Data Sci. Eng. (2016) 1:242–251
DOI 10.1007/s41019-016-0027-8

Pre-computed Region Guardian Sets Based Reverse kNN Queries

Wei Song[1] · **Jianbin Qin**[1] · **Muhammad Aamir Cheema**[2] · **Wei Wang**[1]

Received: 30 June 2016 / Accepted: 16 December 2016 / Published online: 26 January 2017

Abstract Given a set of objects and a query q, a point p is q's Reverse k Nearest Neighbour (RkNN) if q is one of p's k-closest objects. RkNN queries have received significant research attention in the past few years. However, we realize that the state-of-the-art algorithm, SLICE, accesses many objects that do not contribute to its RkNN results when running the filtering phase, which deteriorates the query performance. In this paper, we propose a novel RkNN algorithm with pre-computation by partitioning the data space into disjoint rectangular regions and constructing the *guardian set* for each region R. We guarantee that, for each q that lies in R, its RkNN results are only affected by the objects in R's guardian set. The advantage of this approach is that the results of a query $q \in R$ can be computed by using SLICE on only the objects in its guardian set instead of using the whole dataset. Besides, we raise two new useful variants of RkNN and propose algorithms. Our comprehensive experimental study on synthetic and real the proposed approaches are the most efficient algorithms for RkNN and its variants.

Keywords RkNN · Variants · Pre-computation · Guardian Set · SLICE

✉ Wei Song
wsong@cse.unsw.edu.au

[1] School of Computer Science and Engineering, University of New South Wales, Sydney, NSW, Australia

[2] Clayton School of Information Technology, Faculty of Information Technology, Monash University, Clayton, Melbourne, VIC, Australia

1 Introduction

RkNN: Given a facility set F, a user set U and a query $q \in F$, the RkNN returns users $u \in U$ for which, q is one of their k-closest facilities.

As shown in [1], SLICE is the state-of-the-art RkNN algorithm, which consists of filtering phase and verification phase. SLICE's filtering phase dominates the total query processing cost [2]. We observe that SLICE needs to access many unnecessary facilities and this adversely affects the query performance.

Motivated by above observation and by reviewing [3, 4, 5–11], in this paper, we propose a solution based on pre-computation that divides the whole data space into a set of disjoint rectangular regions. Given a value k, for each rectangular region R, we compute a set of objects $F_g \subseteq F$ s.t. the results of every RkNN query q that lies in R can be computed using only the facilities in F_g. We name F_g guardian set of R and $f \in F_g$ guardian facility of R. When processing query, we determine the region R containing q and then use SLICE on its guardian set to compute the results. Since guardian set size is way smaller than the whole dataset, this approach improves the performance significantly.

We remark that although there exists other pre-computation-based approaches, our approach is unique as its pre-computation does not depend on users set. For example, the technique proposed in [12] pre-computes, for each user u_i, its k-th closest facility f_k and creates a circle C_i centred at u_i with radius $dist(u_i, f_k)$. All such circles are indexed by an R-tree, and a RkNN query q is answered using the circles containing q. A disadvantage of it is that any change w.r.t. user set U requires updating or reconstructing the index. On the other hand, our guardian sets do not depend on U and do not require update with the change in U. This is a desirable property especially because in many real-world applications the updates in the locations of facilities (e.g. restaurants) are less common as compared to the locations of users (e.g. people).

Next, we summarize our contributions.

- To the best of our knowledge, we are the first to propose a pre-computation-based approach that does not depend on the set of users. Our pre-computation significantly reduces the number of facilities to be accessed for SLICE and improves the query processing cost.
- We come up with two R*k*NN variants which are useful in practice and we proposed effective algorithms.
- Our comprehensive experimental study on real and synthetic datasets demonstrates that our algorithm for R*k*NN significantly improves SLICE in query processing and both the algorithm for R*k*NN and variants algorithms outperform existing ones.

The rest of the paper is organized as follows. We introduce related works in Sect. 2. Section 3 presents our techniques constructing rectangular region-based guardian set F_g. We also present our method partitioning universe to many small regions. Section 4 briefly recalls SLICE, and we propose two R*k*NN variants in Sect. 5. In Sect. 6, we do experimental study over three proposed algorithms, respectively. Then we conclude this paper in Sect. 7.

2 Related Works

Six Regions [13] is a region-based technique proposed for R*k*NN. It consists of two phases, namely *Filtering phase* and *Verification phase*. In filtering phase, six regions centre at query q partitioning universe into six regions, each of which has a subtending angle of 60°. In each region, it computes q's k-th nearest neighbour NN_k and construct an arc by centring at q with a radius of $dist(q, NN_k)$. In verification phase, each u locates above the arc in its region as shown u in P_2 (Fig. 1a) cannot be a result and only users lie under the arc of its region can be returned as candidates to be verified.

Influence Zone [14] is a half-space based technique proposed for R*k*NN, denotes *InfZone*. It keeps constructing perpendicular bisector between each facility f_i and q and halving the universe to two parts. The half where f_i locates is pruned by f_i as any u in this area must have closer distance to f_i than to q. For areas pruned by at least k facilities (shaded area in Fig. 1b) cannot return any result. *InfZone* guarantees every u in the unpruned area is a result. Such unpruned area is called *Influence Zone*.

