Efficiently Computing Reverse k Furthest Neighbors

Shenlu Wang*, Muhammad Aamir Cheema†, Xuemin Lin*, Ying Zhang‡ and Dongxi Liu§
*School of Computer Science and Engineering, The University of New South Wales, Australia
Email: {swan398,lxue}@cse.unsw.edu.au
†Faculty of Information Technology, Monash University, Australia. Email: aamir.cheema@monash.edu
‡Quantum Computation and Intelligent Systems, University of Technology, Sydney. Email: ying.zhang@uts.edu.au
§Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia. Email: dongxi.liu@csiro.au

***Abstract*—Given a set of facilities F, a set of users U and a query facility q, a reverse k furthest neighbors (RkFN) query retrieves every user $u \in U$ for which q is one of its k-furthest facilities. RkFN query is the natural complement of reverse k-nearest neighbors (RkNN) query that returns every user u for which q is one of its k-nearest facilities. While RkNN query returns the users that are highly influenced by a query q, RkFN query aims at finding the users that are least influenced by a query q. RkFN query has many applications in location-based services, marketing, facility location, clustering, and recommendation systems etc. While there exist several algorithms that answer RkFN query for $k = 1$, we are the first to propose a solution for arbitrary value of k. Based on several interesting observations, we present an efficient algorithm to process the RkFN queries. We also present a rigorous theoretical analysis to study various important aspects of the problem and our algorithm. An extensive experimental study is conducted using both real and synthetic data sets, demonstrating that our algorithm outperforms the state-of-the-art algorithm even for $k = 1$. The accuracy of our theoretical analysis is also verified by the experiments.**

I. Introduction

Given a set of facilities F, a user u is said to be influenced by a query facility q if q is one of the k nearest facilities of u. This is because users usually prefer nearby facilities and are more likely to be influenced by the advertisements sent from these facilities. Motivated by this, a reverse k nearest neighbor (RkNN) query q returns every user u who is most influenced by q, i.e., q is one of the k closest facilities of u. RkNN query has been extensively studied [1], [2], [3], [4], [5], [6], [7], [8], [9], [10] ever since it was introduced in [11]. Also of interest is to find the users that are least influenced: the reverse k-furthest neighbors (RkFN) query has also received significant attention [12], [13], [14], [15], [16]. Given a set of facilities F, a set of users U and a query facility q, a RkFN query returns every user u for which q is one of its k-furthest facilities.

We illustrate RkFN and RkNN queries using the example in Fig. 1 that shows three facilities (dots) and four users (triangles). Assuming $k = 1$, R1FN of q are the users u_1 and u_2 because q is the furthest facility for u_1 and u_2. Note that although u_3 is the furthest user from q, it is not R1FN of q because q is not its furthest facility. In fact, as q is the closest facility for u_3, u_3 is the R1NN of q, and is highly influenced by q. u_4 is also the R1NN of q.

Similar to RkNN query, RkFN query also has numerous applications in location-based services, marketing, resource allocation, clustering, recommendation systems, profile-based management and graphics rendering etc.

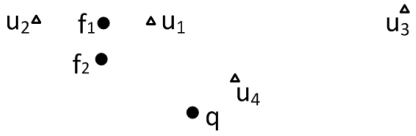


Fig. 1. u_1 and u_2 are RkFN ($k = 1$) of q

Example 1. BotFighters is a pervasive location-based mobile game that is designed to be a MMORPG (massively multiplayer online role-playing game) played in an urban environment. The mission of the game is to locate and shoot other players. Since only nearby players can be shot, it is natural that players are more inclined to look for their nearby players (to shoot them or to avoid being shot by them). In other words, the players usually tend to ignore the furthest players. This can be exploited by a user. Specifically, he may issue a RkFN query to find the players for whom he is among the furthest players. Such players are vulnerable targets. Note that this strategy may also be used in real world battlefields.

Example 2. Painter's algorithm, a popular rendering approach, gives priority to furthest objects from the view point. An object that has many RkFNs (i.e., is among the furthest objects from many view points) is likely to be accessed first for rendering. Identifying such objects and keeping them in cache can significantly improve the rendering speed [17], [18].

Other Examples. RkFN query has been used in reverse facility location problems [19], [20] where the location of an *obnoxious* facility (e.g., a chemical plant) is decided such that the number of its least influenced users (i.e., its RkFNs) is maximized. The computation of maximum spanning tree requires furthest neighbor forest [21], [22] which is a special graph where each node is connected to its reverse furthest neighbors. Recently, a recommender algorithm [23], [24] was proposed that aims at mitigating the popularity bias and increase diversity by using furthest neighborhood that is created by connecting each user to its RkFNs.

Motivated by the above and other applications [16], RkFN query has gained significant attention in recent years. However, all of the existing techniques [12], [13], [14], [15], [16] study RkFN query only for $k = 1$ (called RFN queries hereafter). To the best of our knowledge, we are the first to study this problem for arbitrary value of k. Although the state-of-the-art algorithm for RFN query can be extended for $k > 1$, we show that based on several novel observations, we can devise an efficient algorithm that significantly outperforms the state-of-the-art algorithm for arbitrary value of k, including $k = 1$.

Our algorithm has three phases. In the **first phase**, we access the set of facilities only, and exploit certain problem characteristics to efficiently identify whether a query is *futile* or not. A query is futile if it is guaranteed *not* to have any RkFN regardless of users' locations. The algorithm terminates if the query is futile. Furthermore, the proposed technique to identify futile queries is *tight*, i.e., we prove that a query is futile if and only if it satisfies the proposed criteria.

If the query is not futile, our algorithm enters into the **pruning phase** where search space that cannot contain RkFNs is pruned. This is done by realizing a novel application of k*-depth contour* [25] which has been used earlier in Computational Geometry and Statistics to solve different problems. First, we prove that only *shallow facilities*, namely the facilities that lie on or outside k-depth contour are required to answer the query. The expected number of shallow facilities is $O(k \log |F|)$, where $|F|$ is the total number of facilities. This significantly reduces the I/O and CPU cost of our algorithm. Second, we propose two cheap yet effective pruning strategies. The first pruning strategy utilizes the vertices of k-depth contour in contrast to facilities used by previous techniques. The second pruning strategy adopts a brand new pruning paradigm and calculates *sweeping regions* based on ranges of angles in a polar coordinate system centered at the query.

Finally, in the **verification phase**, the users that lie in unpruned area are identified. These candidate users may or may not be the RkFNs of the query. We introduce the concept of *shadow*, where every user that lies in this area is guaranteed to be a RkFN of the query. The majority of the users are either pruned during the pruning phase or are confirmed by shadow. The remaining candidate users are verified one by one by checking whether q is one of its k-furthest facilities or not. This can be done easily using only the shallow facilities.

Our contributions are summarized below.

- To the best of our knowledge, we are the first to study the problem of RkFN query for arbitrary value of k. Based on several interesting observations and novel pruning techniques, we devise an efficient algorithm to process RkFN queries.
- We present a rigorous theoretical analysis to analyse two different aspects of the problem, namely the expected area of *k-depth contour* and the area pruned by our algorithm. This helps in analysing the expected number of futile queries, number of facilities required for computing a RkFN query, number of unpruned users and I/O cost etc. Also, to the best of our knowledge, we are the first to analyse the expected area of k-depth contour which is of stand-alone interest.
- We conduct extensive experimental study on both real and synthetic data sets, and demonstrate that our algorithm is up to several orders of magnitude faster than the existing algorithms in terms of both CPU and I/O cost. We also conduct experiments to evaluate the accuracy of our theoretical analysis.

The rest of this paper is organized as follows. Section II present an overview of the related work. Our techniques are presented in Section III followed by theoretical analysis in Section IV. An extensive experimental study is provided in Section V. Section VI concludes the paper.

II. Related Work

RFN Query. Kumar *et. al* [12] presented an approximate approach to compute RFNs. It requires pre-calculation of so called FN-ball of points. Tran *et. al* [15] address RFN query on road network based on Network Voronoi Diagram and pre-computation of network distances. Liu *et. al* [13] proposed PIV algorithm that constructs metric indexes and employs triangle inequality to do pruning. Their follow up work [14] proposed PIV$^+$, which identifies a safe area that confirms RFN results without invoking the PIV process. All these techniques require expensive pre-computation and, therefore, are not suitable for dynamic data sets or for arbitrary value of k.

The state-of-the-art algorithm for RFN query in 2D Euclidean space is CHFC (Convex Hull Furthest Cell) proposed by Yao *et. al* [16]. CHFC computes the convex hull (CH) of the set of facilities F. They proved that the furthest neighbor (FN) of a user is a facility that lies on CH. Therefore, q cannot have any RFN if lies inside CH. For example, in Fig. 2, f_9 cannot have any RFN because it lies inside CH. If q lies on CH, CHFC proceeds to calculate *furthest voronoi cell* (FVC) of q, which is a convex polygon in the space such that a point is RFN of q if and only if it is inside FVC.

Extending CHFC for arbitrary value of k. CHFC can be extended to answer RkFN queries for arbitrary value of k. We define k*-th convex hull* as the convex hull of only the facilities that lie within $(k-1)$-th convex hull for $k > 1$, where $k = 1$ corresponds to the usual convex hull. Fig. 2 shows first two convex hulls, i.e., the outer polygon is the 1st convex hull and the inner polygon is the 2nd convex hull. It is easy to prove that a query q cannot have any RkFN if it lies within the k-th convex hull. For example, f_8 cannot have any R2FN because it lies within the 2nd convex hull. To answer a RkFN query, CHFC computes k-th convex hull, and it computes result in the same way of computing FVC if q lies on or outside k-th convex hull.

