7-2008

Ranked Reverse Nearest Neighbor Search

Ken C. K. LEE
Pennsylvania State University

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

Wang-Chien LEE
Pennsylvania State University

DOI: https://doi.org/10.1109/TKDE.2008.36

Citation

LEE, Ken C. K.; ZHENG, Baihua; and LEE, Wang-Chien. Ranked Reverse Nearest Neighbor Search. (2008). *IEEE Transactions on Knowledge and Data Engineering*. 20, (7), 894-910. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/766

Ranked Reverse Nearest Neighbor Search

Ken C. K. Lee† Baihua Zheng‡ Wang-Chien Lee†
† Pennsylvania State University, USA. {cklee, wlee}@cse.psu.edu
‡ Singapore Management University, Singapore. bhzheng@smu.edu.sg

***Abstract*— Given a set of data points $\mathcal{P}$ and a query point q in a multidimensional space, Reverse Nearest Neighbor (RNN) query finds data points in $\mathcal{P}$ whose nearest neighbors are q. Reverse k-Nearest Neighbor (RkNN) query (where $k \geq 1$) generalizes RNN query to find data points whose kNNs *include* q. For RkNN query semantics, q is said to have influence to all those answer data points. The degree of q's influence on a data point p ($\in \mathcal{P}$) is denoted by κ_p where q is the κ_p-th NN of p. We introduce a new variant of RNN query, namely, *Ranked Reverse Nearest Neighbor* (RRNN) query, that retrieves t data points most influenced by q, i.e., the t data points having the smallest κ's with respect to q. To answer this RRNN query efficiently, we propose two novel algorithms, κ-Counting and κ-Browsing that are applicable to both monochromatic and bichromatic scenarios and are able to deliver results progressively. Through an extensive performance evaluation, we validate that the two proposed RRNN algorithms are superior to solutions derived from algorithms designed for RkNN query.**

I. Introduction

A. Definitions and Motivations

The Reverse Nearest Neighbor (RNN) search problem has received a lot of attentions from the database research community for its broad application base such as marketing, decision support, resource allocation and data mining since its introduction [8]. Given a set of data points $\mathcal{P}$ and a query point q in a multidimensional space, RNN query finds every data point in $\mathcal{P}$ with q as its nearest neighbor (NN). Such RNN query is also called *monochromatic* RNN since the answer data points and their NNs are all from the same set of data points, i.e. $\mathcal{P}$[1]. On the other hand, *bichromatic* RNN searches answer data points from one set of data points, $\mathcal{P}$, with their NNs taken from another set of data points, say $\mathcal{Q}$. Reverse k-Nearest Neighbor (RkNN) with $k \geq 1$ generalizes RNN to find data points whose kNN *include* q. RkNN query is different from (and even more complicated than) kNN query because of asymmetric NN relationship between two data points in a dataset. That means if a query point q has found the nearest neighbor point p ($\in \mathcal{P}$), p may have other data points else (i.e., other than q) as its nearest neighbors.

The primary goal of RkNN query is to determine the *influence set*, i.e., a subset of data points in $\mathcal{P}$ considered to be influenced by a given query point q if q is the immediate nearest neighbor to them. The term *degree of influence*, denoted as κ_p, is defined in Definition 1 to quantify the influence of a query point q on a data point p in $\mathcal{P}$. In this paper we assume data points and query point are in Euclidean space. Hence, when q is the NN to a data point p, q is said to have the most significant influence on p and the corresponding κ_p is one. When q is the second NN of another data point p', q is the second most influential point to p' and $\kappa_{p'}$ is two, and so on. Based on the definition of κ, RkNN query can be interpreted as to retrieve data points with their κ's not exceeding a given threshold parameter k as formally stated in Definition 2.