SLICE [2] is the most efficient algorithm before our work for R*k*NN, which is integrated in our query processing algorithm in this paper. We introduce SLICE in detail in Sect. 4.

3 Techniques

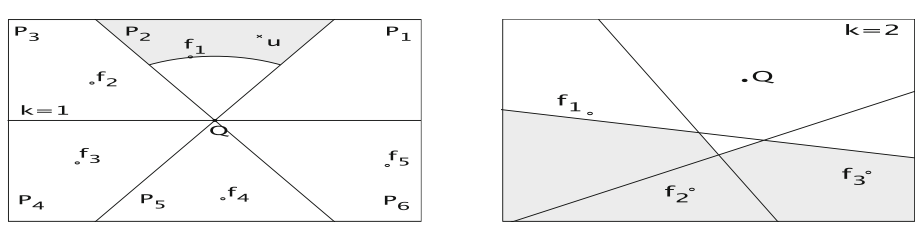
Motivated by [14] that given a query q and a k, we may compute a set of facilities that q's R*k*NN is only affected by them. Similarly, given a rectangular region R and k, we compute a set F_g of facilities that for any $q \in R$, all facilities affecting q's R*k*NN are contained in F_g and the rest facilities $f \notin F_g$ cannot affect q's R*k*NN. As for any $q \in R$, whose R*k*NN results are only guarded by R's F_g. We define F_g *guardian set* and $f_g \in F_g$ *guardian facility* of R.

By partitioning U into many small rectangular regions R s.t for any two $R_i, R_{j(i \neq j)} \in U$, we have $R_i \cap R_j = \emptyset$ and $\bigcup R_{i=1 \ldots n} = U$, we compute and obtain every guardian set F_{gi} of R_i.

3.1 Computing Guardian Set of a Rectangular Region

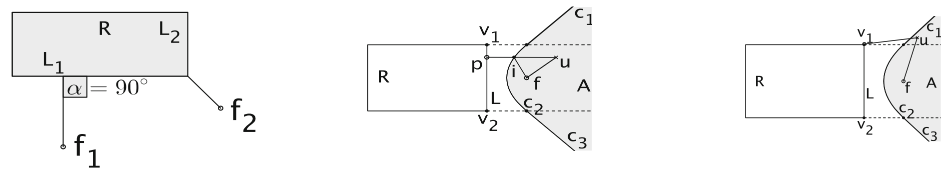
Definition 1 Given a set F of facility f, a rectangular region R and k, the *guardian set* F_g of R consists of a few facilities that for any $q \in R$, q's R*k*NN results are only affected by $f \in F_g$. For any facility $f \notin F_g$, it does not affect q's R*k*NN result. Such $f \in F_g$ is **guardian facility**, denote f_g.

Definition 2 For any f locates outside R, we draw the line segment with minimum distance from f to R joining R at v. We define f is owned by its nearest side L of R and f is in the range of L if v lies on L. If v is a vertex of R, we define f is owned by R's two sides intersecting at v and f is out of range of any R's side. As shown f_1, f_2 in Fig 2a.



(a) Six regions **(b)** Influence zone

Fig. 1 Related works. **a** Six regions, **b** influence zone



(a) Facility ownership (b) Lemma1 proof1 (c) Lemma1 proof2

Fig. 2 Definition 2 & Lemma 1 proof (a) Facility ownership (b) Lemma1 proof1 (c) Lemma1 proof2

Lemma 1 *For any f locates outside R and owned by only one side L of R, we construct a combined curve C consists of:* (i) ***sub-curve1:*** *A partial parabola in the range of L constructed by f as the focus and L as the directrix*; (ii) ***sub-curve2:*** *Two radials that are parts of perpendicular bisectors between f and two L' vertices, respectively, towards directions that are out of L's range. C divides universe into two parts, for any user locates in the same area A with f,* it *has a closer distance to f than to any point in R, we define*: *f* ***prunes*** *A*.

Proof (Lemma 1) We prove Lemma 1 by considering two cases:

Case 1: For any p lies on L, any u locates in the same area A with f has smaller distance to f than to p:

Case 1.1: For any u lies in A within the range of L (as shown in Fig. 2(b)), the min distance $mindist(u, L) = |up|$, where up is the line segment passing c_2 at i and p is the intersection between up and L. Due to the property of parabola, $|pi| = |if|$ and $|up| = |ui| + |if|$. By triangle inequality, $|up| > |uf|$.

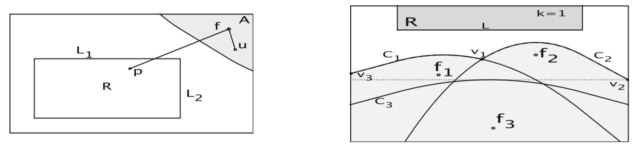
Case 1.2: For u lies in A but is out of L's range (shown in Fig. 2(c)), the minimum distance from u to L is $|uv_1|$, where v_1 is the vertex of L locates at the same side with u w.r.t.f. As u is in the half dominated by f, $|uf| < |uv_1| \leq |up|$.

Case 2: For any p lies in R or on R's sides (exclude L), any u locates in A has smaller distance to f than to p:

Case 2.1: It is easy to show for any u lies in A within the range of L the triangle inequality still holds by changing $|pi| = |if|$ to $|pi| > |if|$, then $|up| > |uf|$.

