the proof for this case.

Case 1b: $dist(u, q) > -\Delta$. Without loss of generality, assume that $dist(u, q) = -\Delta + y$ (see Fig. 16(b)). The score of q is $score(u, q) = q_s + w[d+1](-\Delta + y) = q_s - w[d+1]\frac{q_s - f_s}{w[d+1]} + w[d+1] \cdot y$. In other words, $score(u, q) = f_s + w[d+1] \cdot y$. Since $score(u, f) = f_s + w[d+1] \cdot dist(u, f)$, we prove that $score(u, f) \geq score(u, q)$ by showing that $dist(u, f) \geq y$.



(a) Case 1a: $dist(u, q) \leq -\Delta$ (b) Case 1b: $dist(u, q) > -\Delta$

Fig. 16 Lemma 18, Case 1 ($\Delta \leq 0$)

Since $dist(u, q) = -\Delta + y$, u lies somewhere on the dotted arc in Fig. 16(b). Since f lies outside the partition P, $dist(u, f)$ is minimum when u lies on one of the end points of the dotted arc (see Observation 1 in [5] [3]). Without loss of generality, assume that u lies on the line connecting q and M (as shown in Fig. 16(b)). We draw a circle centered at u with radius y (see the small circle in Fig. 16(b) drawn in broken line) and show that f lies outside this circle (i.e, $dist(u, f) > y$).

Since $dist(u, q) \leq r_{B:P}$ (because u lies within bounding arc) and $dist(u, q) = -\Delta + y$, we have $r_{B:P} \geq -\Delta + y$ which implies $r_{B:P} + \Delta \geq y$. Since the radius of the circle centered at M is $r_{B:P} + \Delta$ and $y \leq r_{B:P} + \Delta$, the circle centered at M completely contains the circle centered at u. Since f lies outside the circle centered at M, it also lies out-

[3] The observation 1 in [5] can be summarized as follows. For a circle C centered at q, let x be the point such that $dist(f, x) = mindist(f, C)$ where $mindist(f, C)$ is the minimum distance between f and any point of the circle C. Let u be a user located on the perimeter of this circle. Then, $dist(f, u)$ monotonically increases if u moves along the circle in clockwise or counter clockwise direction from x. Since f lies outside the partition, x also lies outside the partition and this implies that $dist(u, f)$ is minimum when u is at one of the end points of the arc.

side the smaller circle centered at u. Hence, $dist(u, f) > y$ which completes the proof for this case.

Case 2. $\Delta > 0$ (see Fig. 17). Assume that $dist(u, q) = y$, e.g., u lies on the dotted arc in Fig. 17. As argued earlier, the minimum distance from f to u is when u lies on one of the end points of this arc. Without loss of generality, assume that u lies on the line connecting q and M. We draw a circle centered at u with radius $y + \Delta$ (see the circle in Fig. 17 shown in broken line). Since the circle centered at M has a radius $r_{B:P} + \Delta$ and $y \leq r_{B:P}$, it is easy to see that the circle centered at u (the broken circle) is contained by the circle centered at M. Hence, $dist(u, f) > y + \Delta$ because f lies outside of the circle centered at M. So, the score of f is $score(u, f) > f_s + w[d + 1] \cdot (y + \Delta)$ or $score(u, f) > f_s + w[d+1] \cdot y + w[d+1] \frac{q_s - f_s}{w[d+1]}$ which gives $score(u, f) > q_s + w[d+1] \cdot y$. Since $dist(u, q) = y$, the right hand side of this inequality is equal to $score(u, q)$. Hence, $score(u, f) > score(u, q)$ which completes the proof for this case. □

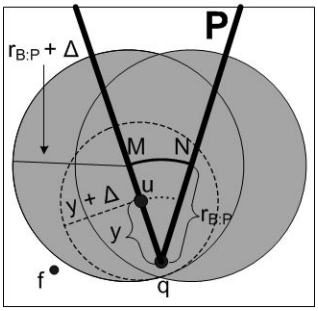


Fig. 17 Lemma 18, Case 2

The next lemma extends Lemma 18 for an entry e of the facility R*-tree.

Lemma 19 *Let M and N be the points where the bounding arc intersects with the partition P (as shown in Fig 16). Every facility $f \in e$ is an insignificant facility if $f \notin P$ and $mindist(e, M) > r_{B:P} + \Delta_e^{max}$ **and** $mindist(e, N) > r_{B:P} + \Delta_e^{max}$.*

Proof Since $f \in e, dist(f, M) \geq mindist(e, M) > r_{B:P} + \Delta_e^{max}$ and $dist(f, N) \geq mindist(e, N) > r_{B:P} + \Delta_e^{max}$. Furthermore, for every $f \in e$, $\Delta_f \leq \Delta_e^{max}$. Therefore, $dist(f, M) > r_{B:P} + \Delta_f$ and $dist(f, N) > r_{B:P} + \Delta_f$. Hence, according to Lemma 18, f cannot be a significant facility. □

4.3.3 Algorithm

Now, we are ready to present the algorithm to answer spatial reverse top-k queries.

Filtering Algorithm. Algorithm 5 shows the details of the filtering algorithm. The space is divided into t equal sized partitions. A list is introduced to store intermediate nodes and initialized to empty. A min-heap is initialized with the root entry of the facility R*-tree. Since the entries that have smaller static scores and are closer to the query are expected to prune a larger area, each entry e is inserted in the heap with key $\Delta_e^{max} + w[d + 1] \cdot mindist(e, q)$ (line 22). The entries are iteratively deheaped from the heap until the heap becomes empty. Each deheaped entry e is processed as follows.