Note that CHFC can not identify that f_6 and f_7 are futile for $k = 2$ since they do not lie inside the 2nd convex hull. In contrast, we use a much stronger condition that is able to identify f_6 and f_7 as futile. Furthermore, computing k-th convex hull is quite expensive, but the condition used by us can be applied at a much lower cost.

RkNN Query. Most of the existing techniques adopt a pruning and verification framework. The two most notable pruning techniques are *regions-based pruning* [26], [10] and *half-space pruning* [2], [27]. The best known algorithm in terms of I/O cost is *influence zone* [2] and the-state-of-the-art algorithm in terms of overall running time is SLICE [10]. A brief survey and empirical comparison of some of the most popular RkNN algorithms is provided in [28].

k-depth contour k-depth contour (also known as k-hull) is a well studied topic in the field of Computational Geometry and Statistics. Several efficient algorithms [29], [30], [31] have been proposed in the past. However, these algorithms assume that the data can fit in main-memory. Böhm *et. al* [32] proposed I/O optimal algorithms for $k = 1$. Cheema *et. al* [25] proposed an I/O optimal algorithm for arbitrary value of k.

III. TECHNIQUES

Some queries cannot have any RkFN regardless of users' locations. These are **futile** queries. Efficiently identifying futile queries can avoid unnecessary pruning and verification phases, and hence reduces computational cost significantly. We present such techniques in Section III-A. If a query is not futile, we use several pruning strategies to effectively prune the search space that cannot contain any RkFN. These pruning strategies are presented in Section III-B. Users lie in the unpruned space may be RkFNs. They are candidate users. We present techniques to efficiently verify whether a candidate user is a RkFN or not in Section III-C.

A. Determine futile query

Given a query q, a facility f, and a point p, if $dist(p, f) > dist(p, q)$, then p is disqualified from being RFN of q due to f, and we say f **prunes** p. When pruned by at least k facilities, p cannot be RkFN. We divide the space around q in four quadrants Q_1 to Q_4 as shown in Fig. 3.

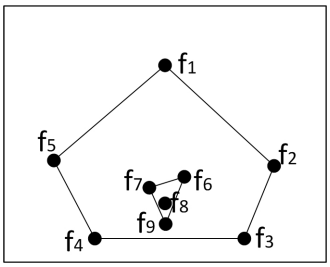


Fig. 2. CHFC

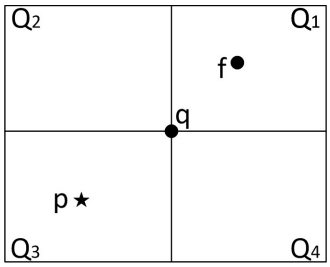


Fig. 3. Quadrant test

Lemma 1: A query q cannot have any RkFN if every quadrant has at least k facilities.

Proof: (Fig. 3) Without loss of generality, we prove that the quadrant Q_3 cannot have any RkFN. Consider a point p in Q_3 and a facility f in Q_1. Let p_x and p_y be x and y coordinates of p. For any facility f in Q_1, we have $|p_x - f_x| > |p_x - q_x|$ and $|p_y - f_y| > |p_y - q_y|$. Hence, $dist(p, f) > dist(p, q)$. If there are at least k facilities in Q_1, q cannot be among the k-furthest facilities of p. ■

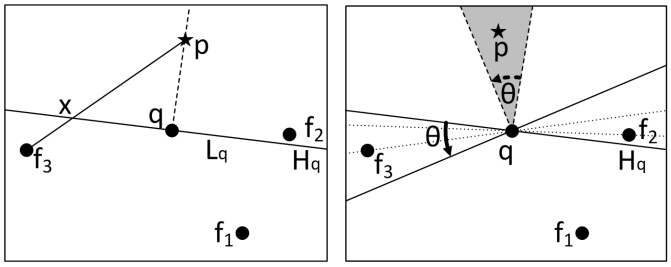
Let L_q be a line passing through q. It divides the space in two halves. Let H_q denote one of the two half-spaces. We define the **P-ray** (perpendicular ray) of H_q to be the ray that is perpendicular to L_q and whose only point in H_q is q (see the dashed ray in Fig. 4(a)).

Lemma 2: Let f be a facility in H_q and p be a point on the P-ray of H_q. f prunes p (i.e., $dist(p, f) > dist(p, q)$).

Proof: (Fig. 4(a)) Let x be the point where the line $\overline{pf}$ intersects L_q. As either $\triangle xqp$ is a right angle triangle or x lies on q, we have $dist(p, x) \geq dist(p, q)$. Hence, $dist(p, f) > dist(p, q)$. ■

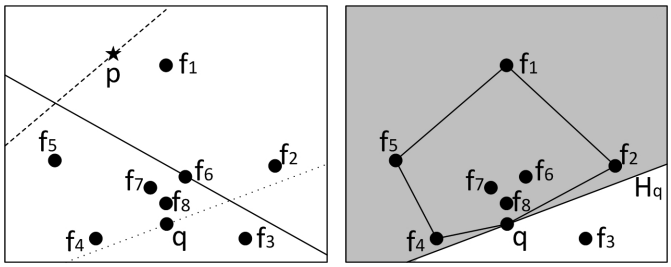
We define the **depth** of a half-space H_q as the number of facilities that lie in H_q and denote it by $|H_q|$. In Fig. 4(a), H_q contains facilities q, f_1 and f_3 and its depth $|H_q| = 3$.

Lemma 3: A point p on the P-ray of H_q cannot be RkFN of q if $|H_q| > k$.



(a) p cannot be R2FN (b) No RkFN in shaded area

Fig. 4. P-ray pruning ($k = 2$)



(a) Location depth (b) q is 2FN of at least one point

Fig. 5. Location depth and its relationship with RkFN

Proof: Since $|H_q| > k$, there are at least k facilities prunes p (Lemma 2). Therefore, p cannot be RkFN of q. ■

Now assume L_q (i.e., H_q) is rotated anti-clockwise, and H_q always contains more than k facilities (at least k other facilities apart from q). During this process, P-ray of H_q also rotates and sweeps an area. According to Lemma 3, no point in this area can be RkFN of q. If the P-ray sweeps the whole space (i.e., rotated 360°), and H_q always contains more than k facilities, we know q is futile. Assume $k = 2$ in Fig. 4(b), H_q is rotated $\theta°$ anti-clockwise, and it always contains at least 3 facilities, namely $\{q, f_1, f_3\}$, $\{q, f_1, f_2, f_3\}$ or $\{q, f_1, f_2\}$. The shaded area is swept by the P-ray and cannot contain any R2FN. Now we adapt the concept of location depth in Computational Geometry to our problem, and define location depth of q as the minimum number of facilities in H_q when H_q is rotated 360°.

Definition 1: **Location depth** of a query point q (denoted as $|q|$) is the minimum depth of any half-space H_q defined using a line L_q passing through q.

Fig. 5(a) shows nine facilities (q and f_1 to f_8) and an arbitrary point p. The location depth of q is $|q| = 2$ because the number of facilities in the half-space defined by the dotted line is 2 (q and f_3) which is minimum. Similarly, location depth of f_6 is $|f_6| = 3$ (see the half-space defined by the solid line that contains f_1, f_2 and f_6). Note that the location depth of a facility is at least one because each half-space contains the facility itself. In contrast, the location depth of an arbitrary point p may be zero. For example, in Fig. 5(a), $|p| = 0$ (see the half-space defined by the dashed line).

Lemma 4: A query facility q cannot have any RkFN if $|q| > k$.

Proof: Let p be an arbitrary point, and H_q be a half-space whose P-ray passes through p. Since $|q| > k$, we have

$|H_q| > k$. According to Lemma 3, p cannot be RkFN of q. ■

Next we show the tightness of Lemma 4 by proving that q cannot be futile if $|q| \leq k$.

Lemma 5: If $|q| \leq k$, there exists at least one point p which is RkFN of q.

Proof: Since $|q| \leq k$, there is at least one half-space H_q that contains at most k facilities. Assume $k = 2$ in Fig. 5(b), H_q (white area) contains only two facilities, q and f_3. It can be proved that there exists a point p for which q is its furthest facility among the facilities lying in the shaded area. Since the number of facilities in H_q is at most k including q, there are at most $(k-1)$ facilities that prune p, i.e., p is RkFN of q.

We construct a convex hull (the polygon) using q and all the facilities that lie outside H_q. It is clear that q is one of the vertices of the convex hull. We construct the furthest Voronoi cell of q using the facilities that lie on or inside this convex hull. Since q is a vertex on the convex hull, the furthest Voronoi cell of q cannot be empty [33]. This implies that there exists at least one point p for which q is the furthest neighbors among the facilities that lie outside H_q. ■

Lemma 4 and 5 state that a query q is futile if and only if $|q| > k$. We remark that although a query q that is not futile has at least one RkFN point, q may or may not have a RkFN user. In other words, the set that contains all RkFN users of a query q may be empty even if q is not futile. Next, we present our pruning strategies to prune the search space that cannot contain any RkFN of the query after we have determined that the query is not futile.