Case 2.2: For u in A locates out of range of L (in Fig. 2(c)), as $mindist(u, R) = |v_1 u| < |up|$, and $|uf| < |v_1 u|$, as a result, $|up| > |uf|$. □

Lemma 2 *For any f locates outside R and owned by R's two sides L_1, L_2, we construct its combined curve C consists of: (i)* ***Sub-curve1:*** *Two partial parabolas in the range of L_1 and L_2 constructed by f as the focus and L_1,L_2 as directrixes, respectively*; (ii) ***Sub-curve2:*** *Three partial perpendicular bisectors between f and three vertices of L_1, L_2 , respectively, towards directions that are out of range of L_1, L_2. C divides universe into two parts, for any u lies in the same area A with f, u has a closer distance to f than to any point p* in *R and we define*: *f* ***prunes*** *A* (Fig. 3(a)). (*Note that two perpendicular bisector radials at the two ends of C do not necessary both exist due to location between R* and *f.*)



(a) Lemma2 (b) Pruning rule 1

Fig. 3 Lemma 2 and pruning rule 1, **a** Lemma2, **b** Pruning rule 1

With Lemmas 1 and 2, we compute R's guardian set F_g by traversing all $f \in F$ and keep f_g whose pruning area is pruned by other facilities at most $k - 1$ times as a guardian facility.

Apparently, this algorithm is costly that every accessed facility f will be checked by every existed facility, costing a time complexity of $O(|N|^2)$, where |*N*| is the total number of facilities. To avoid such problem, we propose two pruning rules to improve the efficiency.

Pruning Rule 1 *For any $f \in F$ owned by only one side L of R and locates outside R, we find the vertex v_g on the combined curve C_{fgi} of current guardian facility f_{gi} s.t. v_g*

- *is an intersection between other guardian facility's combined curve and C_{fgi} or between C_{fgi} and universe boundary*;
- *lies at the same side with f w.r.t. L*;
- *is pruned exactly k-1 times and has the farthest distance to L or L's extension if v_g is out of L's range; if $dist(L,f) > 2dist(v_g, L)$, f has no influence on constructing R's guardian set and can be pruned.*

Proof As Fig. 3b shows, with f_1 and f_2 existed, the shaded area has been pruned by f_1 and f_2 jointly and v_2 is the vertex described in pruning rule 1. As $dist(L,f_3) > 2dist(v_2, L)$, it guarantees the area pruned by f_3 is pruned by f_1 and f_2 jointly; therefore, f_3 is pruned.

Pruning Rule 2 *For facility f fails meeting pruning rule 1 , we compute its combined curve C_f and keep each intersection on C_f that is*:

- *intersection between C_f and combined curves of other guardian facilities*;
- *intersection between C_f and universe boundaries*;
- *intersection that is joint point between different sub-curves of C_f*.

If all such intersections on the C_f have been pruned by other facilities for at least k times, f has no effect on constructing R's guardian set and can be pruned.

Proof We prove it by contradiction. Assume f and its combined curve C_f meet statement of pruning rule 2 but cannot be pruned, it has at least a part C_i on C_f cannot be pruned and at least existing two intersections that are two vertices of C_i pruned by other facilities for at most k-1 times, which contradicts with the statement of pruning rule 2. Therefore, pruning rule 2 holds. □

Algorithm We compute R's guardian set by indexing all facilities in a R-tree and accessing each node in an ascending order of its minimum distance to R. For each f that is not pruned by pruning rule 1, its combined curve C_f is created. Then we compute intersections between C_f and existed combined curves and apply pruning rule 2 to prune more existed guardian facilities. We update the temporary guardian set by removing facilities that meet pruning rule 2 and adding f if it is not pruned. The algorithm finishes when all nodes are accessed.

3.2 Partition Universe

Before the computation in Sect. 3.1, we partition the universe U to small rectangular regions R. It is not avoidable that each R contains facilities. However, with more facilities contained inside, more guardian facilities locate outside of R are likely to be pruned when q is given. Therefore, we set a threshold T_f as the maximum number of facilities each R contains when partitioning to reduce the number of such guardian facilities that are possibly pruned in query process.

We partition U in a kd-tree [15] liked manner. Specifically, a big region R_u is split into four disjoint child regions R_l if R_u contains more than T_f facilities. For each region partitioned, we first find the median x-coordinate of all facilities in R_u and partition R_u into two smaller intermediate regions by the median. Then we partition two intermediate regions into four smallest ones following the same way by focusing on their y coordinates instead. Such procedure is conducted recursively until every region meets the threshold.

As we do not consider those facilities locate inside R when computing F_g of R in Sect. 3.1 and they are likely to affect q's RkNN. Therefore, we add all of them to R's guardian set after computing guardian facilities of R in case missing facilities that may affect q's RkNN results. Such facilities are also guardian facilities w.r.t.R.