Algorithm 5: Filtering

```
1  Divide the space around q in t equally sized partitions;
2  Initialize an empty list L;
3  Insert root of facility R*-tree in a min-heap h;
4  while h is not empty do
5      deheap an entry e;
6      if maxdist(e,q) < -Δ_e^max then // Lemma 10
7          continue;
8      if MinMaxdist(e,q) < Δ_e^min then // Lemma 8
9          if parent of e exists in L then
10             k ← k + 1;
11             remove parent of e from L;
12         if maxdist(e,q) < Δ_e^min then // Lemma 7
13             k ← k − |e|;
14             if k ≤ 0 then terminate algorithm and return empty;
15             continue;
16         else // satisfies conditions in Lemma 8
17             k ← k − 1;
18             insert e in L;
19         if k ≤ 0 then terminate algorithm and return empty;
20     if e may contain a significant facility for at least one partition
       then
           //Lemma 17 & 19 for each partition
21         if e is an intermediate node or leaf then
22             insert every child c of e into the heap h with key
               Δ_c^max + w[d + 1]mindist(q,c);
23         else                          // e is a data object
24             pruneSpace(e)                    //Algorithm 4
```

In lines 6 to 19, we apply the observations presented in Section 4.2.2 and Section 4.2.3. Below, we provide a quick overview and the readers are referred back to these sections for a detailed explanation of the ideas. If every facility $f \in e$ is an un-necessary facility (see Section 4.2.3), e is discarded (line 7). The algorithm then checks whether the entry e contains one or more facilities that prune the whole data space by using Lemmas 7 and 8 presented in Section 4.2.2. Note that an entry e can satisfy Lemma 7 only if it satisfies the condition in Lemma 8. Therefore, we first check if the condition in Lemma 8 is satisfied (line 8). To correctly update the value of k, we first increment k by 1 if the parent of e is present in the list L (line 10) and then delete its parent from L (line 11). If all the facilities in f prune the whole data space (i.e., e satisfies Lemma 7), then the value of k is decremented by $|e|$. In this case, the entry can be discarded because its children are not needed to be accessed (line 15). On the other hand, if the entry e satisfies Lemma 8 but does not satisfy the condition in Lemma 7, then the value of k is decremented by one because there exists at least one facil-

ity that prunes the whole data space (line 17). Such an entry e is inserted in L to correctly update the value of k later (line 18). At any stage, if the value of k is not greater than zero (lines 14 and 19), the algorithm returns empty because the query is a futile query.

The rest of the algorithm (lines 20 to 24) is quite similar to the filtering algorithm (Algorithm 1) for RkNN queries. Specifically, if an entry e may contain a significant facilities, it is processed as follows. If e is an intermediate node or leaf node, its children are inserted in the heap. However, if e is an object (i.e., a facility), it is used to prune the search space by using Algorithm 4.

Verification Algorithm. The verification algorithm is quite similar to the verification algorithm for RkNN queries (Algorithm 3). In the verification phase, the users that cannot be filtered are shortlisted and are called candidate users. This is done by iteratively accessing the nodes of the user R*-tree and checking whether the node can be pruned by upper arcs of the relevant partitions or not. If not, its children are inserted in a stack to be iteratively accessed later. If the accessed entry is a data object (e.g., a user) and cannot be filtered, it is a candidate object and is verified by calling Algorithm 6.

Algorithm 6: isReverseTopk(u)

```
   Output : Returns true if u is reverse top-k. 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 r^L_{f:P} > dist(u, q) then
5          return true;
6      if score(u, f) < score(u, q) then
7          counter ← counter + 1;
8          if counter ≥ k then
9              return false;
10 return true;
```

Algorithm 6 presents the details of how to check if a candidate user u is an answer or not. It first determines the partition P in which the candidate user u lies. The algorithm also initializes a counter to zero that records the number of facilities that have a better score than q. Then, it iteratively accesses the facilities in the *sigList* of the partition P in ascending order of $r^L_{f:P}$. For each accessed facility f, the algorithm increments the counter if $score(u, f) < score(u, q)$. If the counter reaches k, the user u cannot be the reverse top-k and the algorithm returns false (see lines 6 to 9). If the counter does not reach k and all the facilities in the *sigList* have been accessed the algorithm returns true indicating that u is an answer (line 10). At any stage, if $r^L_{f:P}$ of the next accessed facility f is larger than $dist(u, q)$, the algorithm returns true (line 5), i.e., u is an answer. This is because 1) f cannot prune u because $r^L_{f:P} > dist(u, q)$ (Lemma 15); and 2) since the algorithm accesses the facilities in ascending order of $r^L_{f:P}$, no remaining facility can prune the user u because for each such remaining facility f', $r^L_{f':P} > dist(u, q)$.

