Pre-computed Region Guardian Sets Based Reverse kNN Queries

Wei Song[1], Jianbin Qin[1], Wei Wang[1], and Muhammad Aamir Cheema[2]

[1]The University of New South Wales, Sydney, Australia
[2]Monash University, Melbourne, Australia
[1]{wsong,jqin,weiw}@cse.unsw.edu.au
[2]{aamir.cheema}@monash.edu

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 realise 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 Rk'NN results are only affected by the objects in R's guardian set, where $k' \leq k$. 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. Our comprehensive experimental study on synthetic and real datasets demonstrates the proposed approach is the most efficient algorithm for RkNN.

Keywords: RkNN, Pre-computation, Guardian Set, SLICE

1 Introduction

Reverse k nearest neighbour queries (RkNN) are classified into *bichromatic* RkNN and *monochromatic* RkNN.

Bichromatic RkNN Given a set of facilities F, a set of users U and a query $q \in F$, the Bichromatic RkNN (denoted as $biR\text{k}NN$) returns every user $u \in U$ for which q is one of its k-closest facilities.

Example : For a given McDonald's q, the people for which q is one of their k-closest McDonald's restaurants are its $biR\text{k}NN$. These people are its potential customers and can be attracted by targeted marketing. In this paper, the objects providing some service (e.g., McDonald's, supermarkets) are called $facilities$ and the objects that use the facilities (e.g., residents, customers) are called $users$.

Monochromatic RkNN Given a set of facilities F and a query $q \in F$, the Monochromatic RkNN (denoted as $monoR\text{k}NN$) returns every facility f for which q is one of its k-closest facilities.

Example : Consider the example of hospitals. Given a hospital q, its $monoR\text{k}NN$ are the ones for which q is one of their k nearest hospitals. Such hospitals may seek assistance (e.g., blood, staff) from q in case of emergencies.

Like most of the existing work on RkNN, we address the problem in a Euclidean space. Our proposed algorithm can by applied to both $monoR\text{k}NN$ and $biR\text{k}NN$. For clear presentation, we focus on the $biR\text{k}NN$ unless mentioned specifically.

As shown in a recent experimental study [15], SLICE is the state-of-the-art RkNN algorithm. Like most other RkNN algorithms, SLICE consists of two phases namely **filtering phase** and **verification phase**. SLICE's filtering phase dominates the total query processing cost [3]. We observe that SLICE needs to access many unnecessary facilities in its filtering phase and this adversely affects the query performance.

Motivated by the above observation, 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 $F_g \subseteq F$ such that the results of every Rk'NN query q that lies in R (and $k' \leq k$) can be computed using only the facilities in F_g. The set of objects F_g is called the guardian set of R and a facility $f \in F_g$ is called a guardian facility of R. During the query processing time, we determine the region R that contains q and then use SLICE on its guardian set instead of the whole dataset to compute the results. Since the size of guardian set is significantly smaller than the whole dataset, this approach significantly improves the performance as demonstrated in our experimental study.

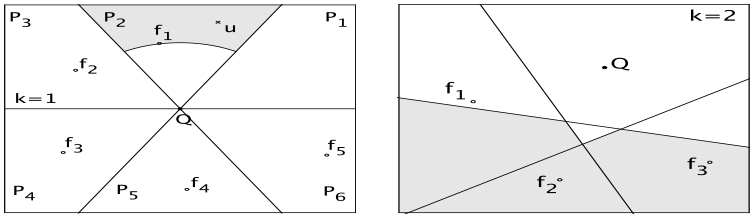
We remark that although there exists other pre-computation based approaches, our approach is unique in that its pre-computation does not depend on the set of users. For example, the technique proposed in [7] pre-computes, for each user u_i, its k-th closest facility f_k and creates a circle C_i centred at u_i with radius $dist(u_i, f_k)$. All such circles are indexed by an R-tree and a Rk'NN query q for which $k' \leq k$ is answered using the circles that contain q. A disadvantage of such approach is that any change in the set of users U requires updating or reconstructing the index. On the other hand, our guardian sets do not depend on the set of users 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, fuel stations) is less common as compared to the locations of users (e.g., people, cars).

Next, we summarise 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. The proposed index can be used to answer any Rk'NN query for which $k' \leq k$.
- Our comprehensive experimental study on real and synthetic datasets demonstrates that our algorithm significantly improves SLICE in query processing cost and outperforms all existing RkNN algorithms.