Algorithm 1: ConstructGuardianSet(R, F, k)

```
Input: Rectangular region R, a set F of facilities, value k
Output: R's Guardian set
initialise F_g as ∅;
insert root of R-tree in a min-heap h;   /* we access f in an ascending order of
mindist(R,f) */;
while h is not empty do
    deheap an entry e;
    if e is not a facility then
        Put e's child entry in h;
    else
        if f cannot be pruned by pruning 1 then
            Compute C_f w.r.t. R;
            UpdateExistedGuardianSet(F_g,f,k,C_f);
Return F_g;
```

Algorithm 2: UpdateExistedGuardianSet(F_g, f, k, C_f)

Input: Existing Guardian Set F_g, new facility f, value k, combined curve C_f
Output: Updated Guardian Set F_g

for each $f_g \in F_g$ **do**
  compute and keep intersections between C_f and C_{f_g};
for each $f_g \in F_g$ **do**
  update pruned times of intersections on C_f and C_{fg};
  if f_g *can be pruned by pruning rule 2* **then**
    remove f_g from F_g;
if f *is not pruned by pruning rule 2* **then**
  Put f in F_g;
Return F_g;

4 Query Processing

With guardian sets, given a query q and k, we locate the region R where q locates and retrieve its guardian set w.r.t. k. Then SLICE starts processing query.

Filtering phase Given a query q and k, SLICE partitions the universe U into several regions, each of which has same subtending angle. In [2], the best partition number is 12, but it is 9 in our algorithm according to our preliminary experimental study.

Figure 4 shows an example where the space is divided into nine regions. Consider the perpendicular bisector between f_1 and q in region P, f_1 contributes two arcs in P, namely upper arc U_1 (with radius r_U) and lower arc L_1 (with radius r_L). Normally, every f constructs at least a lower arc to each region that its perpendicular bisector L_{fq} passes and it will contribute an upper arc for each region that L_{fq} passes and the max subtending angle between L_{fq} and such region is smaller than 90°.

In filtering phase, each partitioned region maintains a k-th lower arc and a k-th upper arc dynamically, for each f whose lower arcs are lower than k-th upper arcs in corresponding regions will be kept as a significant facility of such regions to verify users, otherwise it is discarded (like f_2 is discarded in P in Fig. 4). After all facilities are accessed, the filtering phase finishes.

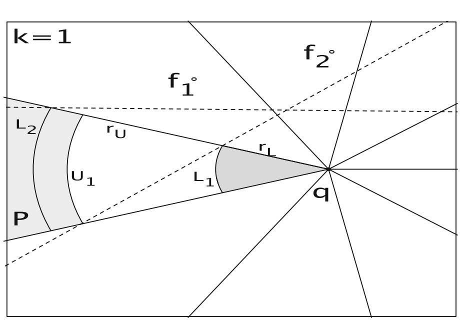


Fig. 4 SLICE filtering

Verification phase All users u are indexed in a R-tree and accessed in the ascending order of their distances to q. For u lies above the k-th upper arc of its region is discarded and every u lies under the k-th lower arc is returned. The remaining users will be checked by significant facilities of this region.

5 Reverse k Nearest Neighbour Variants

5.1 Fixed Value Reverse k Nearest Neighbour Queries

By the definition of RkNN, previous algorithms return results for each of which, q is *at least* kth nearest neighbour. We realize that it is possible to study RkNN by fixing k value to find the exact Reverse k Nearest Neighbours of q and this variant of RkNN is useful in practice. We name it *Fixed Value Reverse k Nearest Neighbour Queries* (*FRkNN*).

Practical Application A supermarket F is expanding its marketing by serving shuttle services. To schedule routes, F may apply *FRkNN* to find users living in a certain distance to F. Then F will be able to organize shuttle routes by results returned through *FRkNN*.

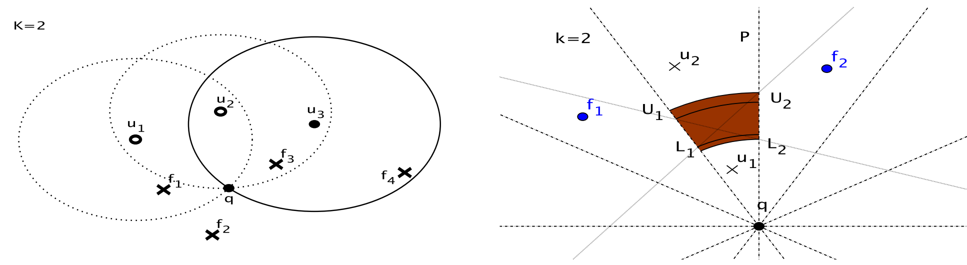
Definition 3 (*Fixed Value RkNN*) Given a query point q, a facility set F, a user set U and a value k, fixed value RkNN returns a subset U_r of user u_i from U s.t. for each $u_i \in U_r$, q is u_i's k-th nearest neighbour, denote *FRkNN*. As shown in Fig. 5a, u_1 and u_2 are Q's *FR2NN*.

Lemma 3 *Base on SLICE* [2], *for each slice P, the candidate users for FRkNN only locate in between the* $(k-1)$*-th lower arc and k-th upper arc. The rest users will be pruned.*

Proof (For Lemma 3) In the slice P,

– for each user u_i locating above k-th upper arc, there are at least k other facilities than q that closer to u_i and q

Fig. 5 FRkNN and Lemma 3. **a** Fixed value RkNN, **b** Lemma 3



(a) Fixed Value RkNN **(b)** Lemma 3

cannot be u_i's k-th nearest neighbour, u_i is pruned. e.g. u_2 in Fig 5b.

- for each user u_i that locates under the (k-1)-th lower arc, there are at most (k-2) other facilities than q that closer to u_i and q is at least $k-1$ closest facility to u_i and cannot be u_i's k-th nearest neighbour, u_i is pruned. e.g. u_1 in Fig 5b.