B. Pruning the search space

1) Limitations of adopting half-space pruning:

Half-space pruning [2], [27] is a well known pruning strategy used for processing RkNN queries. Although it is possible to apply this technique for RkFN queries, it suffers from certain limitations.

The perpendicular bisector between a facility f and a query q divides the space into two halves. Each half is called a **bisection**. We use $B_{f:q}$ to denote the bisection that contains f and $B_{q:f}$ to denote the bisection that contains q. For any point $p \in B_{q:f}$, we have $dist(p, f) > dist(p, q)$, i.e., f prunes p and p cannot be RFN of q. In other words, we say that the bisection $B_{q:f_1}$ is pruned. Clearly, the intersection of at least k such bisections is pruned by at least k facilities, and cannot contain RkFN of q. Fig. 6 shows an example. The point p lies in the bisection $B_{q:f_1}$, and f_1 prunes p. Assuming $k = 2$, the shaded area can be pruned because it is the intersection of two bisections, namely $B_{q:f_1}$ and $B_{q:f_2}$.

Based on the above pruning strategy, we can consider every facility $f \in F$ and prune the space that is intersection of at least k bisections. However, this strategy suffers from two serious limitations: 1) the number of facilities considered for pruning may be quite large and considering all bisections is prohibitively expensive; 2) even if the number of facilities (i.e., bisections) used for pruning is not large, the pruning becomes quite expensive especially when k is not small. The reasons are similar to those mentioned in [10], e.g., given n bisections, it is quite expensive to determine the search space that is pruned by at least k bisections.

2) Discarding un-necessary facilities:

To address the first limitation, we discard un-necessary facilities, and keep only the minimum required to correctly compute the results.

Definition 2: A facility f that has $|f| \leq k$ is called a **shallow** facility. In contrast, a facility f' is called a **deep** facility if $|f'| > k$.

Lemma 6: RkFN of a query q can be correctly computed using only the set of shallow facilities.

Proof: Let $S \subseteq F$ be the set that contains all the shallow facilities. We prove that for *any* point p, q is one of p's k-furthest facilities in F (i.e., p is RkFN of q) i) if and ii) only if q is one of p's k-furthest facilities in S.

i) Let q be one of the k-furthest facilities of p in S. If q is not one of p's k-furthest facilities in F, there is at least one deep facility f which is among p's k-furthest facilities in F. Since f is a deep facility, we have $|f| > k$. According to Lemma 4, f cannot have any RkFN, i.e., f cannot be among p's k-furthest facilities. This contradicts the assumption.

ii) Since $S \subseteq F$, it is clear that q is one of the k-furthest facilities of p in F only if it is one of the k furthest facilities of p in S. ■

Lemma 6 implies that we only need to consider the shallow facilities to compute RkFN of a query q. A major challenge in using Lemma 6 is how to identify the shallow facilities efficiently. A straightforward solution is to compute the location depth of each facility. However, this is quite expensive because it requires accessing the whole data set at least once for each facility. We observed that the set of shallow facilities can be efficiently identified with the help of *k-depth contour* [29], [31] (also known as k-hull) which has been used in the past for different problems in Computational Geometry and Statistics.

Definition 3: **k-depth contour** is a convex polygon such that, for every point p, 1) $|p| \geq k$ if p lies on or inside this polygon and 2) $|p| < k$ if p lies strictly outside this polygon.

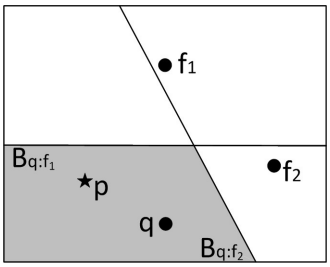


Fig. 6. Half-space pruning

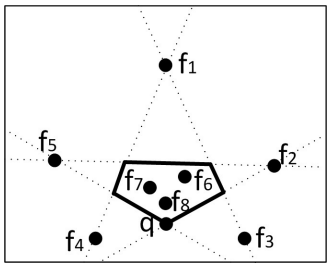


Fig. 7. k-depth contour ($k = 2$)

Fig. 7 shows 9 facilities (q and f_1 to f_8). 2-depth contour is the convex polygon. Note that the vertices of k-depth contour may or may not be the facility objects. The dotted lines define the shape of the 2-depth contour. Each dotted line defines a half-space with depth equal to $k + 1$ such that even a slight rotation reduces the depth to 2. We remark that k-depth contour is the same as convex hull only when $k = 1$.

Lemma 7: A facility f that lies strictly inside the k-depth contour has $|f| > k$.

Proof: We show that every half-space H_f defined by a line L_f passing through f has $|H_f| > k$. Since f lies strictly inside the k-depth contour, the line L_f passes through the k-depth contour. As we show later in Lemma 10, $|H_f| > k$ for every such half-space. Hence, $|f| > k$. ■

According to Lemma 7, to obtain the shallow facilities, we can compute k-depth contour and identify the facilities that lie on or outside k-depth contour. In Fig. 7, the facility q lies on the 2-depth contour and is a shallow facility ($|q| = 2$). The set of shallow facilities consists of q and f_1 to f_5. The facilities f_6, f_7 and f_8 are not shallow facilities and can be discarded when computing RkFN ($k = 2$) of any query.

Lemma 6 and 7 significantly reduce the number of facilities required for computing RkFN queries. It was shown [25] that the number of facilities that lie on or outside k-depth contour is bounded by the cardinality of *k-skyband* [34]. Let $|F|$ be the total number of facilities. Assuming that the two location coordinates are independent of each other, the expected cardinality of k-skyband is $O(k \log |F|)$ [2]. Hence, the expected number of shallow facilities is $O(k \log |F|)$.

3) Implementing the ideas presented so far:

We use the state-of-the-art algorithms to compute k-depth contour [25]. The algorithms utilize the index (e.g., R-tree) constructed on the set of facilities and efficiently return the k-depth contour. Since k-depth contour is always a convex polygon, shallow facilities can be efficiently identified using R-tree (i.e., open only the nodes intersect with the boundary of k-depth contour).

During the computation, we maintain a counter c_i for the number of facilities seen in each quadrant Q_i. If $c_i \geq k$ for each of the four quadrants, we terminate the algorithm as the query is futile (Lemma 1). While the algorithm iteratively computes the k-depth contour, we terminate the algorithm whenever it implies that $|q| > k$ and q is futile (Lemma 4).

Next, we present two cheap yet effective pruning strategies.

4) Pruning using k-depth contour:

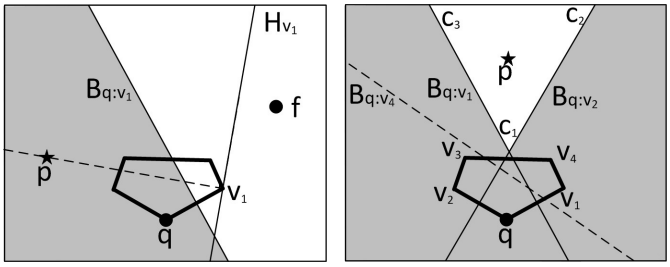
To address the second limitation (Section III-B1), we show that instead of using at least k facilities (i.e., bisections) to prune a point p, we can use the vertices of the k-depth contour, and that a point p is pruned if one vertex prunes it.

Lemma 8: Let v be a vertex[1] of the k-depth contour, no point in $B_{q:v}$ can be RkFN of q.

Proof: Consider the example in Fig. 8(a)[2]. The shaded area illustrates the bisection $B_{q:v_1}$. Let p be a point in $B_{q:v_1}$, we prove that p cannot be RkFN of q. Let H_{v_1} be the half-space that has p on its P-ray. Similar to Lemma 2, every facility f in H_{v_1} has $dist(p, f) > dist(p, v_1)$. Since p lies in the bisection $B_{q:v_1}$, we have $dist(p, v_1) > dist(p, q)$. Hence, for every facility f in H_{v_1} we have $dist(p, f) > dist(p, q)$, i.e., f prunes p. By the definition of k-depth contour, $|v_1| \geq k$. Hence, $|H_{v_1}| \geq k$, there are at least k such facilities that prunes p, and p cannot be RkFN of q. ■

[1]Note that q may lie on a vertex of the k-depth contour as shown in Fig. 8(a). This lemma does not apply to the vertex on which q lies.

[2]Although Fig. 8(a) shows an example where q lies on k-depth contour, Lemma 8 and its proof hold regardless of whether q lies on k-depth contour or not.



(a) Lemma 8 (b) Lemma 9

Fig. 8. Pruning using k-depth contour

We use the bisection $B_{q:v_i}$ for each vertex v_i on the k-depth contour to iteratively prune the search space. The space that cannot be pruned is called **candidate polygon** because a user that lies in this space is a candidate user. In the example of Fig. 8(a), the candidate polygon is initialized to the data space (the square), and is updated to the white area after the vertex v_1 is used for pruning. Then, in Fig. 8(b), the candidate polygon is further updated to the white area after the vertex v_2 is used for pruning.

We observed that certain vertices of k-depth contour are not necessary to be considered for pruning. For example, in Fig. 8(b), the bisection $B_{q:v_4}$ (dashed line) does not prune any point in the current candidate polygon. In other words, v_4 can be discarded after v_1 and v_2 have been used for updating the candidate polygon. The same can be shown for the vertex v_3.