Parameter	Range
Number of facilities($\times$1000)	50, **100**, 150 200
Number of users($\times$1000)	50,**100**, 150 200
Value of k	1, 5, 10, **15**, 20, 25
Data distribution	North America, **Normal**, Uniform
Buffer size(Six-region only)	2, 5, **10**, 20, 40, 100

Table 4 Experimental settings (reverse k nearest neighbors queries)

5 Experimental Evaluation

5.1 Reverse k Nearest Neighbors Queries

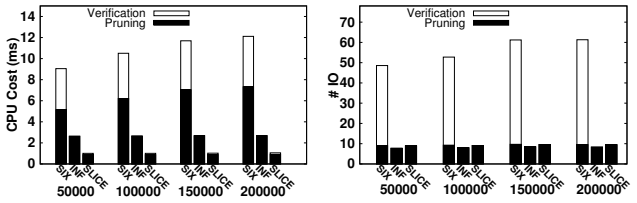
5.1.1 Experimental settings

We compare our algorithm with six-regions approach [26] and InfZone [7] which is the state-of-the-art RkNN algorithm. All algorithms are implemented 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 [7] by our main competitor InfZone. Table 4 shows the detailed settings and the default values are shown in bold. Specifically, we use both the synthetic and real data sets. The real data set consists of $175,812$ points in North America [33] 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 to 200000 points following either Uniform or Normal distribution. The default synthetic data set consists of 100000 points and follows Normal distribution unless mentioned otherwise. We vary k from 1 to 25 and the default value of k is 15. For bichromatic RkNN queries, the number of users is the same as the number of facilities unless specifically mentioned.

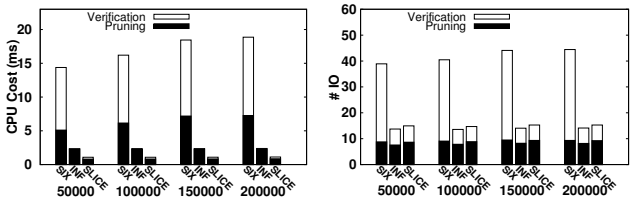
For six-regions approach, a buffer of 10 pages is used which uses random eviction strategy. We remark that InfZone and SLICE do not require buffer because each node is accessed only once by these algorithms. Thus, the buffer favors the six-regions approach (see Fig. 24 for more details).

Next, we compare the performance of the three algorithms for both the monochromatic and bichromatic RkNN queries. Six-regions [26] is shown as **SIX** and InfZone [7] 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 experiments presented in our earlier work [44]. The results reported in the figures correspond to the average cost per query.



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

Fig. 18 Monochromatic queries: effect of data set size (Normal Distribution)



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

Fig. 19 Bichromatic queries: effect of data set size (Normal Distribution)

5.1.2 Effect of data size

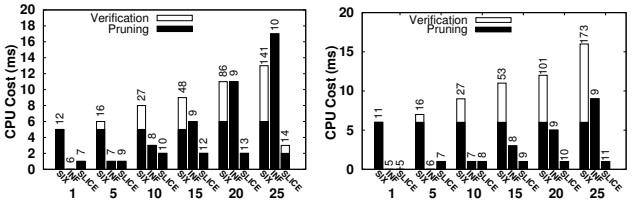
In Fig. 18 and 19, we study the effect of data size for monochromatic and bichromatic RkNN queries, respectively. For bichromatic queries, the number of users in each data set is the same as the number of facilities. Fig. 18(a) and 19(a) show the CPU cost of the three algorithms. Note that the dominant cost for SLICE and InfZone is the pruning phase - the verification phase has negligible cost. This is due to the effective verification techniques employed by these two algorithms.

Fig. 18(b) and 19(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 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 [30] 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.

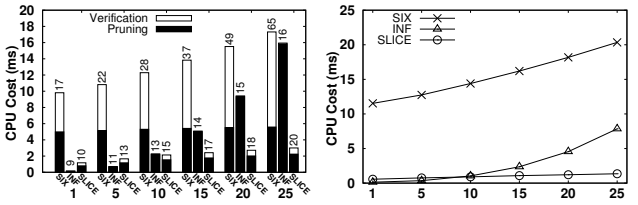
5.1.3 Effect of k

In Fig. 20 and 21, 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. 20 and 21. 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. 20 Monochromatic queries: effect of k



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

Fig. 21 Bichromatic 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 scales much better than the other two algorithms and is several times more efficient for larger k. In Fig. 21(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.

5.1.4 Effect of data distribution

In Fig. 22, 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. 22 demonstrates that SLICE is significantly faster than the other two algorithms for all different combinations of data distribution.

5.1.5 Effect of number of users

In this experiment, we fix the number of facilities to $100,000$ and change the number of users to see the effect of change

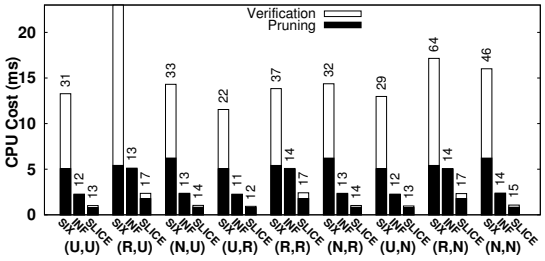


Fig. 22 Effect of data distribution

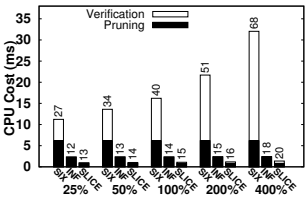


Fig. 23 % of users

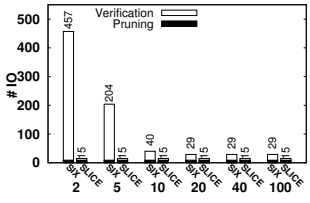


Fig. 24 Effect of buffer size

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. 23 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.