The Lemma 3 is proved. □

Algorithm By learning from Region Guardian Set Based R*k*NN [16], we propose our algorithm about *FRkNN*. $(k-1)$-th lower arc and k-th upper arc dynamically and maintain all facilities that contribute at least one arc. In verification phase, by Lemma 3, we filter some users and for those users locating in the unpruned area are returned to be verified by facilities collected in filtering phase to answer *FRkNN*.

5.2 Weighted Reverse k Nearest Neighbour Queries

By reviewing [16] and its variant in Sect. 5.1, we come up with an useful instance:

Algorithm 3: FRkNN$(q, k, R_U, IndexFile, n)$

```
Input: Query point q, k, Users' R tree U_rtree, IndexFile, partition size n
Output: q's FRkNN
Initialise the result set R=∅ and each slice P_i's contributed facility set F_ci=∅;
Locate q's region and retrieve its guardian set F_g from IndexFile;
Partition space into n equal slices;
Put all f_g in a minHeap H_f;          /* we access f_g in an ascending order of
dist(q, f_g) */;
while H_f is not empty do
    deheap a facility f_g;
    for each slice P_i that f_g's PB_{Q,f_g} passes do
        if f_g is not pruned then
            Update (k-1)-th lower arc and k-th upper arc;
            Maintain contributed facility set F_ci of slice P_i;
Put root of U_rtree into minheap H_u;
while H_u is not empty do
    deheap entry e;
    if entry e is not pruned by Lemma 3 then
        if e is a facility then
            Verify e by F_ci;
            if e is the result then
                Put it into R;
        else
            Enheap each of its child node;
Return R;
```

Like [16], given a query point q, we locate q's region and retrieve its guardian set from index file. Then universe is partitioned into equal slices in filtering phase. Next, we accesses the guardian facilities in the ascending order of the facilities' distances to q. For each slice P, we update P's

Practical Application In order to attract more customers to do shopping, a supermarket M proposes a promotion by reducing prices on some goods, e.g. putting a 30% discount off the RRP on a certain brand washing machine. Offered the discount, customers are possible to come for shopping even if

M is far from where they live if the total cost aggregated by product and transportation is cheaper than the total cost buying same products from their nearby supermarkets. On the other hand, *M* is able to find users to which, it is at least k-th cheap supermarket in terms of the total consumption(by considering products and transportation costs). Then *M* sends offers to such customers notifying them to come for shopping.

Motivated by above application, we come up with a new cost model for users by aggregating products cost and their transportation cost w.r.t. each supermarket and measured by which, we propose a R*k*NN variant over a facility set F and a user set U. The variant is to find the Reverse k Nearest Neighbour of the given facility under the new cost model. We name this variant *Weighted Reverse k Nearest Neighbour Queries* (*WRkNN*).

Definition 4 In Euclidean space, given a facility set F, a user set U and a query point q that $q \in$ F, the weighted RkNN (WRkNN) returns a subset U_R from U, s.t. for each u_r in U_R, q is one of its k facilities with the smallest aggregated score. The aggregated score is defined below.

$$Cost_{total}(q, u_i) = \alpha \times dist(q, u_i) + Cost_{textual}(q). \quad (1)$$

Where α is a factor to convert the spatial attribute to the spatial cost.

We propose a novel algorithm for *WRkNN* in following.

Lemma 4 *For a query point* $q(l, Cost_{textual})$ *and a facility* $f_i(l, Cost_{textual})$, *where* q_l *and* f_{il} *are their locations and* $q_{Cost_{textual}}, f_{i(Cost_{textual})}$ *are their textual costs, respectively. We create a hyperbola H with q and* f_i *as the left and right focuses, respectively, and the absolute value of* $\frac{q_{Cost_{textual}} - f_{i(Cost_{textual})}}{\alpha}$ *as the length of major axis of H:*

If $q_{Cost_{textual}} \geq f_{i(Cost_{textual})}$, *all users locating on the right side of the left branch of H must have a smaller or equal aggregated cost to* f_i *than to q.*
If $q_{Cost_{textual}} < f_{i(Cost_{textual})}$, *all users locating on the right side of the right branch of H must have a smaller aggregated cost to* f_i *than to* q.

Proof (For Lemma 4)

When $q_{Cost_{textual}} \geq f_{i(Cost_{textual})}$, we consider the left branch H_l of H and the extreme case is that u_i has same cost to q and f_i, we obtain $q_{Cost_{textual}} + \alpha \times dist(q_l, u_i) = f_{i(Cost_{textual})} + \alpha \times dist(f_{il}, u_i)$. By transferring, we have $dist(f_{il}, u_i) - dist(q_l, u_i) = \frac{q_{Cost_{textual}} - f_{i(Cost_{textual})}}{\alpha}$. As shown u_1 in Fig. 6a. When u_i move right to the right hand side of H_l(like u_2), according to the definition of H that $b - h = \frac{q_{Cost_{textual}} - f_{i(Cost_{textual})}}{\alpha}$ and the triangle inequality that $e - g < b$, we have $e - (g + h) < \frac{q_{Cost_{textual}} - f_{i(Cost_{textual})}}{\alpha}$, i.e. $dist(f_{il}, u_i) - dist(q_l, u_i) < \frac{q_{Cost_{textual}} - f_{i(Cost_{textual})}}{\alpha}$ α, we have $Cost_{total}(q, u_i) > Cost_{total}(f_i, u_i)$. When u_i is inside the left area of H_l, it is easy to prove $Cost_{total}(q, u_i) < Cost_{total}(f_i, u_i)$ and we omit it.