Definition 1: Degree of influence. Given a dataset $\mathcal{P}$ and a query point q, the degree of influence of q on p ($\in \mathcal{P}$) denoted by κ_p is the number of data points not farther than q to p. Formally, $\kappa_p = \left|\{p' \mid p' \in X \cup \{q\} \wedge dist(p', p) \leq dist(p', q)\}\right|$ where $X = \mathcal{P} - \{p\}$ (monochromatic) or $X = \mathcal{Q}$ (bichromatic)[2]. □

Definition 2: RkNN query. Given a dataset $\mathcal{P}$ (and $\mathcal{Q}$ when bichromatic RkNN is considered) and a query point q, RkNN query returns a set of objects whose κ's do not exceed k, an influence threshold setting, i.e., RkNN(q) $= \{p \mid p \in \mathcal{P} \wedge \kappa_p \leq k\}$. □

RkNN query has no control of the answer set size since the settings of k does not determine the answer set size. For example, as reported in [15], a monochromatic R1NN query in a two-dimensional (2D) space many return none or up to six answer data points. For high-dimensional space and bichromatic scenario, the number of answer data points can vary a lot. Besides, RkNN is not very informative about the influences of a query point on answer data points. It is hard to differentiate one answer data point from another upon influence received from the query point. Therefore, it is useful to determine an *influence rank*, a predetermined number of influenced data points (with their κ's provided) ordered by their κ's. This search has a wide application base. For example, a company has some limited quantity of product samples to send to potential customers for promotion. Assume that the promoted product, other competitors' products and customers' preferences are all captured as data points in a multidimensional feature space. Suppose that customers are more likely to purchase a product if it is closer to their preferences in the feature space. Given the number of available samples t, kNN query with $k = t$ finds customers who preferences match well with the product, but the product may not receive high ranks to those customers due to existence of other products. RkNN can be adopted to find potential customers. Independent of t, it cannot find exact t potential customers to send the samples. Besides, both kNN and RkNN cannot tell which potential customers are the most (or least) suitable targets. This necessitates a new query that searches the t most influenced data points ranked based on the degree of influence.

In this paper, we propose *Ranked Reverse Nearest Neighbor* (RRNN) query, formally defined in Definition 3, to retrieve from $\mathcal{P}$ the t data points most influenced by a query point q, where t is a query parameter. When t is set to 1, RRNN query returns a data point p that q has the most influence on. Notice that κ_p may not necessarily be one. When $t = |\mathcal{P}|$ (i.e., the cardinality of the object set), RRNN renders a sorted list of all data points according to their degrees of influence. Since κ's are not necessarily unique, the distance between the data points and the query point is used as the tiebreaker. Revisit our previous example. An RRNN query with t set to the number of available samples, say 100, one hundred customers best matched with the promoted product are retrieved.

Definition 3: RRNN query. Given a dataset $\mathcal{P}$ (and $\mathcal{Q}$ when bichromatic scenario is considered), a query point q, and a requested number of answer data points t, *Ranked Reverse Nearest Neighbor* (RRNN) query returns t tuples (p, κ_p) where $p \in \mathcal{P}$, and κ_p is p's degree of influence. Formally, $RRNN_t(q) = \{(p, \kappa_p) \mid p \in \mathcal{P}' \wedge |\mathcal{P}'| = t \wedge \mathcal{P}' \subseteq \mathcal{P} \wedge \forall_{x \in (\mathcal{P} - \mathcal{P}')} \kappa_p < \kappa_x\}$. □

[1]For the rest of this paper, we refer to the data points in the answer set as *answer data points*

[2]$dist(x, y)$ denotes the Euclidean distance between x and y.

The RRNN query, a new RNN variant, is functionally more powerful and more informative than RkNN as it can report the top-t most influenced data points with their degrees of influence. This RRNN supports impact analysis as well. Let us consider other examples. A logistic company plans to set up a service center at a given location. An impact analysis based on geographical proximity to their customer bases may be performed at the planning stage. Assume that customers' preferences for logistic services are based on distance. RRNN can show the distribution of impact within a specified number (or percentage) of most influenced subjects. For example, among the top-100 potential customers, how the new center is ranked among existing centers. In this case, RkNN can only figure out the set of potential customers within a specified impact controlled by k. Another interesting RRNN application is in the matching service. When a new member joins, a group of existing members who may be interested in the new member, can be notified by running RkNN query based on calculated matching degree. RRNN query can identify a given number of top-matched candidates, along with their corresponding matching degrees.