5.1.6 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. 24, 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.

5.2 Spatial Reverse Top-k Queries

5.2.1 Experimental Settings

Data sets and parameters. We use both synthetic and real data sets. The real data set consists of $1,000,000$ locations in California [32]. Each facility location is assigned up to 3 static attributes that are obtained from House data set [31] namely monthly owner costs, electricity cost, and property taxes. We normalize the static attributes into a range of 0 to 1 and assume that the smaller values are preferred. For default synthetic data sets, we generate locations following uniform distribution or normal distribution. Due to the space limitation, we only show the results for the uniform distribution - trends are similar for the normal distribution. The static attributes of the synthetic data sets are generated following uniform distribution. The default size of synthetic data sets contains $100,000$ facilities and the number of static attributes varies from 1 to 3. The size of synthetic user data sets varies from $25,000$ to $400,000$. Unless mentioned otherwise, the number of users are the same as the number of facilities in each experiment. The cost shown in the figures corresponds to average query cost. Table 5 summarizes the data sets and parameters used in the experiments.

Parameter	**Range**
Number of facilities($\times 1000$)	100, **1,000**
Number of users($\times 1000$)	25, 50, **100**, 200, 400, 1,000
Number of static attributes	1, **2**, 3
Value of k	1, 5, **10**, 15
Location data distribution	**California**, Uniform
Static data distribution	**House**, Uniform

Table 5 Experimental settings (spatial reverse top-k queries)

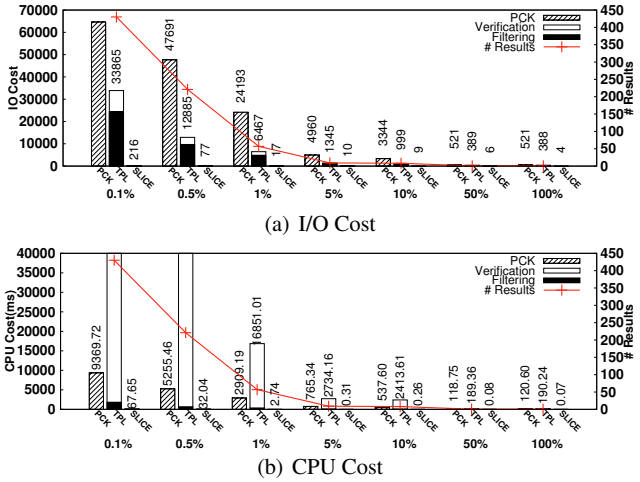
Competitors. We compare our algorithm with state-of-the-art spatial reverse top-k algorithm [21] (called PCK) and the extension of TPL that we presented in Section 4.2. For SLICE and TPL, filtering phase and verification phase are independent and the total cost is the sum of filtering cost and verification cost. For PCK, filtering and verification are blended and it is not possible to distinguish between the filtering cost and verification cost. Therefore, while we display the filtering and verification cost for TPL and SLICE, we only display the total cost for PCK.

Since PCK can only be applied for the settings when the number of static attributes is 1 (i.e., $d = 1$) and $k = 1$, we first compare our algorithm with both PCK and TPL for $k = 1$ on the data sets containing only one static attribute. Later, in Section 5.2.3, we compare our algorithm with TPL for $d \geq 1$ and $k \geq 1$.

5.2.2 Performance evaluation for $d = 1$ and $k = 1$

The default scoring function W is set as $w[1] = w[2] = 0.5$. Later, we show the results for varying the scoring function by increasing $w[2]$ from 0.1 to 0.9. As mentioned in Section 4.2.2, the results for futile queries are empty and the algorithm can be terminated as soon as the query is determined to be futile. This also implies that the processing time depends on the static attributes of queries. Specifically, a query that has better static score is likely to have more reverse top-k users and is expected to be more expensive. Therefore, we first evaluate the effect of query quality on the performance of the three algorithms.

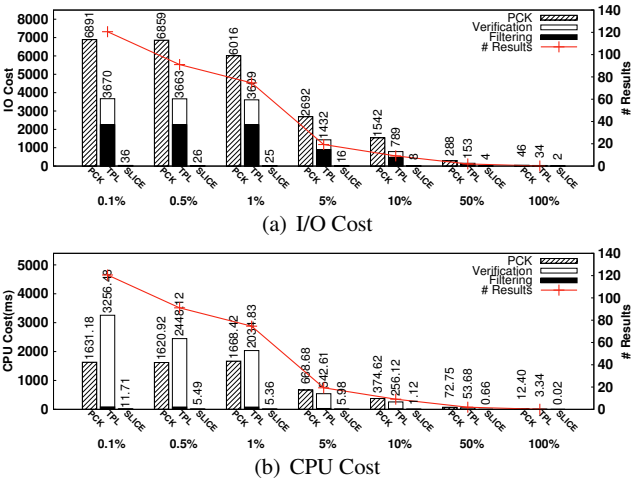
Effect of query quality. In Fig. 25 and Fig. 26, we evaluate the algorithms on different sets of queries where each query



(a) I/O Cost

(b) CPU Cost

Fig. 25 Effect of query quality (real data set)



(a) I/O Cost

(b) CPU Cost

Fig. 26 Effect of query quality (synthetic data set)

set contains 100 queries. Each query set is shown as $x\%$ denoting the quality of queries in the set. A query set corresponding to $x\%$ is selected as follows. First, all the facilities in the data set are sorted in ascending order of their static scores. Then, the first $x\%$ of these facilities are shortlisted. Among these $x\%$ facilities, 100 facilities are uniformly chosen and constitute the query set. For example, 0.1% query set on the real data set consisting of $1,000,000$ facilities is chosen by first shortlisting top-1000 facilities according to their static scores and then uniformly choosing 100 of these top-1000 facilities as queries. Note that the query set corresponding to 100% contains 100 queries uniformly selected from all the facilities. Also note that smaller x corresponds to better query quality, i.e., queries with better static scores.