Lemma 9: A vertex v on k-depth contour cannot prune any point p in the candidate polygon if, for *every* corner c_i of the current candidate polygon, $dist(c_i, q) \geq dist(c_i, v)$.

Proof: (Fig. 8(b)) We prove this by contradiction. Assume p can be pruned by $B_{q:v}$, i.e., $dist(p, q) < dist(p, v)$. Since candidate region is a polygon, $B_{q:v}$ prunes a point p inside it only if it prunes at least one corner c_i of the candidate region, i.e., $dist(c_i, q) < dist(c_i, v)$. This contradicts the condition: $dist(c_i, q) \geq dist(c_i, v)$ for every corner c_i. ■

Algorithm 1 Candidate Polygon

Input: q: the query point; an R-tree that indexes the vertices of k-depth contour
Output: CP: candidate polygon

```
1: initialize CP as the boundary of data space
2: insert root of the R-tree in a queue Q
3: while Q is not empty do
4:    dequeue an entry e
5:    for each corner c_i of CP do
6:       if dist(c_i, q) < maxdist(c_i, e) then
7:          mark e as unpruned; break
8:    if e is unpruned then
9:       if e is an intermediate or leaf node then
10:         insert children of e in Q
11:      else
12:         use B_{q:e} to update CP
13: return CP
```

Algorithm 1 presents the details of computing the candidate polygon using the k-depth contour. For efficiency, we index

the vertices of the k-depth contour in a main-memory R-tree[3]. The algorithm iteratively accesses the entries of this R-tree and ignores the entry if each vertex v in an entry e satisfies the condition specified in Lemma 9. Specifically, if $dist(c_i, q) \geq maxdist(c_i, e)$ for each corner c_i of the candidate polygon then all vertex v in the entry e can be discarded. Otherwise, the entry e may contain some vertices that are required to update the current candidate polygon. The rest of the algorithm is self-explanatory.

5) Pruning using shallow facilities:

According to Lemma 3, P-ray of a half-space H_q cannot contain any RkFN if $|H_q| > k$. In other words, a P-ray may contain RkFN only if $|H_q| \leq k$. This property can be used to prune the search space that cannot contain any RkFN.

Lemma 10: Let L_q be a line passes through the k-depth contour and the query facility q. The half-space H_q defined by L_q has $|H_q| > k$.

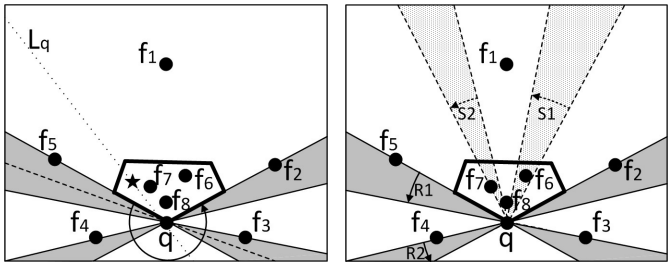
Proof: (Fig. 9(a) illustrates an example of $k = 2$, where 2-depth contour is the polygon shown in thick line.) Let L_q be a line (dotted line) that passes through k-depth contour and defines H_q. Let p be a point (shown as a star) that is inside k-depth contour and lies on L_q. By the definition of k-depth contour, $|p| \geq k$. This implies that $(|H_q| = |H_p|) \geq k$. Now, assume $|H_p| = k$. Since q lies on H_p, an infinitely small rotation can exclude q from H_p resulting in $|H_p| < k$. However, this is not possible because $|p| \geq k$ as per the definition of k-depth contour. Hence, we have $|H_p| > k$, and $|H_q| > k$. ■

According to Lemma 10, we only need to consider the half-spaces H_q that do not pass through the k-depth contour. Hence, we draw the two lines that pass through q and are tangent to the k-depth contour, and we consider only the half-spaces between these two lines. For example, assume $k = 2$ in Fig. 9(a), the two tangent lines are $\overline{qf_5}$ and $\overline{qf_2}$. Let H_q be the half-space that is defined by the line passing through q and f_5 and contains f_4. We rotate H_q anti-clockwise until H_q contains $\overline{qf_2}$. During the rotation, we identify the space for which $|H_q| \leq k$ (shaded areas). For instance, the half-space defined by the broken line in the shaded area contains only two facilities q and f_4.

As we rotate H_q, its P-ray also rotates and sweeps the search space. The area swept by P-ray while $|H_q| \leq k$ holds is called **sweeping region**. In the example of Fig. 9(b), when H_q is rotated in the shaded area marked as $R1$, its P-ray sweeps the dotted area marked as $S1$, and when H_q is rotated in the shaded area marked as $R2$, its P-ray sweeps the dotted area marked as $S2$. $S1$ and $S2$ are the sweeping regions. Any point p that does not lie in $S1$ or $S2$ cannot be RkFN of q. In other words, only the sweeping regions can contain RkFN.

To identify sweeping regions, we first compute the two tangent lines as aforementioned. Then, we pick one of the tangent lines and count the number of facilities in H_q. We maintain the counter for $|H_q|$ as we rotate it: $|H_q|$ is incremented when a facility enters H_q and decremented when a facility leaves H_q. Let x be a point on the horizontal line passing through q and lies on the right of q, and S be the set of shallow facilities, we sort all $f \in S$ by $\angle xqf$, and the next facility encountered during the rotation can be accessed sequentially.

[3]Although the expected number of vertices of k-depth contour is small, it can be much larger, e.g., the convex hull may contain $|F|$ vertices. In this case, the R-tree can be stored in secondary memory



(a) Pruning by rotating H_q (b) Sweeping regions

Fig. 9. Pruning using shallow facilities

Note that the candidate polygon (Section III-B4) and sweeping regions are inherently different. Fig. 10 gives an example of both candidate polygon (shaded area) and sweeping regions (dotted areas). RkFN of a query can be found only in the space that cannot be pruned by *both* pruning strategies (where dotted areas overlap with shaded area).

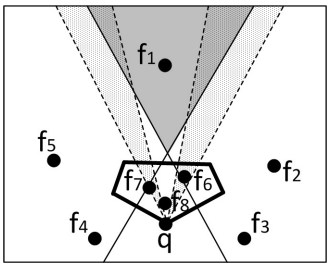


Fig. 10. Space pruned

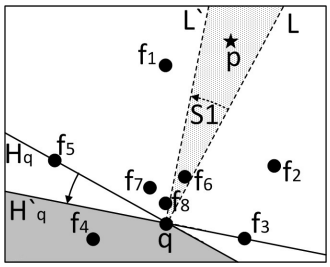


Fig. 11. Pruner region

Next, we present techniques to further prune the area in sweeping regions.

Definition 4: (Fig. 11) Let L and L' be the bounding lines of a sweeping region, and H_q (H'_q) be the half-space such that L (L') is its P-ray. The area contained by both H_q and H'_q (i.e., $H_q \cap H'_q$) is the **pruner region** of this sweeping region. The set of facilities that lie in the pruner region are the **pruners** of this sweeping region.

According to Lemma 2, every point p in a sweeping region is pruned by each of its pruners. In Fig. 11, the pruner region of sweeping region $S1$ is the shaded area, and the pruners of $S1$ is $\{f_4\}$. f_4 prunes every point p in the sweeping region.

Lemma 11: Let P be the set of pruners of a sweeping region. If P contains $k-1$ facilities, a point p in the sweeping region cannot be RkFN if it is also pruned by a facility $f \notin P$.

Proof: The point p is pruned by every facility $f' \in P$ and a facility $f \notin P$. Since P contains at least $k-1$ facilities, p is pruned by at least k facilities and cannot be RkFN of q. ■

Fig 12 shows the same example of Fig. 11, the set of pruners of the sweeping region contains one facility f_4. We draw the perpendicular bisector between f_3 and q. Assuming $k = 2$, according to Lemma 11, no point p in the bisection $B_{q:f_3}$ can be RkFN of q because p is pruned by both f_4 and f_3 (i.e., the dotted white area of the sweeping region can be

pruned). Similarly, the bisection $B_{q:f_5}$ is pruned by f_4 and f_5. Hence, in this example, f_3 and f_5 together can prune the whole sweeping region.

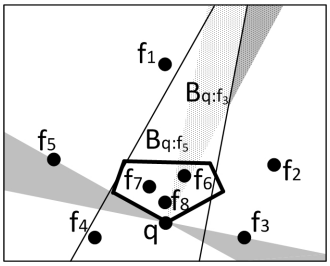


Fig. 12. Pruning sweeping region

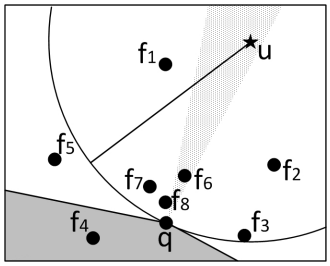


Fig. 13. Verification phase

Note that, whenever the data space (i.e., the square in our example) is pruned, or in other words, unpruned region can only exist outside the data space, our algorithm can be terminated as the query cannot contain any RkFN.

C. Verification

Users lie in the unpruned space can be RkFN and is called candidate users. Each candidate user is verified by checking whether q is one of its k-furthest facilities or not.