The rest of the paper is organized as follows. We introduce related works in section 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 and prove each region's guardian set F_g can be used flexibly for any k' s.t. $k' \leq k$. Section 4 briefly recalls SLICE to do the experimental study in section 5. We conclude this paper in section 6.

2 Related works



(a) Six regions (b) Influence zone

Fig. 1. Related works

Six-Regions[2] is a region-based technique proposed for RkNN. It consists of two phases, namely **Filtering phase** and **Verification phase**. In filtering phase, Six-Regions centres 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 centering 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. 1(a)) cannot be a result and only users lie under the arc of its region can be returned as candidates to be verified.

Influence Zone[1] is a half-space based technique proposed for RkNN, 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. 1(b)) cannot return any result. $InfZone$ guarantees every u in the unpruned area is a result. Such unpruned area is called **Influence Zone**.

SLICE[3] is the most efficient algorithm before our work for RkNN, which is integrated in our query processing algorithm in this paper. We introduce SLICE in detail in section 4.

3 Techniques

As motivated by [1] that given a query q and a k, we may compute a set of facilities that q's RkNN is only affected by such facilities. Similarly, given a

rectangular region R and k, we compute such a set F_g of facilities that for any $q \in R$, all facilities affecting q's RkNN are contained in F_g and the rest facilities $f \notin F_g$ cannot affect any q's RkNN. As for any $q \in R$, whose RkNN results are only guarded by R's F_g. We define F_g **guardian set** and $f_g \in F_g$ **guardian facility** w.r.t. rectangular region R.

Therefore, 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 $\bigcup R_{i=1...n} = U$, we can compute and obtain every guardian set F_{gi} of R_i.

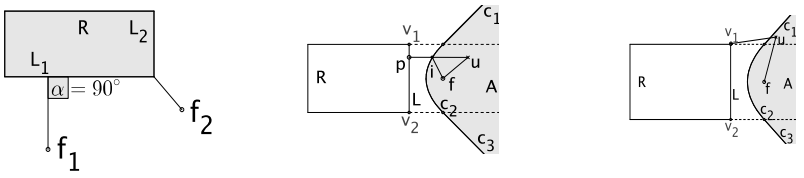
Next, we present our techniques to compute R's guardian set F_g w.r.t.the value of k.

3.1 Computing Guardian Set of a Rectangular Region

First, we give some definitions and Lemmas.

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 RkNN results are only affected by $f \in F_g$. For any facility $f \notin F_g$, it does not affect q's RkNN 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 2(a).*



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

Fig. 2. Definition 2 & Lemma 1 proof

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 vertices of L respectively toward 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 and 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)), we set 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. According to the property of parabola, $|pi| = |if|$, then $|up| = |ui| + |if|$. By triangle inequality, $|up| > |uf|$.

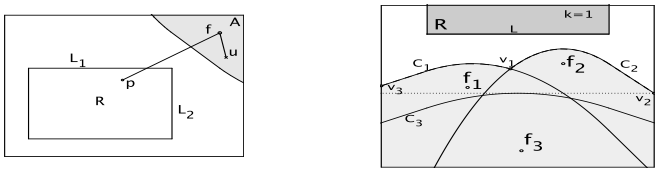
Case 1.2: For u lies in A but is out of L's range (shown in Fig. 2(c)), the minimum distance from u to L is $|uv_1|$, where v_1 is the vertex of L locates at the same side with u w.r.t.f. As u is in the half dominated by f, $|uf| < |uv_1| \leq |up|$.

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 $mindist(u, R) = |v_1u| < |up|$, and $|uf| < |v_1u|$, we have $|up| > |uf|$.

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 toward 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

With Lemma 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. 3(b) 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 Section 3.1, we partition the universe U to small rectangular regions R. It is not avoidable that each R contains facilities. However,

Algorithm 1: ConstructGuardianSet(R, F, k)

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

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 [6] 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 Section 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.