As expected, Fig. 25 and Fig. 26 show that the query processing cost of the algorithms decreases as the query quality degrades. This is because the expected number of reverse top-k users for a query decreases as the query quality degrades. Fig. 25 and Fig. 26 also display the average number of reverse top-k users for each query set (shown in line). Note that the average number of results decreases from more than a hundred to almost zero as the query quality degrades. The average number of results in the real data set is larger than the average number of results in the synthetic data set mainly because real data set contains more users (1 Million) than the synthetic data set ($100,000$ users).

SLICE significantly outperforms the other two algorithms both in terms of I/O cost and CPU cost. PCK performs better than TPL in terms of CPU cost especially for high quality queries. However, TPL is better in terms of I/O cost. This is mainly because PCK needs to access the facility R*-tree multiple times to confirm if a user is a reverse top-k or not.

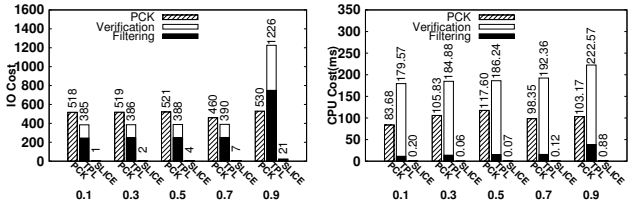
In the rest of the experiments, we display the results on two different types of query sets namely *high quality queries* and *mixed queries*. For *high quality queries*, $x = 5$, i.e., 100 queries are uniformly chosen from top-5% facilities according to their static scores. In *mixed queries*, we set $x = 100$.

Effect of scoring function. Next, we study the effect of the scoring function on real data set by changing the weight of distance criterion (i.e., $w[2]$) from 0.1 to 0.9 (recall $w[1] = 1 - w[2]$). Fig. 27 displays the results for mixed queries whereas Fig. 28 displays the results for high quality queries. SLICE consistently outperforms the other two algorithms by several orders of magnitude in terms of both the CPU cost and I/O cost.

Fig. 27 and Fig. 28 show that the CPU and I/O cost of all algorithms increase as the weight for dynamic attribute $w[2]$ increases. This is mainly due to the following. Recall that a facility f prunes the whole data space if $dist(f, q) < \Delta$ (Lemma 6) and a facility f is an un-necessary facility if $dist(f, q) < -\Delta$ (Lemma 9). Also, recall that $|\Delta| = \frac{|q_s - f_s|}{w[d+1]}$ and, for the case when there is only one static attribute, $|\Delta| = \frac{w[1] \cdot |q[1] - f[1]|}{w[2]}$. As $w[2]$ increases (and $w[1]$ decreases), $|\Delta|$ decreases. Consequently, there are fewer facilities on which Lemma 6 or Lemma 9 can be applied, i.e., there are fewer facilities that can prune the whole data space and there are fewer un-necessary facilities. As a result, the algorithm needs to process more facilities which results in an increased query processing cost.

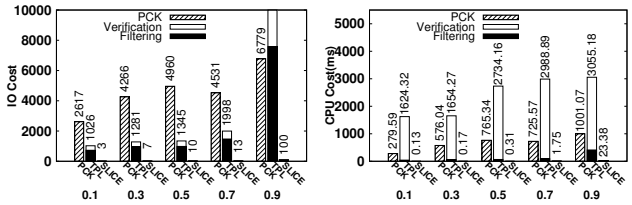
Effect of number of users. Next, we fix the number of facilities to $100,000$ and change the number of users to see the effect of change in the relative size of the two data sets. The data set shown as $x\%$ correspond to the case when the number of users is $x\%$ of the number of facilities, i.e., the number of users is $\frac{100000 \times x}{100}$.

Fig. 29 and Fig. 30 show the results for mixed queries and high quality queries, respectively. As expected, both I/O and CPU cost of all algorithms increase as the number of users increases. This is because the number of candidate users increases resulting in increased verification cost. In terms of I/O cost, PCK scales worse as compared to the other two algorithms. This is because PCK needs to access facility R*-tree every time a user is verified. SLICE consistently



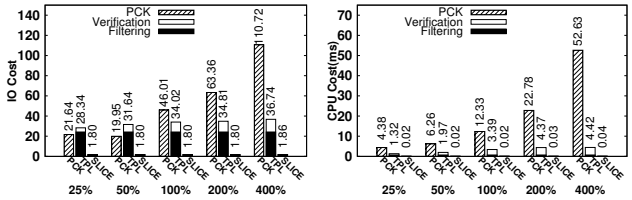
(a) I/O Cost (b) CPU Cost

Fig. 27 Effect of increasing $w[d+1]$ (mixed queries)



(a) I/O Cost (b) CPU Cost

Fig. 28 Effect of increasing $w[d+1]$ (high quality queries)



(a) I/O Cost (b) CPU Cost

Fig. 29 Effect of number of users (mixed queries)

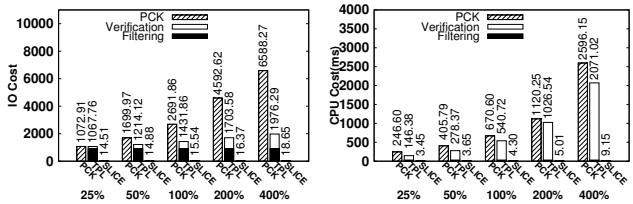
outperforms the other two algorithms and also scales better for both I/O and CPU cost.