B. Possible Solutions

Although RkNN query is also based on the degrees of influence (see Definition 3 and Definition 2), none of the existing algorithms proposed for RkNN search can be directly adopted to efficiently support the RRNN query. An intuitive approach, called *κ-Probing*, is to iteratively invoke an RkNN algorithm by increasing the query parameter k from 1 until t most influenced data points are obtained. First, an R1NN query (where $k = 1$) is first evaluated. The answer set is recorded and the corresponding κ of each answer data point is 1. Next, an R2NN query is reissued. Notice that the query result of the R2NN query subsumes that of the R1NN query in the previous run. Therefore, the answer data points excluding those obtained from the previous run have their corresponding κ's equal to 2. This process repeats with incremented k at each run until t answer data points and their κ's are obtained, which clearly suffers from redundant processing among different runs. A slight improvement can be made by exponentially increasing the k's in the series of RkNN invocations, e.g., setting k to 1, 2, 4, 8 ... etc. If more than t answer data points are collected, the algorithm gradually reduces k to smaller values until t answer data points are found.

From our analysis of the RRNN query, the degree of influence with respect to the query point q for a data point p, κ_p, can be determined by counting the number of data points closer to p than q (i.e. p's NNs). If we draw a circle $cir(p, q)$ rooted at p using distance between p and q, i.e., $dist(p, q)$, as the radius, κ_p equals the number of data points, including q, fallen inside the circle. Thus, a naive approach to processing an RRNN query is to count the numbers of NNs for all data points exhaustively as above mentioned. However, an RRNN query is only interested in the t top-ranked data points most influenced by q. Consequently, it is a waste to figure out the κ's for all the other points. In other words, we should only evaluate a set of potential candidates. This fosters a straightforward approach called *filter-and-rank (FR)*, serving as a baseline in this paper. FR is similar to the filter-and-refinement query processing paradigm commonly used by RkNN search algorithms (to be discussed in Section II). It has two phases: 1) in the filter phase, it retrieves the top K NN data points ($t \leq K$) to a query point as result candidates; 2) then in the rank phase, for each candidate, p, a circle $cir(p, q)$ is formed and the number of data points (i.e., κ_p) inside $cir(p, q)$ is derived. At last, t candidates with the smallest κ's are returned. However, this approach cannot guarantee the result accuracy. It may return an inaccurate result if K is not large enough to cover all potential answer points (i.e., false miss) in the filter phase and thus some other data points among candidates are mistaken as t most influenced data points, rendering incorrect results. Setting K to a large value may avoid false miss but this makes the search suffer a serious performance penalty.

C. Our Proposed Algorithms

Motivated by the value of the RRNN query and the lack of efficient algorithms, in this paper, we propose two novel and efficient algorithms, namely *κ-Counting* and *κ-Browsing*, that progressively obtains κ's for a subset of the data points. The key difference between these two algorithms lies in the adopted ordering functions and the number of data points visited to process the query.

Since data points with small κ's intuitively have short distances to q (i.e, small circles formed), the κ-Counting algorithm examines data points based on *their distance to the query point q*. While we determine the κ of one data point at a time, the κ's of many other data points are incrementally obtained based on findings of the data point under processing. The algorithm elegantly explores the property of the index structure to determine the access order of data points. However, because of asymmetric NN relationship, data points having short distance to the query point might not necessarily have small κ's and hence are excluded from the answer set. Thus, this algorithm, based on distance order, may require to process more data points. The details about κ-Counting will be discussed in Section III.

The κ-Browsing algorithm aims at optimizing the number of data points processed by visiting data points in the order of their degrees of influence (i.e., κ). A notion of $min\kappa$ is introduced and used in the algorithm to facilitate efficient processing of the RRNN query. The $min\kappa$ of a data point is a low bound estimation of κ based on distance metrics and aggregated counts on aR-tree [10]. Several heuristics are obtained via the knowledge of $min\kappa$ to prune the search space and to retrieve answer data points. Details about $min\kappa$ and developed optimization techniques for κ-Browsing are discussed in Section IV.

Both the κ-Counting and the κ-Browsing algorithms support multidimensional datasets. Other than R-tree/aR-tree maintenance, they do not incur any pre-processing overhead, making our algorithms suitable for highly dynamic environments. Moreover, our algorithms are I/O efficient as they look up a required portion of an index only once. Besides, our design of algorithms is compatible to both monochromatic and bichromatic application scenarios. Further, they can support RkNN with minor modification and provide progressive result delivery, which was not achieved by existing RNN/RkNN algorithms. To validate our proposals, we conduct a comprehensive set of experiments via simulation with a wide range of settings, such as different cardinality/dimentionality of the dataset and various values of t (the required number of answer objects). The result indicates that the κ-Browsing algorithm generally performs the best in terms of I/O costs and elapsed time.