According to Lemma 6, we only need the set of shallow facilities S to correctly compute the results. We index shallow facilities in a main-memory R-tree and access only this R-tree to verify the users. This can be done by issuing **boolean k-region query**. Let R be the space outside the circle centered at u with radius $dist(u, q)$ (Fig. 13 shows part of such a circle), a boolean k-region query returns true if and only if there are at least k facilities in R. A user u is RkFN if and only if the query returns false. Assuming $k = 2$, the user u in Fig. 13 is not a RkFN because there are two facilities outside the circle. An optimization of this is to use boolean m-region query ($m < k$). This is an optimization since true is returned sooner, especially when the difference between m and k is large.

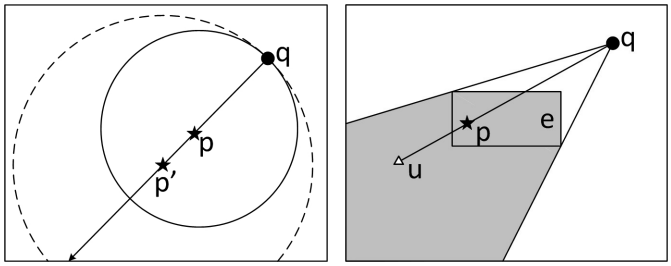
Lemma 12: Let S be the set of shallow facilities, and P be the set of pruners of a sweeping region. If P contains $k - m$ ($m \leq k$) facilities, a point p in the sweeping region cannot be RkFN if it is also pruned by m facilities in $S \backslash P$.

The proof is similar to Lemma 11 and is omitted. During the rotation of H_q, we store for each sweeping region its pruner region and the number of its pruners. To verify an user u, we identify the sweeping region that contains u. Let R be the region that outside both the corresponding pruner region and the circle centered at u with radius $dist(u, q)$, we then issue a boolean m-region query with region R. Assuming $k = 2$ in the example of Fig. 13, the pruner region of u is the shaded area and it contains one facility f_4 (i.e., $m = 1$). In this example, R is the space outside the circle excluding the shaded area. As $k - m = 1$, we issue a boolean 1-region query with region R, and it returns true as f_5 lies in R. Hence, u is not R2FN.

Due to the large number of candidate users, the verification phase could still be very time consuming. Next, we present techniques that help us to identify an area such that candidate users lie in this area are guaranteed to be RkFN and, therefore, do not require verification.

Lemma 13: (Fig. 14(a)) Let p be RkFN of q, $\overrightarrow{qp}$ be the ray whose endpoint is q and passes through p, and p' be a point on $\overrightarrow{qp}$ that has $dist(p', q) > dist(p, q)$. p' is RkFN of q.

Proof: (Fig. 14(a)) Let $Circ(x)$ denote the circle centered at x with radius $dist(x, q)$. We draw two circles $Circ(p)$ and $Circ(p')$. Since both p and p' lie on $\overrightarrow{qp}$ and $dist(p', q) > dist(p, q)$, we know that $Circ(p')$ contains $Circ(p)$. Since p is RkFN of q only if there are less than k facilities outside $Circ(p)$, we know that there are less than k facilities outside $Circ(p')$, which implies p' is RkFN of q. ■



(a) Lemma 13 (b) Lemma 14

Fig. 14. Improving verification using shadows

Definition 5: Let e be a node entry of the R-tree that indexes users. Let p be any point in e. The **shadow** of e consists of every point p' that lies on the ray $\overrightarrow{qp}$ and has $dist(p', q) \geq dist(p, q)$. A node e is a **RkFN node** if every point p in e is RkFN of q.

In Fig. 14(b), the shadow of the entry e is shown shaded. Assuming that q is the light source and e is an item in its way, the shaded area is the shadow of e.

Lemma 14: A user u lies in the shadow of a RkFN node is RkFN.

Proof: Since u lies in the shadow of e, the line $\overline{uq}$ passes through a point p in e. According to Lemma 13, since p is RkFN of q, u is also RkFN of q. ■

Algorithm 2 Verification

```
1: insert root of the R-tree in a queue Q
2: while Q is not empty do
3:    dequeue an entry e from Q
4:    if e is an intermediate or leaf node then
5:       if e is partially inside the unpruned space then
6:          insert children of e in Q
7:       if e is completely inside the unpruned space then
8:          if e is covered by shadows or is RkFN node then
9:             add users in e to result
10:            update the list of shadows using e
11:         else
12:            insert children of e in Q
13:   else                                  /* e is a data object*/
14:      if e lies in one of the shadows then
15:         add e to result
16:      else
17:         if verified by boolean m-region query then
18:            add e to result
19: return result
```

Recall that a facility f prunes a point p if $dist(p, f) > dist(p, q)$. There exists a point in a R-tree node e that is pruned by a facility f if $maxdist(e, f) > mindist(e, q)$. We count the number of such facilities. This represents the maximum number of facilities that can possibly prune a point p in e. If the number is less than k, then e is a RkFN node. Note that e can be a RkFN node only if it lies entirely in the unpruned space. Otherwise, we do not need to do this check. Algorithm 2 presents the details of our verification algorithm.

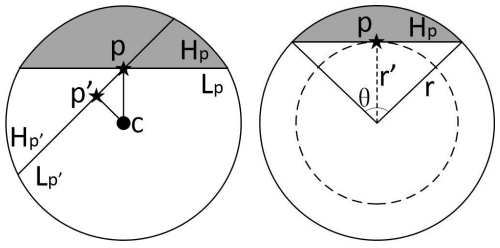
IV. Theoretical Analysis

Assume that the data space is a circle with radius $r = 1$ and that facilities and users are uniformly distributed (Note that this is more challenging for k-depth contour algorithms [25]). Let $|F|$ ($|U|$) be the total number of facilities (users). We analyse the expected area of k-depth contour, and the expected area that cannot be pruned by our algorithm.

A. Expected area of k-depth contour

To the best of our knowledge, we are the first to analyse the expected area of k-depth contour. This is of stand-alone interest and also helps in analysing various important aspects of our algorithm. For instance, the expected number of shallow facilities and the probability of a random query to be futile. Besides, the I/O cost of our pruning phase corresponds to the cost of computing k-depth contour. Since KnightRider [25] does not access any node lies completely inside the k-depth contour, the expected area of k-depth contour also helps in analysing the I/O cost of our pruning phase.

Let c be the center of the circular data space, and p be any point in it. Let L_p be a chord passing through p, and H_p be the corresponding segment that do not contain c. The area of H_p is inversely proportional to the minimum distance from c to L_p, and H_p takes the minimum area when L_p is perpendicular to the line $\overline{cp}$. As we assume uniform distribution, $|p|$ is the number of facilities in H_p when H_p takes the minimum area. In the example of Fig. 15(a), the area of H_p (shaded segment) is smaller than the area of $H_{p'}$ (white segment).



(a) Area of half-space (b) Location-depth

Fig. 15. Area of k-depth contour

Recall that every point p lies on or inside k-depth contour has $|p| \geq k$. Consider the point p in Fig. 15(b), let H_p be the minimum segment that has p on its chord (shaded segment). $|p| = k$ if there are k facilities in H_p. Let θ be the central angle of H_p, r' be the distance from c to p. $r' = r\cos\frac{\theta}{2}$ is the radius of k-depth contour (circle in broken line) because every point p' farther to c than p has $|p'| < k$ and every point p' closer to c than p has $|p'| > k$.

First, we compute θ. In Fig. 15(b), let A_s be the area of H_p, it is equal to the area of the circular sector minus the area of the triangular portion, that is $A_s = \frac{1}{2}r^2\theta - \frac{1}{2}r^2\sin\theta$. Since $r = 1$, we have $A_s = \frac{1}{2}(\theta - \sin\theta)$. We know that the area of data space is $\pi r^2 = \pi$ which contains $|F|$ facilities. Since H_p contains k facilities, we expect $A_s = \frac{k\pi}{|F|}$. Hence, we have $\frac{1}{2}(\theta - \sin\theta) = \frac{k\pi}{|F|}$. We approximate $\theta - \sin\theta$ by Eq. (1) obtained using nonlinear curve fitting[4].

$$\theta - \sin\theta \approx e^{a - b/(\theta + c)} \tag{1}$$

where $a = 3.87031$, $b = 11.09062$ and $c = 0.93407$. Given k, we can derive Eq. (2) from $\frac{1}{2}(\theta - \sin\theta) = \frac{k\pi}{|F|}$.

$$\theta = \frac{b}{a - \log_e \frac{2k\pi}{|F|}} - c \tag{2}$$

Now, the area of k-depth contour denoted by A_{kdc} can be computed using Eq. (3).

$$A_{kdc} = \pi(r')^2 = \pi\cos^2\frac{\theta}{2} \tag{3}$$

B. Expected area of sweeping region

Given a query facility q, we analyse the expected area of sweeping region (Section III-B5). This is used to estimate the expected number of candidate users and the I/O cost of verification phase.

In the example of Fig. 16(a), assume the circle in dashed line is the k-depth contour. The two lines $\overline{qt_1}$ and $\overline{qt_2}$ are tangent to the k-depth contour. t_1 and t_2 are their points of tangency, respectively. The line $\overline{qb_1}$ ($\overline{qb_2}$) is the P-ray of the half-space defined by $\overline{qt_1}$ ($\overline{qt_2}$). The sweeping region is shown shaded. This sweeping region consists of two parts (Fig. 16(b)): i) triangle portion $\triangle qb_1b_2$ and ii) the segment whose chord is $\overline{b_1b_2}$ (thick dashed line). Let A_{tri} be the area of $\triangle qb_1b_2$, A_{seg} be the area of the segment, and A_{SR} be the area of sweeping region, $A_{SR} = A_{tri} + A_{seg}$.