5.2.3 Performance evaluation for $d \geq 1$ and $k \geq 1$

In this section, we compare TPL and SLICE for the spatial reverse top-k queries for $k \geq 1$ and on the data sets with one or more static attributes. We vary k from 1 to 15 and the number of static attributes d from 1 to 3. We choose $d \leq 3$ because TPL could not compute the results for $d > 3$ even after a few hours due to its very high CPU cost. The default values are $k = 10$ and $d = 2$. The scoring function is set by assigning equal weight to each attribute, e.g., $w[1] = w[2] = \cdots = w[d+1] = \frac{1}{d+1}$.

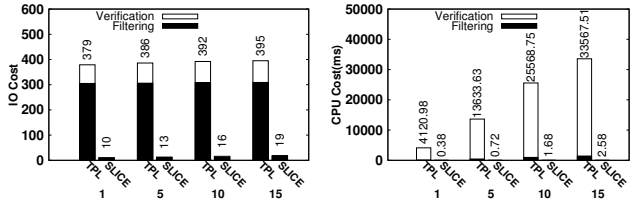
Effect of k. Fig. 31 and Fig. 32 show the effect of k on real data set for mixed queries and high quality queries, respectively. As expected, the I/O and CPU cost of both algorithms increases with the increase in k. However, SLICE significantly outperforms TPL and scales better. As shown in the figures, both I/O cost and CPU cost of two algorithms increase as k increases.

Effect of number of static attributes. Fig. 33 and Fig. 34 study the effect of number of static attributes on real data set for mixed queries and high quality queries, respectively. SLICE significantly outperforms TPL in all settings in terms of both I/O and CPU cost. Fig. 33 shows that the I/O cost



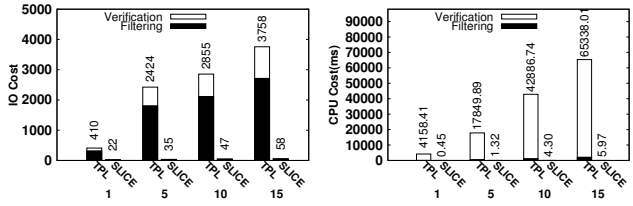
(a) I/O Cost (b) CPU Cost

Fig. 30 Effect of number of users (high quality queries)



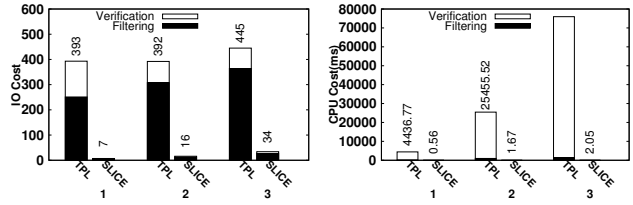
(a) I/O Cost (b) CPU Cost

Fig. 31 Effect of k (mixed queries)



(a) I/O Cost (b) CPU Cost

Fig. 32 Effect of k (high quality queries)

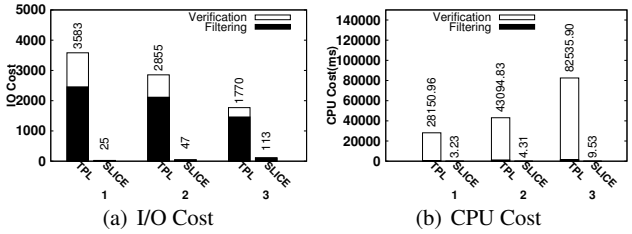


(a) I/O Cost (b) CPU Cost

Fig. 33 Effect of number of static attributes (mixed queries)

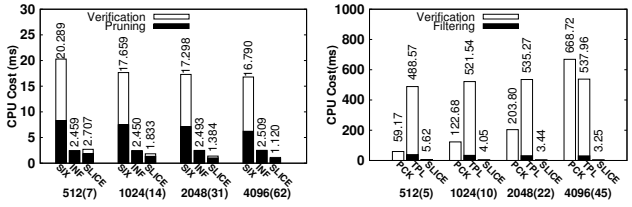
and CPU cost of both algorithms increases as the number of attributes increases. Fig. 34 shows the same trend except that the I/O cost for TPL decreases with the number of static attributes. This anomaly in the trend for I/O cost is possibly due to the different opposing factors that may contribute positively or negatively towards the overall cost as explained below.

The cost of algorithms is affected by several factors. Firstly, as d increases, the weight of dynamic attribute decreases (recall, $w[i] = \frac{1}{d+1}$) which contributes towards reducing the overall cost (as explained earlier for Fig. 28). On the other hand, as the number of attributes increases, the size of R*-tree increases which contributes towards increasing the overall cost. Also, as d increases, the difference between the static scores between the query and facilities decreases, i.e., $|q_s - f_s|$ decreases. This is because as the number of static attributes increases, it becomes less likely



(a) I/O Cost (b) CPU Cost

Fig. 34 Effect of number of static attributes (high quality queries)



(a) RkNN query (b) SRTk query

Fig. 35 Effect of page size

that one facility is better than another facility on *all* static attributes - this behavior is due to the same reason the size of skyline grows for increasing dimensionality. Since the static score gaps decreases, $|\Delta|$ also decreases which contributes towards increasing the overall cost.