D. Organization of the Paper

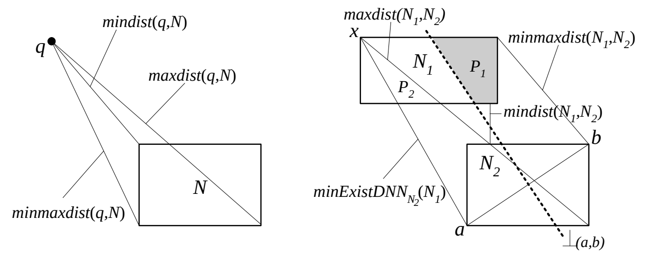
The remainder of the paper is organized as follows. Section II reviews the R-tree and existing RNN/RkNN search algorithms. Section III and Section IV present κ-Counting and κ-Browsing algorithms, respectively. For ease of illustration, the discussion of the algorithms is based on a 2D space, while our algorithms can support RRNN search in a multidimensional space. The performance evaluation of our algorithms is conducted and presented in Section V. Finally, Section VI concludes this paper.

II. Related Work

This section briefly reviews R-tree [6], an efficient index for many NN and RNN/RkNN search algorithms, and the existing search algorithms for RNN/RkNN query.

A. R-tree and MBB Distance Metrics

R-tree (including its variants R*-tree [2] and aR-tree [10]) is a data partitioning index that clusters closely located data points and abstracts them as minimum bounding boxes (MBBs), based on which the index is built. Because of tightly bounding enclosed data points, each side of an MBB must touch at least one enclosed data point. Consequently, many useful distance metrics, such as *mindist*, *minmaxdist*, and *maxdist*, have been identified [13]. As shown in Figure 1(a), $mindist(q, N)$ and $maxdist(q, N)$ represent the lower and upper bounds of the distance between any data point inside an MBB N and a single point q; $minmaxdist(q, N)$ defines the upper bound of the distance between a point q and its NN inside an MBB N. In other words, a point q should have at least one point located inside MBB N whose distance does not exceed $minmaxdist(q, N)$.



(a) Between a data point and an MBB (b) Between two MBBs

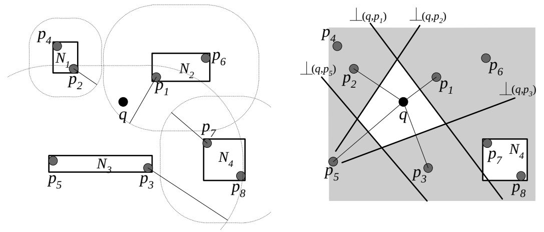
Fig. 1. Distance metrics

Besides, another set of distance metrics are defined between two MBBs. With the same terminologies, *mindist*, *minmaxdist*, and *maxdist* [4] are exemplified in Figure 1(b). $mindist(N_1, N_2)$ and $minmaxdist(N_1, N_2)$ are respectively referred to as the lower and upper bounds of the distance between the closest pair of data points from MBBs N_1 and N_2. $maxdist$ (N_1, N_2) is the upper bound distance of the farthest pair of data points in respective MBBs. In addition, $minExistDNN_{N_2}(N_1)$ [20] represents the minimal upper bound of distance from any point in MBB N_1 to its NN in MBB N_2. As shown in Figure 1(b), an MBB N_1 is partitioned by a perpendicular bisector $\perp_{(a,b)}$, where a and b are diagonal points in N_2, into two portions, P_1 (shaded) and P_2 (not shaded). Conservatively, any data point in P_1 (or P_2) should have its NN not farther than b (or a respectively). Here, $minExistDNN_{N_2}(N_1)$ is $dist(x, a)$, the distance from x to a. $minExistDNN$ is asymmetric that $minExistDNN_{N_2}(N_1)$ and $minExistDNN_{N_1}(N_2)$ are different. All these distance metrics are useful to derive $min\kappa$ in the κ-Browsing algorithm to be discussed later.

B. RNN/RkNN Search Algorithms

Here, we discuss RNN/RkNN search algorithms that can be broadly categorized as *pre-computation based* approaches and *dynamic* approaches.