When $q_{Cost_{textual}} < f_{i(Cost_{textual})}$, the proof is similar to above-proved case except that we considering the right branch H_r of H, thus we omit it due to space limit.

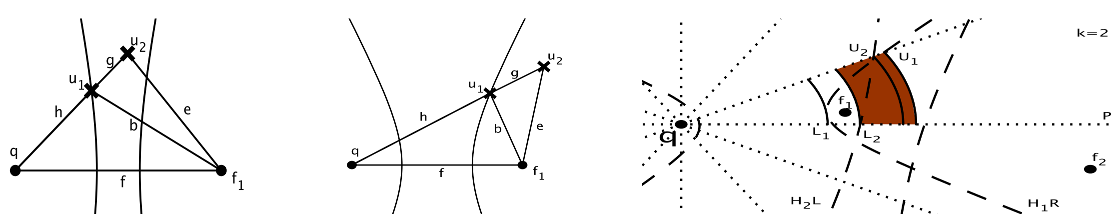
□

Algorithm Our algorithm for *WRkNN* is motivated by SLICE [14] consisting of filtering phase and verification phase. *Filtering phase:* like SLICE [14], we partition universe first. Then for each f_i, we take one branch H_h of its hyperbola $H_{f_i,q}$ w.r.t. q by considering the textual costs between q and f_i. Following, H_h contributes a lower arc and an upper arc (if possible) for the slice it passes. We apply pruning rules in SLICE [14] to filter more facilities. After all facilities are accessed, each slice maintains a *k*-th lower arc and k-th upper arc and a set of candidate facilities to verify users. Figure 6c *Verification phase:* The filtering phase is similar to SLICE, and we omit it here.

6 Experimental Study

6.1 Experimental Setup

We compare our algorithm of RkNN with InfZone [14] and SLICE [2]. All algorithms are implemented in C++, and the experiments are run on a 64-bit PC with Intel Xeon 2.66 GHz quad CPU and 4 GB memory running Linux. We use the synthetic and real datasets in experiments. The real dataset consists of 175,812 points in North America, and we randomly divide it into two sets of equal size as facility



(a) Lemma 4 case1 **(b)** Lemma 4 case2 **(c)** WRkNN filtering

Fig. 6 Lemma 4 and WRkNN filtering. **a** Lemma 4 case1, **b** Lemma 4 case2, **c** WRkNN filtering set and user set. For synthetic datasets, each dataset consists of 50,000, 100,000, 150,000, 200,000 points following either uniform or normal distribution and the facility sets, user sets are obtained like real dataset. The default synthetic dataset contains 100,000 points following Normal distribution. For k, we set it from 1 to 25 and make 10 at the default value.The page size for R-tree used in our experiment is 4096 bytes.

As big indices are kept in disc practically, I/O cost cannot be avoided when processing query. Therefore, a penalty of I/O will be charged for each communication between disc and CPU. In our study, we estimate the I/O cost [17, 18] by 0.1 ms which is the lowest time cost at the moment. We calculate the total time cost of each query in experiments by:

$$T = Cost_{I/O} * P_{I/O} + Cost_{CPU} \quad (2)$$

where $Cost_{I/O}$, $Cost_{CPU}$ are I/O cost, CPU time cost and $P_{I/O}$ is the I/O penalty. Through experimental study, our algorithm runs times faster than InfZone and improves SLICE performance significantly.

6.2 Evaluating Query Performance

In this section, we compare performance of three algorithms. Three algorithms InfZone, SLICE and our pre-computed one are shown as INF, SLICE and PRE, respectively. The number of partition for SLICE and PRE is 12 and 9 based on experiments in [16]. The experimental results shown in this subsection are an average cost for one query in terms of total time (in millisecond) and I/O.

Effect of data size and various k values In Fig. 7, we study the effect of data size and various k values for *RkNN*. Figure 7a, b show average total time and I/O cost for both filtering and verification phases. For INF and SLICE, the major cost in Fig. 7a is filtering phase whereas PRE which improves the filtering phase by one time faster by pre-computing guardian sets narrows the difference between two phases. Figure 7c, d shows effects of k on *RkNN*. In Fig. 7c, numbers displayed above the bars correspond to the number of *I/O*s. Figure 7d shows results for normal distribution using lines to demonstrate clearly how algorithms scale with the increasing k. Under our experimental setting, PRE absolutely outperforms other algorithms.

Effect of data distribution In Fig. 8, we study the effect of data distribution on algorithms. Distribution of the facilities and users is shown as (D_f, D_u) where D_f and D_u correspond to distributions of facilities and users. **U**, **R** and **N** stand for uniform, real and normal distributions, respectively. The synthetic datasets contain the same number of points as real dataset. Figure 8 demonstrates PRE outperforms others whatever combination of data distributions used.