5.3 Experiments in main memory

In this section, we evaluate the performance of the algorithms for main memory data sets. Specifically, we vary the page size of R*-tree from 512 bytes to $4,096$ bytes and show the effect on the algorithms in Fig. 35. Each numbers in parenthesis corresponds to the fan-out of the tree for the given page size.

Fig. 35(a) shows the results for RkNN queries (default settings) and Fig. 35(b) shows the results for SRTk queries using default synthetic data set and high quality queries. The impact of page size on the CPU cost is not huge as can be seen from the figures. In fact, the CPU cost of both the region-based algorithms (e.g. Six-regions and SLICE) decrease as the page size increases. This is mainly because the number of facilities m considered by SLICE increases as the page size increases which results in a larger pruned area improving the overall cost. Note that the pruning cost of region based techniques is $O(1)$ once d_k has been computed. Although the pruned area for the half-space based approaches also increases as the page size increases, the overall cost also increases or remains similar mainly because the pruning cost also increases (recall the pruning cost is at least $O(m)$ for half-space based approaches).

6 Conclusion

In this paper, we present efficient algorithms for reverse k nearest neighbors (RkNN) queries and spatial reverse top-k (SRTk) 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 efficient algorithms to compute RkNN queries and SRTk queries. We conduct an extensive experimental study on both real and synthetic data sets. The results show that our proposed algorithm, SLICE, performs significantly better than the existing RkNN algorithms in terms of CPU cost and scales better as k increases. SLICE also outperforms other SRTk algorithms by several orders of magnitude.

Acknowledgments. Muhammad Aamir Cheema is supported by ARC DE130101002 and DP130103405. Xuemin Lin is supported by NSFC61232006, NSFC61021004, ARC DP120 104168 and DP110102937. The research of Ying Zhang is supported by ARC DP130103245 and DP110104880. Wenjie Zhang is supported by ARC DP150103071 and DP150102 728.

References

1. 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*, pages 886–897, 2009.
2. R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest neighbor and reverse nearest neighbor queries for moving objects. In *IDEAS*, pages 44–53, 2002.
3. 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*, pages 339–350, 2011.
4. T. Bernecker, T. Emrich, H.-P. Kriegel, M. Renz, , and S. Z. A. Züfle. Efficient probabilistic reverse nearest neighbor query processing on uncertain data. *PVLDB*, 4(10):669–680, 2011.
5. 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*, pages 189–200, 2010.
6. 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.*, 22(4):550–564, 2010.
7. M. A. Cheema, X. Lin, W. Zhang, and Y. Zhang. Influence zone: Efficiently processing reverse k nearest neighbors queries. In *ICDE*, pages 577–588, 2011.
8. M. A. Cheema, X. Lin, Y. Zhang, W. Wang, and W. Zhang. Lazy updates: An efficient technique to continuously monitoring reverse knn. *PVLDB*, 2(1):1138–1149, 2009.
9. M. A. Cheema, Z. Shen, X. Lin, and W. Zhang. A unified framework for efficiently processing ranking related queries. In *Proc. 17th International Conference on Extending Database Technology (EDBT), Athens, Greece, March 24-28, 2014.*, pages 427–438, 2014.
10. M. A. Cheema, W. Zhang, X. Lin, and Y. Zhang. Efficiently processing snapshot and continuous reverse k nearest neighbors queries. *VLDB J.*, 21(5):703–728, 2012.
11. 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. *The VLDB Journal*, 2012.