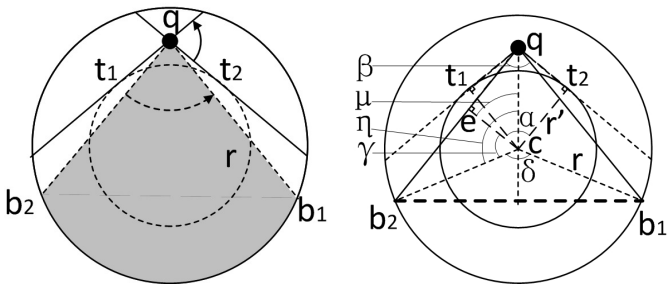
We first compute A_{tri} (Fig. 16(b)). Let $\delta = \angle b_2cb_1$. Let b be the length of the base $\overline{b_1b_2}$. Consider $\triangle b_2cb_1$, we can obtain $b = 2r\sin(\frac{\delta}{2}) = 2\sin(\frac{\delta}{2})$. Let h be the height of $\triangle qb_1b_2$, and d be the distance from c to q. We have $h = d + r\cos(\frac{\delta}{2}) = d + \cos(\frac{\delta}{2})$. Hence we can get $A_{tri} = \frac{1}{2}bh = \sin(\frac{\delta}{2})(d + \cos(\frac{\delta}{2}))$. We will show how to compute δ later.

We then compute A_{seg} (Fig. 16(b)). As this can be easily obtained from $A_{seg} = \frac{1}{2}r^2\delta - \frac{1}{2}r^2\sin\delta = \frac{1}{2}(\delta - \sin\delta)$, the area of sweeping region $A_{SR} = A_{tri} + A_{seg}$ can be calculated by Eq. (4).

$$A_{SR} = \sin(\frac{\delta}{2})(d + \cos(\frac{\delta}{2})) + \frac{1}{2}(\delta - \sin\delta) \tag{4}$$

Now, we compute δ (Fig. 16(b)). Let e be a point on $\overline{qb_2}$ such that the line $\overline{ce}$ is perpendicular to $\overline{qb_2}$. Let $\mu = \angle qce$, $\gamma = \angle ecb_2$ and $\eta = \mu + \gamma$. Since $\triangle qcb_2$ and $\triangle qcb_1$ are

[4]We obtain non-linear curve fitting using Origin downloaded from www.originlab.com



(a) Shaded area is sweeping region (b) Computing sweeping region

Fig. 16. Expected area of the sweeping region

symmetric, we have $2\eta + \delta = 2\pi$. Therefore, we have $\delta = 2\pi - 2\eta = 2\pi - 2\mu - 2\gamma$, and we need to compute μ and γ.

Now, we compute μ (Fig. 16(b)). Let $\alpha = \angle t_1 c t_2$ and $\beta = \angle b_1 q b_2$. Considering the right angle triangle $\triangle qec$, we have $\mu = \frac{\pi}{2} - \frac{\beta}{2}$. Note that $\overline{ct_1}$ is parallel to $\overline{qb_1}$ because they are both perpendicular to $\overline{qt_1}$. Similarly, $\overline{ct_2}$ is parallel to $\overline{qb_2}$. Hence, $\beta = \alpha$. Considering the right angle triangle $\triangle ct_1 q$ or $\triangle ct_2 q$, we have $\cos(\frac{\alpha}{2}) = \frac{r'}{d}$. Hence, we have $\alpha = \beta = 2\arccos(\frac{r'}{d})$, and $\mu = \frac{\pi}{2} - \arccos(\frac{r'}{d})$.

Now, we compute γ (Fig. 16(b)). Let $dist(c, e)$ be the distance from c to e. Considering the right angle triangle $\triangle qec$, we have $dist(c, e) = d \cdot \sin(\frac{\beta}{2})$. Considering the right angle triangle $\triangle ceb_2$, we have $\cos(\gamma) = \frac{dist(c,e)}{r} = dist(c, e) = d \cdot \sin(\frac{\beta}{2})$. Therefore, we have $\gamma = \arccos(d \cdot \sin(\frac{\beta}{2}))$. Since $\beta = 2\arccos(\frac{r'}{d})$, we have $\gamma = \arccos(d\sqrt{1 - (r'/d)^2})$.

Now, we can compute $\delta = 2\pi - 2\mu - 2\gamma$ using Eq. (5).

$$\delta = \pi - 2\arccos(d\sqrt{1 - (\frac{r'}{d})^2}) + 2arccos(\frac{r'}{d}) \quad (5)$$

V. Experiments

A. Experimental settings

We compare our algorithm (labelled as **Our**) with the state-of-the-art algorithm **CHFC** [16] (extended for $k > 1$ as described in Section II). Both algorithms are implemented in C++, compiled by g++ with flag -O3. The experiments are run on a 32-bit PC with Intel Xeon 2.40GHz dual CPU and 4GB memory running Debian Linux. As stated earlier, our algorithm adopts existing algorithms [25] to compute the k-depth contour. Two algorithms were proposed in [25], namely SkyRider and KnightRider. KnightRider is an I/O optimal algorithm, while SkyRider consumes slightly more I/Os but is more efficient in terms of CPU cost. Although any of the two algorithms can be embedded in our framework, in our implementation, we prefer better CPU cost and use SkyRider.

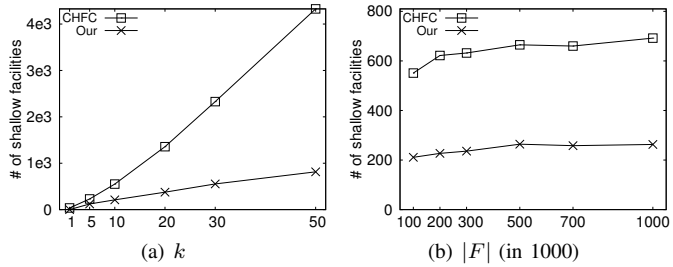
The experimental settings are similar to those used in [16]. Specifically, we use both synthetic and real data sets. The real data set consists of $476,587$ points, and is a combination of 4 data sets[5], namely nodes in California road network (CA), San Francisco road network (SF), road network of North American (NA) and points of interest in CA. Longitude and latitude are normalized to the range [0,100000]. We also removed duplicated points which reduced the data size to $474,655$. The synthetic data set consists of $3,000,000$ points following uniform distribution.

We vary k from 1 to 50 with default value 10. We vary $|F|$ and $|U|$ from $100,000$ to $1,000,000$ on synthetic data set, and from $50,000$ to $200,000$ on real data set (the facilities and users are randomly selected points from the respective data sets). Similar to existing works, we set $|F| = |U|$ in each experiment unless mentioned otherwise.

Each of the set of facilities and users is indexed by a R*-tree. Similar to existing works, the construction cost of these indices is not included in our evaluation. Everything else is calculated on-the-fly (e.g., k-depth contour, the main-memory R*-tree containing shallow facilities) and the relevant cost is included. The page size is set to 4096 bytes. Since the I/O cost is highly system specific [28], instead of reporting total time (i.e., sum of CPU time and I/O time), we report the CPU cost and the number of I/Os separately (which is a better evaluation approach as suggested in [28]). For each experiments, 1000 points are randomly selected from the facilities data set and correspond to the queries (unless mentioned otherwise), and we report the average cost per query.

B. Shallow facilities

Our algorithm requires only the set of shallow facilities to answer RkFN queries, while CHFC requires access to the facilities that lie on or outside k-th convex hull (Section II). We compare the number of shallow facilities with the number of facilities on or outside k-th convex hull. This is important because the CPU cost and I/O cost of each algorithm is significantly affected by the number of such facilities.



(a) k (b) $|F|$ (in 1000)

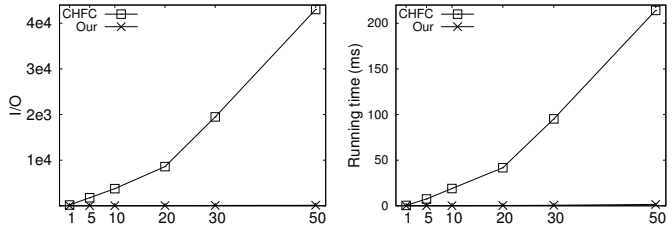
Fig. 17. Number of shallow facilities

Figure 17 compares the number of shallow facilities with the number of facilities on or outside of k-th convex hull. Figure 17(a) shows the effect of k. For $k = 1$, our algorithm must access the same number of facilities as CHFC since 1-depth contour is exactly the same as convex hull. However, for $k > 1$, the number of shallow facilities is much smaller than the number facilities on or outside of k-th convex hull, e.g., for $k = 50$, the number of facilities CHFC must access is approximately 5 times more than that of our algorithm. Figure 17(b) shows the effect of $|F|$. In this case, the number of facilities CHFC must access is approximately 3 times more than our algorithm for various $|F|$. Also, note that our algorithm scales better for increasing k and $|F|$.