Theoretical Analysis : [**How many regions to be computed for guardian sets**] As each time a region R_u is split into four smaller regions R_l if it contains at least $|T_f| + 1$ facilities. Therefore the whole universe has at most $\log_4(\frac{|N|}{|T_f|+1}) + 1$ level, where $|N|$ is the total number of facilities. At the lowest level that is level $\log_4(\frac{|N|}{|T_f|+1})+1$, there is no more region to be split. As a result, the total number of regions to be computed for guardian sets is $4^{\log_4(\frac{|N|}{|T_f|+1})+1}$, which is $4\frac{|N|}{|T_f|+1}$.

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 ∈ F_g do
    compute and keep intersections between C_f and C_{f_g};
for each f_g ∈ 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;
```

3.3 Region R's guardian set w.r.t k is compatible for k' s.t. $k' \leq k$

Next, we prove R's guardian set of k can be used flexibly on any Rk'NN s.t. $k' < k$.

Lemma 3 *Given a rectangle region R, R's guardian set $F_{gk'}$ of k' is a subset of its guardian set F_{gk} of k, where $k' < k$.*

Proof. We prove Lemma 3 by contradiction. Assume there is at least one facility $f_{outside}$ excluded from F_g that is R's guardian facility of k', where $k' < k$. Therefore, the area pruned by $f_{outside}$ (denote $A_{foutside}$) holds:

$$((\bigcup A_{f_{gk}}) \cap A_{foutside}) \subset A_{foutside} \tag{1}$$

where f_{gk} is the guardian facility of R and whose partial combined curve contributes to the boundary of unpruned area of R w.r.t.k. However, as $f_{outside}$ is excluded from F_g of R, according to lemma 1 and 2, $((\bigcup A_{f_{gk}}) \cap A_{foutside}) = A_{foutside}$, which contradicts with equation 1, then Lemma 3 holds.

By Lemma 3, we mark facilities to each R's Rk'NN guardian set they are included. When given q, we retrieve R's guardian set w.r.t. k' ($k' \leq k$) to avoid accessing more facilities that have no influence on R, then start processing query by SLICE.

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 [3], the best

partition number is 12 but it is 9 in our algorithm according to our preliminary experimental study.

Fig. 4 shows an example where the space is divided into 9 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.

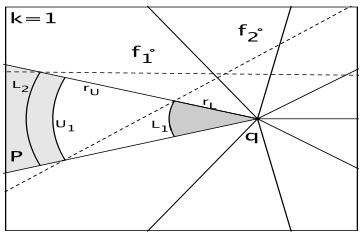


Fig. 4. SLICE filtering

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 remain users will be checked by significant facilities of this region.

Algorithm 3: RkNN($IndexFile, q, k$)

Input: IndexFile, query point q,value k
Output: RkNN results
1 locate q's region R;
2 retrieve R's guardian set F_gw.r.t. k;
3 invoke $SLICE$ with F_g;
4 return RkNN results;

5 Experimental Study

5.1 Experimental Setup

We compare our algorithm with InfZone [1] and SLICE [3] which are the latest algorithms for RkNN and SLICE is also the most efficient algorithm before our pre-computed one. All algorithms are implemented in C++ and the experiments are run on a 64-bit PC with Intel Xeon 2.66GHz quad CPU and 4GB 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. One of them is facility set, the other one is 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 unless mentioned otherwise. 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 disk practically, I/O cost cannot be avoided when processing query. Therefore a penalty of I/O will be charged for each communication between disk and CPU. In our study, we estimate the I/O cost [16], [17] by 0.1ms which is the lowest time cost at the moment. We calculate the total time cost of each query in experiments by:

$$T = Cost_{I/O} * P_{I/O} + Cost_{CPU} \tag{2}$$