6.3 Experimental Study Over R*k*NN Variants

Fixed Value Reverse k Nearest Neighbour Queries To study the performance of our algorithm, we create a baseline algorithm by running *SLICE* for *RkNN* and picking users that q is their exact k-th nearest neighbours. We use *SLICE* and *PRE* to indicate baseline algorithm and our one in experimental figures and all experimental results are collected by conducting 100 queries.

Effect of dataset size and various k values Figure 9a, b shows two algorithms' performances over various datasets. In Fig. 9a, PRE runs about 50% faster than SLICE.Besides,our algorithm saves I/O cost in filtering phase in PRE,

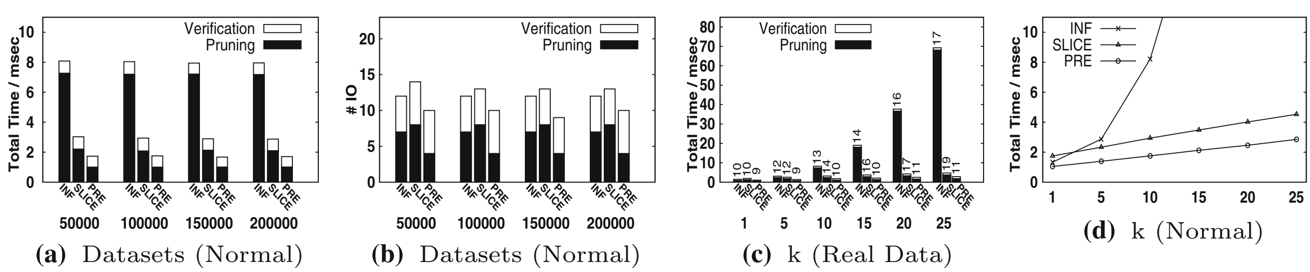
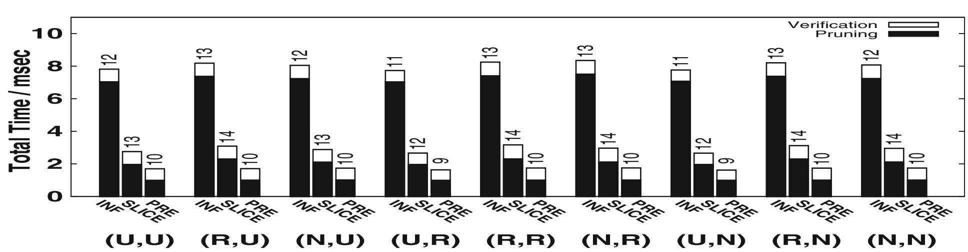
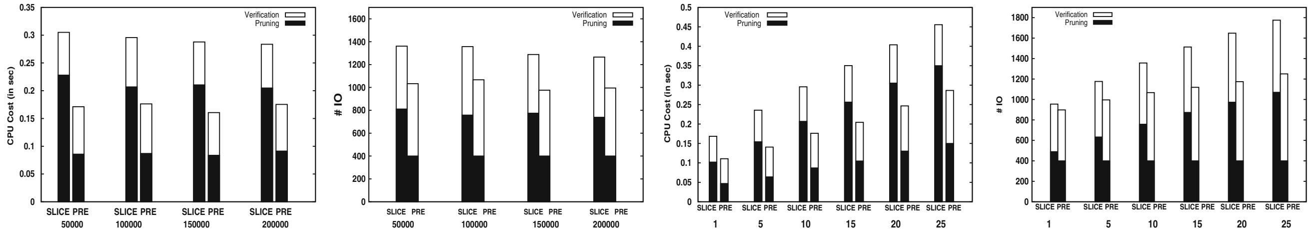


Fig. 7 RkNN effect of data set size and k. **a** Datasets (Normal), **b** Datasets (Normal), **c** k (Real Data), **d** k (Normal)

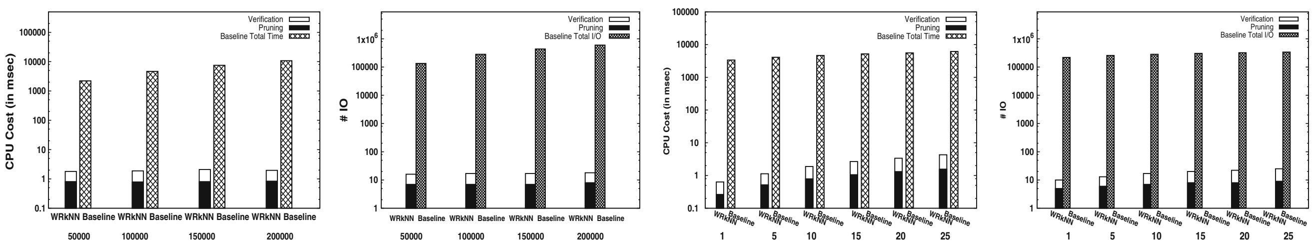
Fig. 8 RkNN effect of data distribution





(a) Datasets **(b)** Datasets **(c)** k **(d)** k

Fig. 9 FRkNN effect of datasets and k (Normal Distribution). **a** Datasets, **b** Datasets, **c** k, **d** k



(a) datasets **(b)** datasets **(c)** k **(d)** k

Fig. 10 WRkNN Effect of datasets and k, **a** datasets, **b** datasets, **c** k, **d** k

as demonstrated in Fig. 9b. Relatively, the verification phase becomes the major PRE cost in terms of I/O. Figure 9c, d demonstrates the PRE is much more efficient than SLICE in time and I/O w.r.t. various k values. The major reasons leading to such results are: **1** pre-computation's filtering over facilities; **2** The lemma 3 prunes a number of users in verification phase.