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

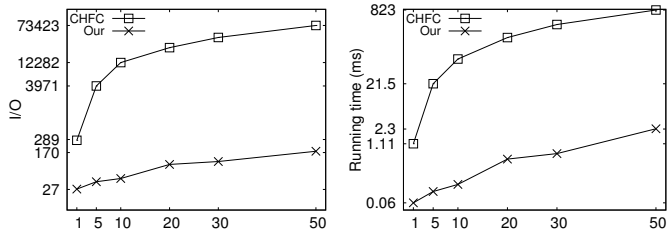
C. Performance evaluation

1) Effect of k: Figure 18 and 19 study the effect of k on real and synthetic data sets, respectively. Specifically, Figure 18(a) and 19(a) show the I/O cost and Figure 18(b) and 19(b) show the CPU time. Although the results on both synthetic and real data sets demonstrate similar trend, we use linear scale in Figure 18 to clearly show how each algorithm scales with k, and log-scale in Figure 19 to better compare the performance for each value of k. Clearly on both data sets, our algorithm is several orders of magnitude better than CHFC in terms of both I/O and CPU cost and scales better.



(a) k (b) k

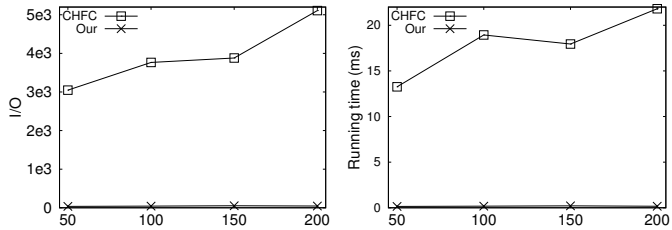
Fig. 18. Real data set



(a) k (b) k

Fig. 19. Synthetic data set

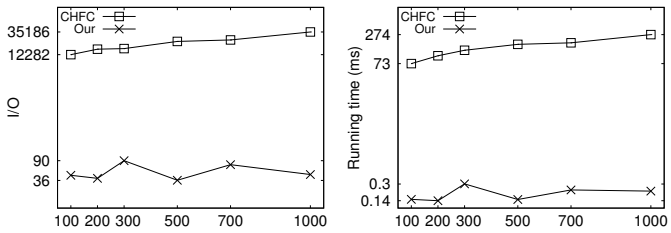
Although for $k = 1$ k-th convex hull and k-depth contour are the same, our algorithm is significantly better than CHFC on both data sets. This is mainly because our algorithm can terminate in case of futile query even before the computation of k-depth contour has been completed in contrast to CHFC that can terminate only after the convex hull has been computed.



(a) $|F|$ and $|U|$ (in 1000) (b) $|F|$ and $|U|$ (in 1000)

Fig. 20. Real data set

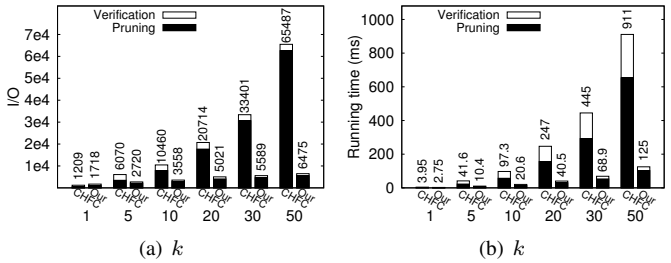
2) Effect of data size: Figure 20 and 21 study the effect of data size. Figure 20(a) and 21(a) show the I/O cost whereas Figure 20(b) and 21(b) show the CPU cost. For the similar reasons mentioned earlier, linear scale is used in Figure 20 and log scale is used in Figure 21. Our algorithm is several orders of magnitude better than CHFC in terms of both CPU and I/O cost. Furthermore, the cost of our algorithm is not affected by the data set size in contrast to the cost of CHFC that increases significantly. This is mainly because the verification cost of CHFC is linear to $|U|$, whereas our algorithm is not significantly affected by $|U|$ due to more sophisticated filtering and verification strategies adopted.



(a) $|F|$ and $|U|$ (in 1000) (b) $|F|$ and $|U|$ (in 1000)

Fig. 21. Synthetic data set

3) Non-futile queries: In the previous section, we reported the results where each query corresponds to a randomly selected facility from the data set. Since the number of shallow facilities is much smaller as compared to the total number of facilities, a majority of the queries are futile queries (a query that is not a shallow facility is a futile query). Is it possible that our algorithm is better than CHFC for futile queries but worse for non-futile queries? In this section, we address this question and select only the queries that are not futile. Specifically, we randomly generate 1000 points outside k-depth contour and each point corresponds to one query (which is guaranteed not to be futile).



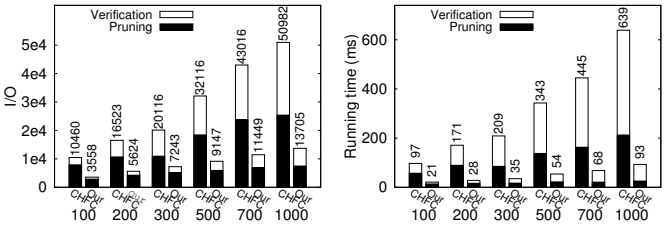
(a) k (b) k

Fig. 22. Non-futile queries (effect of k)

Figure 22 shows the effect of k. Note that the I/O cost of our algorithm is larger than CHFC for $k = 1$. This is because CHFC uses an I/O optimal algorithm to compute convex hull, whereas in our implementation, we use SkyRider which is not I/O optimal. Our algorithm can easily adopt an I/O optimal algorithm (e.g., KnightRider) to reduce the I/O cost. For $k > 1$, our algorithm is significantly better than CHFC for non-futile queries (e.g., for $k = 50$, the number of I/Os for CHFC is more than ten times of that of our algorithm). In terms of CPU cost, our algorithm is faster than CHFC for all values of k including $k = 1$. Figure 22 also demonstrates that pruning cost is the dominant cost for both algorithms.

Figure 23 shows the effect of data set. Clearly, our algorithm performs significantly better in terms of both I/O and CPU cost, e.g., for the largest data set, CHFC consumes around 5 times more I/O and is around 7 times slower.

4) Effectiveness of proposed techniques: Since our algorithm requires pruning and verification only for the non-futile

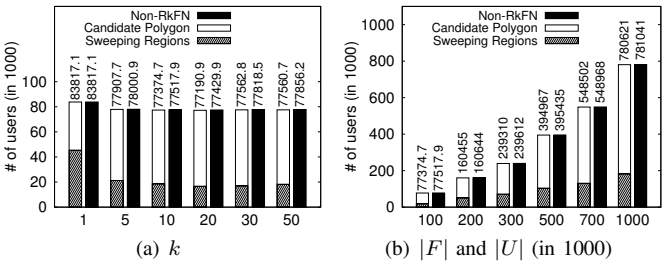


(a) $|F|$ and $|U|$ (in 1000) (b) $|F|$ and $|U|$ (in 1000)

Fig. 23. Non-futile queries (effect of data set)

queries, we randomly generate 1000 queries that lie outside the convex hull, and report the average cost per these non-futile queries.

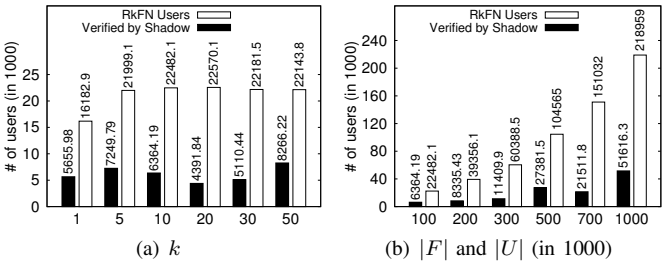
Effectiveness of pruning strategies. Figure 24 evaluates the efficiency of our pruning techniques by showing the number of users pruned by the candidate polygon (Section III-B4) and the number of users pruned by the sweeping regions (Section III-B4). To show the effectiveness of the pruning strategies, we also show the total number of users that are not RkFNs. Since only the users that are not RkFN can be pruned by a pruning strategy, this corresponds to the maximum possible number of users that any pruning technique can prune.



(a) k (b) $|F|$ and $|U|$ (in 1000)

Fig. 24. Effectiveness of pruning techniques

Figure 24(a) evaluates the effectiveness of pruning techniques for varying values of k with $|U|$ and $|F|$ set to $100,000$. Figure 24(b) evaluates the effect of varying data set size with k set to 10. Note that our pruning techniques can prune almost all users that are not RkFNs, more than 99.5% of which are pruned by our pruning techniques. Furthermore, both strategies are quite effective and prune a significant number of users.



(a) k (b) $|F|$ and $|U|$ (in 1000)

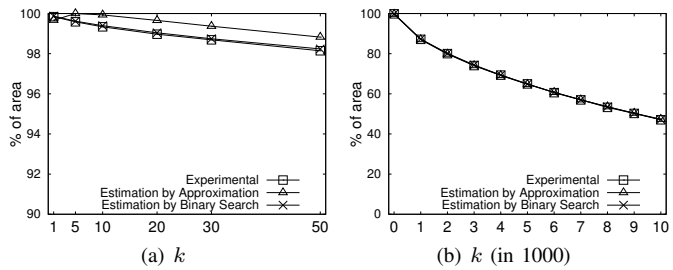
Fig. 25. Effectiveness of shadows

Effectiveness of shadows. Recall that a candidate user u can be confirmed as RkFN and does not require verification if it lies in a shadow. We evaluate the effectiveness of the shadows by showing the number of users that are not required to be verified. Figure 25 shows that around $25-35\%$ of the users lie in the shadows and are not required to be verified. This results in significant saving because the boolean k-region queries are not required for such users.