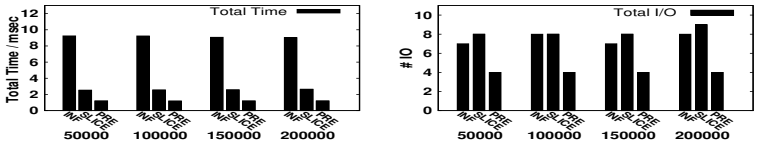
where $Cost_{I/O}$, $Cost_{CPU}$ are I/O cost, CPU time cost and $P_{I/O}$ is the I/O penalty. Through experimental study, our algorithm runs times faster than InfZone and improves SLICE performance significantly. Although [15] claims the I/O cost is highly system specific, we will prove our algorithm is the most efficient whatever I/O penalty charged and our algorithm outperforms others much more largely when higher I/O cost charged.

5.2 Evaluating Query Performance

In this section, we compare performance of three algorithms on both $monoR\text{k}NN$ and $biR\text{k}NN$. 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 are 12 and 9 based on [3] and our preliminary experiments (see Fig. 11(a)). The experimental results shown in this sub-section is an average cost for one query in terms of total time (in millisecond) and I/O.

Effect of data size: In Fig. 5 and 6, we study the effect of data size for both $monoR\text{k}NN$ and $biR\text{k}NN$. For $monoR\text{k}NN$, Fig. 5(a) and 5(b) show the average total time and I/O cost. In $biR\text{k}NN$, Fig. 6(a) and 6(b) show average total time and I/O cost for both filtering and verification phases.

In Fig. 5(a), PRE performs best as only guardian facilities f_g of the region R where query locates are accessed. SLICE starts with considering all facilities in

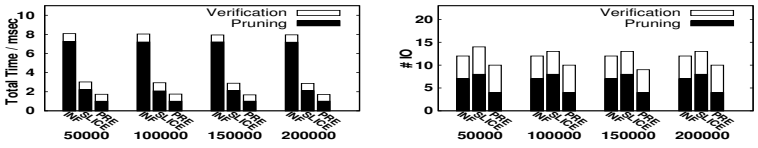


(a) Number of facilities (b) Number of facilities

Fig. 5. Monochromatic Queries: Effect of data set size (Normal Distribution)

filtering phase, costing more time. $InfZone$ costs the most time as every facility to be pruned needs to compute with every vertex of Influence Zone, which is time-consuming.

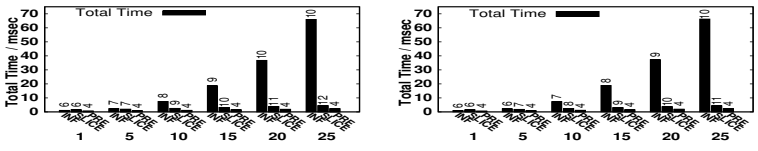
Fig. 5(b) demonstrates PRE costs least I/O as all guardian facilities of a region can be indexed in one disk page and can be located by a small constant number of I/Os. As expected, the I/O cost of SLICE is slightly larger than InfZone's as InfZone prunes a larger area.



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

Fig. 6. Bichromatic Queries: Effect of data set size (Normal Distribution)

Fig. 6 shows results for $biR\text{k}NN$. For INF and SLICE, the major cost in Fig 6(a) 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.

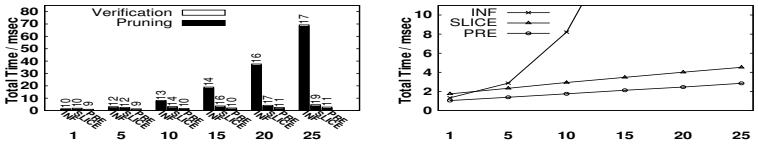


(a) Varying k (Real Data) (b) Varying k (Normal Distribution)

Fig. 7. Monochromatic Queries: Effect of k

Due to space limitations, in the rest part, we focus on the total time cost (**I/O cost included**) of algorithms. **The numbers displayed above the bars correspond to the number of I/Os unless mentioned otherwise**.

Effect of k In Fig. 7 and 8, we study the effect of k on $monoR\text{k}NN$ and $biR\text{k}NN$. The performance of InfZone deteriorates rapidly with the increasing k as time complexity is $O(km^2)$ and m increases as k increases, where m is the number of influence zone's vertices. PRE still performs best with least cost on both total time and I/O.



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

Fig. 8. Bichromatic Queries: Effect of k

Fig. 8(b) shows results for normal distribution using lines to demonstrate clearly how algorithms scale with the increasing k. Under our experimental setting, with accessing less facilities in filtering phase, PRE absolutely outperforms other algorithms.

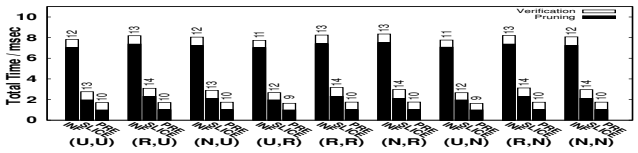


Fig. 9. Effect of Data Distribution