Weighted Reverse k Nearest Neighbour Queries To study our algorithm's performance, we implement a baseline algorithm based on Brute-Force. In our *WRkNN*, to make the difference of products prices stay in a reasonable range, we set the lowest price is 20% cheaper the highest one. We set the convert factor 1, and the space is partitioned into 12 through preliminary experimental study. We use *Baseline* and *WRkNN* to indicate baseline algorithm and our one in experimental figures.

Effect of datasets and various k values In Fig. 10, we study performance of *WRkNN* over various datasets and k values. As *WRkNN* applies pruning rules in filtering phase and the bounding arcs reduce the number of users to be verified significantly, our WRkNN performs better than the baseline algorithm. Figure 10a, b demonstrates *WRkNN* outperforms the baseline over 4 and 5 magnitudes in terms of the processing time and I/O cost, respectively. Figure 10c, d illustrates *WRkNN*'s performance over different k values. Given a large k value, *WRkNN* costs much time and more I/Os due to pruning rules become less powerful relatively. But our algorithm still outperforms the baseline one by a large margins.

7 Conclusion

In this paper, we propose a novel pre-computed R*k*NN algorithm by computing guardian set for each disjoint rectangular region *R* in universe and guarantee that for each possible query *q* in *R*, all its R*k*NN results are only affected by those facilities in guardian set. Through pre-computation, we reduce the number of facilities to be accessed from the whole facility set to only tens before processing query by SLICE. Besides, we come up with two useful variants of R*k*NN and proposed algorithms. The extensive experimental study demonstrates our algorithms outperform all existed algorithms.

Acknowledgements Research of Wei Wang is supported by ARC DP130103401 and DP130103405. Muhammad Aamir Cheema is supported by ARC DE130101002 and DP130103405.

References

1. Yang S, Cheema MA, Lin X, Wang W (2015) Reverse k nearest neighbors query processing: experiments and analysis. Proc VLDB Endow 8(5):605–616

2. Yang S, Cheema MA, Lin X, Zhang Y (2014) Slice: reviving regions-based pruning for reverse k nearest neighbors queries. In: IEEE 30th international conference on Data engineering (ICDE), pp 760–771
3. Tao Y, Papadias D, Lian X (2004) Reverse kNN search in arbitrary dimensionality. In: Proceedings of the thirtieth international conference on Very large data bases- VLDB endowment, vol 30, pp 744–755
4. Stanoi I, Agrawal D, El Abbadi A (2000) Reverse nearest neighbor queries for dynamic databases. In: ACM SIGMOD workshop on research issues in data mining and knowledge discovery, pp 44–53
5. Yang C, Lin KI (2001) An index structure for efficient reverse nearest neighbour queries. In: Proceedings of 17th international conference on data engineering, pp 485–492
6. Tao Y, Papadias D, Lian X, Xiao X (2007) Multidimensional reverse kNN search. VLDB J 16(3):293–316
7. Papadias D, Tao Y, Lian X, Xiao X (2007) Multi-dimensional reverse kNN search. Int J VLDB 6(3):293
8. Cao X, Chen L, Cong G, Jensen CS, Qu Q, Skovsgaard A, et al (2012) Spatial keyword querying. In: Conceptual modeling. Springer, Berlin, pp 16–29
9. Cao X, Cong G, Jensen CS, Ooi BC (2011) Collective spatial keyword querying. In: Proceedings of the 2011 ACM SIGMOD international conference on management of data pp 373–384
10. Cheema MA, Brankovic L, Lin X, Zhang W, Wang W (2010) Multi-guarded safe zone: an effective technique to monitor moving circular range queries. In: IEEE 26th international conference on data engineering (ICDE), pp 189–200
11. Cheema MA, Zhang W, Lin X, Zhang Y (2012) Efficiently processing snapshot and continuous reverse k nearest neighbors queries. VLDB J 21(5):703–728
12. Korn F, Muthukrishnan S (2000) Influence sets based on reverse nearest neighbor queries. In: ACM SIGMOD Record vol 29, pp 201–212
13. Wu W, Yang F, Chan CY, Tan KL (2008) Finch: evaluating reverse k-nearest-neighbor queries on location data. In: Proceedings of the VLDB Endowment, vol 1, pp 1056–1067
14. Cheema MA, Lin X, Zhang W, Zhang Y (2011) Influence zone: Efficiently processing reverse k nearest neighbors queries. In: IEEE 27th international conference on data engineering (ICDE), pp 577–588
15. Güting RH (1994) An introduction to spatial database systems. Int J VLDB 3(4):357–399
16. Song W, Jianbin Q, Wei W Cheema MA (2016) Pre-computed region guardian sets based reverse kNN queries. In: International conference on database systems for advanced applications. Springer International Publishing, pp 98–112
17. Ruemmler C, Wilkes J (1993) UNIX disk access patterns. In: USENIX Winter, vol 93, pp. 405–420
18. Tsirogiannis D, Harizopoulos S, Shah MA, Wiener JL, Graefe G (2009) Query processing techniques for solid state drives. In: Proceedings of the 2009 ACM SIGMOD international conference on management of data, pp 59–72