D. Theoretical analysis evaluation

This section evaluates the accuracy of our theoretical analysis. In each experiment, we generate $100,000$ facilities and $100,000$ users following uniform distribution in a circle of unit radius.

1) Area of k-depth contour: Recall that, in Section IV-A, we obtain the value of θ using Equation (1) that approximates $\theta - \sin(\theta) = \frac{2k\pi}{|F|}$ using non-linear curve fitting. We remark that the value of θ can also be approximated to desired accuracy from $\theta - \sin(\theta) = \frac{2k\pi}{|F|}$ by using a binary search, e.g., continue a binary search on θ until $\theta - \sin(\theta)$ is almost equal to $\frac{2k\pi}{|F|}$. In our experiments, we use both of these methods to approximate θ, and in turn compute the expected area of k-depth contour. The result of the two methods are labeled *Estimation by Approximation* and *Estimation by Binary Search*, respectively. The so-called actual area of the k-depth contour (shown as *Experimental*) is estimated by generating 1 Million points uniformly at random in the data universe and finding how many of these points lie inside the k-depth contour. The area is shown relative to the total area of the data space (e.g., $x\%$ means the area of k-depth contour is $x\%$ of the total space).



(a) k (b) k (in 1000)

Fig. 26. Area of k-depth contour

Figure 26(a) shows the result for k from 1 to 50. It can be observed that the estimation by binary search is almost identical to the experimental result. However, although estimation by approximation follows the trend, it is not as accurate as the estimation by binary search. This is because the nonlinear curve fitting is optimized for the value of θ for the range from 0 to π which is useful when k varies a lot. To confirm this, in Figure 26(b), we vary k from 1 to $10,000$ and evaluate the theoretical analysis. As it can be seen, the estimation is quite close to the experimental value.

We remark that the analysis of k-depth contour is not only useful for RkFN queries but also of stand-alone interest due to its applications in computational geometry and statistics [25].

2) Area of sweeping regions: We randomly generate $1,000$ non-futile queries and report the average area of sweeping regions for different values of k (Figure 27). The experimental results are quite close to the theoretical results. Recall that the area outside the sweeping regions can be pruned. Figure 27 shows that the area of sweeping region is quite small (less than 12% of the total space for $k = 50$), hence, a large area can be pruned using the sweeping regions strategy.

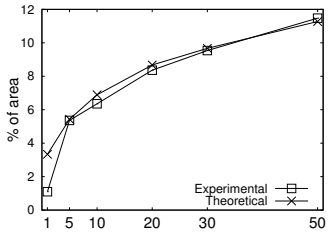


Fig. 27. Area of sweeping region (varying k)

VI. Conclusions

A reverse k-furthest neighbors (RkFN) query returns the users that are least influenced by the query facility. RkFN query has numerous applications in location-based services, marketing, outlier detection, clustering, and profile-based management etc. To the best of our knowledge, we are the first to study RkFN query for arbitrary value of k. Based on several interesting observations and optimisations, we present an efficient algorithm to process RkNN queries. For evaluation purpose, we extend the state-of-the-art algorithm for $k = 1$ to solve RkFN queries for arbitrary value of k. Our extensive experimental study on real and synthetic data sets demonstrates that our algorithm outperforms the state-of-the-art algorithm even for $k = 1$. We also provide a rigorous theoretical analysis that is verified by the experimental study.

Acknowledgment

Muhammad Aamir Cheema is supported by ARC DE130101002 and DP130103405. Xuemin Lin is supported by NSFC61232006, NSFC61021004, NSFC61321064, ARC DP120104168, ARC DP140103578 and ARC DP150102728. Ying Zhang is supported by ARC DE140100679.

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*, 2009, pp. 886–897.

[2] M. A. Cheema, X. Lin, W. Zhang, and Y. Zhang, "Influence zone: Efficiently processing reverse k nearest neighbors queries," in *ICDE*, 2011, pp. 577–588.

[3] J. M. Kang, M. F. Mokbel, S. Shekhar, T. Xia, and D. Zhang, "Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors," in *ICDE*, 2007.

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

[5] M. Sharifzadeh and C. Shahabi, "Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries," *PVLDB*, vol. 3, no. 1, pp. 1231–1242, 2010.

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

[7] Y. Tao, D. Papadias, X. Lian, and X. Xiao, "Multidimensional reverse *k* nn search," *VLDB J.*, vol. 16, no. 3, pp. 293–316, 2007.

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

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

[10] S. Yang, M. A. Cheema, X. Lin, and Y. Zhang, "Slice: Reviving regions-based pruning for reverse k nearest neighbors queries," in *ICDE*, 2014, pp. 760–771.

[11] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *SIGMOD Conference*, 2000, pp. 201–212.

[12] Y. Kumar, R. Janardan, and P. Gupta, "Efficient algorithms for reverse proximity query problems," in *GIS*, 2008, p. 39.

[13] J. Liu, H. Chen, K. Furuse, and H. Kitagawa, "An efficient algorithm for reverse furthest neighbors query with metric index," in *DEXA (2)*, 2010, pp. 437–451.

[14] J. Liu, H. Chen, K. Furuse, and H. Kitagawa, "An efficient algorithm for arbitrary reverse furthest neighbor queries," in *Web Technologies and Applications - 14th Asia-Pacific Web Conference, APWeb 2012, Kunming, China, April 11-13, 2012. Proceedings*, 2012, pp. 60–72.

[15] Q. T. Tran, D. Taniar, and M. Safar, "Reverse k nearest neighbor and reverse farthest neighbor search on spatial networks," *T. Large-Scale Data- and Knowledge-Centered Systems*, vol. 1, pp. 353–372, 2009.

[16] B. Yao, F. Li, and P. Kumar, "Reverse furthest neighbors in spatial databases," in *ICDE*, 2009, pp. 664–675.

[17] J. Gibson, "Compact gps tracker and customized mapping system," Mar. 6 2001, uS Patent 6,198,431. [Online]. Available: https://www.google.com.au/patents/US6198431

[18] G. Frieder, D. Gordon, and R. A. Reynolds, "Back-to-front display of voxel-based objects," *IEEE COMP. GRAPHICS APPLIC.*, vol. 5, no. 1, pp. 52–60, 1985.

[19] B. B. Bhattacharya and S. C. Nandy, "New variations of the reverse facility location problem," in *Proceedings of the 22nd Annual Canadian Conference on Computational Geometry, Winnipeg, Manitoba, Canada, August 9-11, 2010*, 2010, pp. 241–244. [Online]. Available: http://cccg.ca/proceedings/2010/paper64.pdf

[20] S. Cabello, J. M. Díaz-Báñez, S. Langerman, C. Seara, and I. Ventura, "Facility location problems in the plane based on reverse nearest neighbor queries," *European Journal of Operational Research*, vol. 202, no. 1, pp. 99–106, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.ejor.2009.04.021

[21] P. K. Agarwal, J. Matousek, and S. Suri, "Farthest neighbors, maximum spanning trees and related problems in higher dimensions," *Comput. Geom.*, vol. 1, pp. 189–201, 1991. [Online]. Available: http://dx.doi.org/10.1016/0925-7721(92)90001-9

[22] C. L. Monma, M. Paterson, S. Suri, and F. F. Yao, "Computing euclidean maximum spanning trees," *Algorithmica*, vol. 5, no. 3, pp. 407–419, 1990. [Online]. Available: http://dx.doi.org/10.1007/BF01840396

[23] A. Said, B. Fields, B. J. Jain, and S. Albayrak, "User-centric evaluation of a k-furthest neighbor collaborative filtering recommender algorithm," in *Computer Supported Cooperative Work, CSCW 2013, San Antonio, TX, USA, February 23-27, 2013*, 2013, pp. 1399–1408.

[24] A. Said, B. J. Jain, B. Kille, and S. Albayrak, "Increasing Diversity Through Furthest Neighbor-Based Recommendation," in *Proceedings of the WSDM'12 Workshop on Diversity in Document Retrieval (DDR'12)*, 2012.

[25] M. A. Cheema, Z. Shen, X. Lin, and W. Zhang, "A unified framework for efficiently processing ranking related queries," in *EDBT*, 2014.

[26] I. Stanoi, D. Agrawal, and A. El Abbadi, "Reverse nearest neighbor queries for dynamic databases," in *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000, pp. 44–53.

[27] Y. Tao, D. Papadias, and X. Lian, "Reverse knn search in arbitrary dimensionality," in *VLDB*, 2004, pp. 744–755.

[28] S. Yang, M. A. Cheema, X. Lin, and W. Wang, "Reverse k nearest neighbors query processing: Experiments and analysis," *PVLDB*, 2015.

[29] R. Cole, M. Sharir, and C.-K. Yap, "On k-hulls and related problems," in *STOC*, 1984, pp. 154–166.

[30] S. Krishnan, N. H. Mustafa, and S. Venkatasubramanian, "Hardware-assisted computation of depth contours," in *SODA*, 2002.

[31] K. Miller, S. Ramaswami, P. Rousseeuw, J. A. Sellarès, D. L. Souvaine, I. Streinu, and A. Struyf, "Fast implementation of depth contours using topological sweep," in *SODA*, 2001, pp. 690–699.

[32] C. Böhm and H.-P. Kriegel, "Determining the convex hull in large multidimensional databases," in *DaWaK*, 2001, pp. 294–306.

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

[34] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.