Effect of data distribution: In Fig. 9, 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** correspond to Uniform, Real and Normal distribution respectively. In this experiment, the synthetic datasets contain the same number of points as real dataset. Fig. 9 demonstrates our algorithm outperforms others whatever combination of data distributions used.

Effect of number of users relative to number of facilities: In this experiment, we fix the number of facilities to 100,000 and change the number of users to see the effect of change in the relative size of the two data sets. Fig. 10 shows that all three algorithms' verification phases cost more with increasing number of users. PRE's filtering phase is much efficient making its total time cost dominated by verification phase. For the rest two, filtering phase dominates the total time cost as it is still costly. Overall, PRE cost least I/O and runs much faster than InfZone and SLICE.

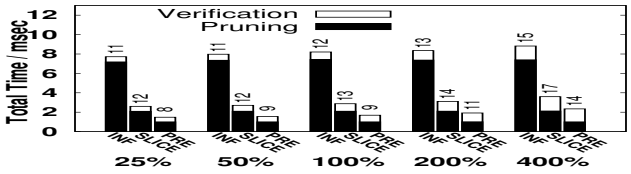
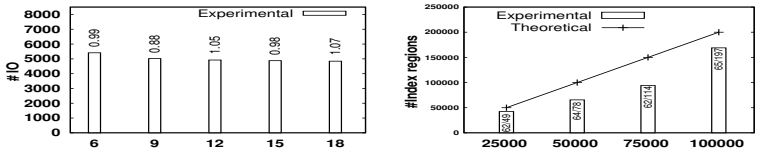


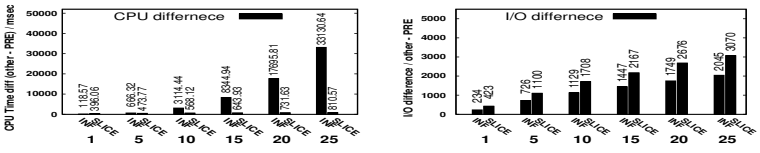
Fig. 10. users = x%facilities



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

Fig. 11. Evaluation on Pre-computation

Total Time Cost and I/Os To clearly demonstrate our pre-computation algorithm change under different partitions, we process 500 queries and collect the total I/O and total time cost. Fig. 11(a) shows the total I/O cost for 500 queries and the number above each bar is the total time cost in second. We conclude that when partition number is 9, our algorithm reaches a best performance as when partition number is smaller than 9, verification phase takes much time due to only a small area pruned by filtering phase. Although its pruning power becomes stronger with larger partition number, the filtering phase is still costly and affects the performance.



(a) k (b) k

Fig. 12. CPU and I/O difference

5.3 Evaluating Pre-Computation Algorithm

Number of pre-computed rectangular regions Fig. 11(b) shows the number of regions computed practically. Experimental results are under our theoretical analysis in section 3.2, demonstrating the number of regions computed practically is bounded by our theoretical analysis. The number inside each bar

is the average number of guardian facilities in the guardian set followed by the size of index file in MB.

5.4 PRE is efficient whatever I/O penalty charged:

Following equation 2, let total time of PRE : $T_{PRE} = Cost_{PRE.I/O} * P_{I/O} + Cost_{PRE.CPU}$. T' denotes total time cost of any other algorithm : $T' = Cost'_{I/O} * P_{I/O} + Cost'_{CPU}$. The total time difference between other algorithm and PRE is $\delta = T' - T_{PRE}$:

$$\delta = (Cost'_{I/O} - Cost_{PRE.I/O}) * P_{I/O} + (Cost'_{CPU} - Cost_{PRE.CPU}) \quad (3)$$

Then we prove PRE is the most efficient algorithm processing RkNN whatever I/O penalty charged.

Proof : Proving PRE is the most efficient is equivalent to prove $\delta > 0$, i.e.

$$P_{I/O} > \frac{Cost_{PRE.CPU} - Cost'_{CPU}}{Cost'_{I/O} - Cost_{PRE.I/O}} \quad (4)$$

Fig. 12(a) shows CPU time cost difference and Fig. 12(b) shows I/O difference by using corresponding cost of other algorithm minus ours. To clearly show results, we collect data by processing 500 queries.

As shown in Fig. 12(a) and 12(b), PRE cost least I/O and CPU time, which makes right part of inequality 4 become small than 0, as the I/O penalty is positive, the inequality always holds whatever is the I/O cost. As a result, PRE outperforms other RkNN algorithms no matter what I/O penalty charged.

Total time difference with larger $P_{I/O}$ As equation 3 is a δ's function of $P_{I/O}$, which increases monotonically with the increasing I/O penalty. Thus, PRE will outperform other algorithm by a larger margin with larger I/O penalty charged.