12. S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Indexing reverse top-k queries in two dimensions. In *Database Systems for Advanced Applications*, pages 201–208. Springer, 2013.
13. T. Emrich, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Incremental reverse nearest neighbor ranking in vector spaces. In *SSTD*, 2009.
14. O. Gkorgkas, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Discovering influential data objects over time. In *Advances in Spatial and Temporal Databases*, pages 110–127. Springer, 2013.
15. C. Jin, R. Zhang, Q. Kang, Z. Zhang, and A. Zhou. Probabilistic reverse top-k queries. In *Database Systems for Advanced Applications*, pages 406–419. Springer, 2014.
16. J. M. Kang, M. F. Mokbel, S. Shekhar, T. Xia, and D. Zhang. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*, pages 806–815, 2007.
17. F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*, pages 201–212, 2000.
18. H.-P. Kriegel, P. Kröger, M. Renz, A. Züfle, and A. Katzdobler. Incremental reverse nearest neighbor ranking. In *ICDE*, pages 1560–1567, 2009.
19. X. Lian and L. Chen. Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. *VLDB J.*, 18(3):787–808, 2009.
20. K.-I. Lin, M. Nolen, and C. Yang. Applying bulk insertion techniques for dynamic reverse nearest neighbor problems. In *IDEAS*, pages 290–297, 2003.
21. J.-h. Park, C.-W. Chung, and U. Kang. Reverse nearest neighbor search with a non-spatial aspect. *Information Systems*, 54:92–112, 2015.
22. F. P. Preparata and M. I. Shamos. *Computational Geometry An Introduction*. Springer, 1985.
23. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995.*, pages 71–79, 1995.
24. M. Safar, D. Ebrahimi, and D. Taniar. Voronoi-based reverse nearest neighbor query processing on spatial networks. *Multimedia Syst.*, 15(5):295–308, 2009.
25. M. Sharifzadeh and C. Shahabi. Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. *PVLDB*, 3(1):1231–1242, 2010.
26. I. Stanoi, D. Agrawal, and A. E. Abbadi. Reverse nearest neighbor queries for dynamic databases. In *ACM SIGMOD Workshop*, pages 44–53, 2000.
27. I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi. Discovery of influence sets in frequently updated databases. *PVLDB*, pages 99–108, 2001.
28. Y. Tao, D. Papadias, and X. Lian. Reverse knn search in arbitrary dimensionality. *VLDB*, pages 744–755, 2004.
29. Y. Tao, M. L. Yiu, and N. Mamoulis. Reverse nearest neighbor search in metric spaces. *IEEE Trans. Knowl. Data Eng.*, 18(9):1239–1252, 2006.
30. D. Tsirogiannis, S. Harizopoulos, M. A. Shah, J. L. Wiener, and G. Graefe. Query processing techniques for solid state drives. In *SIGMOD*, pages 59–72, 2009.
31. `https://international.ipums.org/international/`.
32. `https://www.census.gov/geo/maps-data/data/tiger-line.html`.
33. `www.cs.fsu.edu/\%7Elifeifei/SpatialDataset.htm`.
34. A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. In *ICDE*, pages 365–376, 2010.
35. A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 365–376. IEEE, 2010.
36. A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Norvag. Monochromatic and bichromatic reverse top-k queries. *Knowledge and Data Engineering, IEEE Transactions on*, 23(8):1215–1229, 2011.
37. A. Vlachou, C. Doulkeridis, and K. Nørvåg. Monitoring reverse top-k queries over mobile devices. In *Proceedings of the 10th ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 17–24. ACM, 2011.
38. A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Branch-and-bound algorithm for reverse top-k queries. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 481–492. ACM, 2013.
39. W. Wu, F. Yang, C. Y. Chan, and K.-L. Tan. Continuous reverse k-nearest-neighbor monitoring. In *MDM*, pages 132–139, 2008.
40. W. Wu, F. Yang, C. Y. Chan, and K.-L. Tan. Finch: Evaluating reverse k-nearest-neighbor queries on location data. *PVLDB*, 1(1):1056–1067, 2008.
41. T. Xia and D. Zhang. Continuous reverse nearest neighbor monitoring. In *ICDE*, pages 77–86, 2006.
42. C. Yang and K.-I. Lin. An index structure for efficient reverse nearest neighbor queries. In *ICDE*, pages 485–492, 2001.
43. S. Yang, M. A. Cheema, X. Lin, and W. Wang. Reverse k nearest neighbors query processing: Experiments and analysis. *PVLDB*, 8(5):605–616, 2015.
44. S. Yang, M. A. Cheema, X. Lin, and Y. Zhang. SLICE: Reviving regions-based pruning for reverse k nearest neighbors queries. In *ICDE*, pages 760–771, 2014.
45. M. L. Yiu and N. Mamoulis. Reverse nearest neighbors search in ad hoc subspaces. *IEEE Trans. Knowl. Data Eng.*, 19(3):412–426, 2007.
46. M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao. Reverse nearest neighbors in large graphs. *IEEE Trans. Knowl. Data Eng.*, pages 540–553, 2006.
47. A. W. Yu, N. Mamoulis, and H. Su. Reverse top-k search using random walk with restart. *Proceedings of the VLDB Endowment*, 7(5):401–412, 2014.

APPENDIX

Lemma 20 *Consider a facility f, a ray L^θ, a critical point p on L^θ and a user u that lies on L^θ and $dist(u,q) > d^\theta$. Then, $dist(u,f) \neq dist(f,p) + dist(p,u)$.*

Proof Note that $dist(u,f) \leq dist(f,p) + dist(u,p)$ due to the triangular inequality. Since p and u lie on L^θ, $dist(u,f) = dist(f,p) + dist(p,u)$ if and only if f also lies on L^θ (i.e., $\theta = 0°$) and p lies between u and f (see Fig. 36). We prove by contradiction that p cannot lie between u and f.

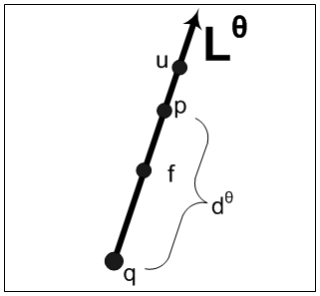


Fig. 36 Lemma 20

Assume that p lies between u and f as shown in Fig. 36. Since $\theta = 0°$, $dist(p,q) = \frac{dist(q,f)^2 - \Delta^2}{2(\Delta + dist(q,f)\cos 0)} = \frac{dist(q,f) - \Delta}{2}$. As $dist(q,f) > |\Delta|$, we have $dist(p,q) < \frac{dist(q,f) + dist(q,f)}{2}$. In other words, $dist(p,q) < dist(q,f)$. Furthermore, since $dist(u,q) > d^\theta$ (i.e., $dist(u,q) > dist(p,q)$), this implies that p cannot be between f and u which contradicts the assumption. □