According to [15]'s claim that SLICE is the most efficient RkNN algorithm before us, we conclude that PRE outperforms all existed RkNN algorithms.

6 Conclusion

In this paper, we propose a novel pre-computed RkNN 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 Rk'NN results are only affected by those facilities in guardian set, where $k' \leq k$. 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. The extensive experimental study demonstrates our algorithm outperforms all existed algorithms on both total time and I/O cost.

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

References

1. Cheema, M. A., Lin, X., Zhang, W., Zhang, Y.: Influence zone: Efficiently processing reverse k nearest neighbors queries. In: 27th International Conference on Data Engineering (ICDE), (pp. 577-588), IEEE (2011)
2. Wu, W., Yang, F., Chan, C. Y., Tan, K. L.: Finch: Evaluating reverse k-nearest-neighbor queries on location data. In: Proceedings of the VLDB Endowment, 1(1), (pp. 1056-1067), (2008)
3. Yang, S., Cheema, M. A., Lin, X., Zhang, Y.: Slice: Reviving regions-based pruning for reverse k nearest neighbors queries. In: 30th International Conference on Data Engineering (ICDE), (pp. 760-771), IEEE (2014)
4. Tao, Y., Papadias, D., Lian, X.: Reverse kNN search in arbitrary dimensionality. In: Proceedings of the Thirtieth international conference on Very large data bases-Volume 30 (pp. 744-755), VLDB Endowment (2004)
5. Stanoi, I., Agrawal, D., El Abbadi, A.: Reverse Nearest Neighbor Queries for Dynamic Databases. In: ACM SIGMOD workshop on research issues in data mining and knowledge discovery (pp. 44-53), (2000)
6. Gting, R. H. : An introduction to spatial database systems. In: The International Journal on Very Large Data Bases, 3(4), (pp. 357-399), (1994)
7. Korn, F., Muthukrishnan, S.: Influence sets based on reverse nearest neighbor queries. In: ACM SIGMOD Record (Vol. 29, No. 2, pp. 201-212), ACM (2000)
8. Yang, C., Lin, K. I.: An index structure for efficient reverse nearest neighbour queries. In: 17th International Conference on Data Engineering Proceedings. (pp. 485-492), IEEE (2001)
9. Tao, Y., Papadias, D., Lian, X., Xiao, X.: Multidimensional reverse kNN search. In: The VLDB Journal, 16(3), (pp. 293-316), (2007)
10. Papadias, D., Tao, Y., Lian, X., Xiao, X.: Multi-dimensional reverse kNN search. In: The international journal on very large data bases, 6(3), (pp. 293), (2007)
11. Cao, X., Chen, L., Cong, G., Jensen, C. S., Qu, Q., Skovsgaard, A., ... Yiu, M. L.: Spatial keyword querying. In: Conceptual Modeling (pp. 16-29), Springer Berlin Heidelberg (2012)
12. Cao, X., Cong, G., Jensen, C. S., Ooi, B. C.: Collective spatial keyword querying. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 373-384). ACM (2011)
13. Cheema, M. A., Brankovic, L., Lin, X., Zhang, W., Wang, W.: 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). IEEE (2010)
14. Cheema, M. A., Zhang, W., Lin, X., Zhang, Y.: Efficiently processing snapshot and continuous reverse k nearest neighbors queries. In: The VLDB Journal, 21(5), (pp. 703-728), (2012)
15. Yang, S., Cheema, M. A., Lin, X., Wang, W.: Reverse k nearest neighbors query processing: experiments and analysis. In: Proceedings of the VLDB Endowment, 8(5), (pp. 605-616), (2015)
16. Ruemmler, C., Wilkes, J.: UNIX disk access patterns. In: USENIX Winter (Vol. 93, pp. 405-420), (1993)
17. Tsirogiannis, D., Harizopoulos, S., Shah, M. A., Wiener, J. L., Graefe, G.: Query processing techniques for solid state drives. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (pp. 59-72), ACM (2009)