Pre-computation Based RNN/RkNN Search Algorithms. Pre-computation based approaches pre-execute kNN search for each point p and determine $dist(p, p')$ between p and its kth NN point p' based on a given k. Further, for each object p, a vicinity circle $cir(p, p')$, centered at p with $dist(p, p')$ as the radius, is created. If a query point q is inside $cir(p, p')$, p is the RkNN answer data point. To facilitate the lookup of answer data points, all the vicinity circles are indexed using RNN-tree [8], an R-tree variant specific for vicinity circles. Rather than physically including the vicinity circle, RdNN-tree [22], another R-tree variant, was proposed to keep both data points and their vicinity circle radius. This RdNN-tree can efficiently support both NN and RNN search simultaneously. Figure 2(a) depicts four MBBs of an RdNN-tree containing eight data points $\{p_1, \cdots p_8\}$. Given a query point, q, RNN search locates N_2 and N_3 for potential answer data points as their extended MBBs cover q. Then p_1 and p_5 are retrieved as the answer data points. However, these approaches are limited to support RkNN queries for a fixed k and they incur a very high index construction and update overhead [12]. To support various k, [1], [18] suggested to estimate kNN distance at the run time instead of maintaining actual possible kNN distances.



(a) RdNN-tree (b) TPL algorithm

Fig. 2. RNN algorithms

Dynamic RNN/RkNN Search Algorithms. Dynamic RNN search algorithms perform search based on a general index like R-tree that can be efficiently updated [11], [21]. Stanoi et al., [16] derives Voronoi cells based on R-tree to determine bichromatic RNNs. Other proposed monochromatic RNN/RkNN algorithms adopt a filter-and-refine query processing paradigm [14], [15], [17], in which the search is separated into *filter* phase and *refine* phase. In the filter phase, potential RNN/RkNN answers are identified as candidates from the entire dataset, which may include false hits. In the refine phase, all candidates are evaluated with kNN search and those candidates with more than kNNs found are removed.

Stanoi et al. [15] suggested to partition a 2D search space centered at the query point into six equal-sized sectors. It is proved that those NN objects of q found in each sector are the only candidates of the RNNs. Thus, in the filter step, constrained NN search [5] is conducted to find the NN data point in each sector. The efficiency of Stanoi's algorithm is owing to the small number of candidates, at most six for monochromatic RNN in 2D space. When the dimensionality increases, the number of subspaces for candidates increases exponentially. Singh et. al [14] proposed another algorithm to alleviate the curse of dimensionality. Their algorithm first retrieves KNN data points to the query point as candidates where K (reasonably larger than k of RkNN query) is randomly selected. However, the accuracy and performance of this algorithm is highly dependent on K. The larger K is, the more candidates are identified. Consequently, it is more likely that a complete answer set is returned but with a higher processing cost. A small K favors the efficiency but it may incur many false misses. The filter-and-rank algorithm (discussed in Section I-B) borrows this idea.

To guarantee the completeness of results, Tao et al. [17] proposed TPL algorithm that exploits an half-plane property in space to locate RkNN candidates. The algorithm examines data points based on distance browsing [7]. Every time when an unexplored nearest neighbor data point p to a query point, q, is identified, a *half plane* is constructed along the perpendicular bisector $\perp_{(q,p)}$ between p and q. It is guaranteed that any object p' (or node) falling inside the half plane containing p must have p closer than q to it. Thus, if a data point is covered by k or more half-planes, it should not be an RkNN answer data point, thus can be safely discarded from detail examination. The filter phase terminates when all candidate data points are collected and the others are discarded. As depicted in Figure 2(b), four objects p_1, p_2, p_3, and p_5 are identified as the candidates for R1NN and other data points (e.g., p_4 and p_6) or MBBs (e.g., N_4 that encloses a set of data points) inside the (shadowed) half-planes of candidates are filtered out. Later in the refine step, NN search is performed on these candidates to remove the false hits. The final result set is $\{p_1, p_5\}$.

C. Other RNN Algorithms

Various RNN/RkNN algorithms consider different application scenarios, such as data stream [9], graph network [24], moving objects [3], ad-hoc subspace [23], and object monitoring [19]. Unlike all those reviewed RNN algorithms which identify influenced data set with $\kappa \leq k$, our work in this paper focuses on searching for top t influenced data points ranked with respect to a query point. Besides, the work of influential site ranking [20] is for bichomatic RNN scenarios, aiming at finding a rank list of data points from a set of query points, $\mathcal{Q}$, that influence most of data points in $\mathcal{P}$. In other words, this work intends to find most *influential query points*. Different from this work, our work finds *most influenced data points to a single query point* and ranks them.

III. κ-Counting Algorithm

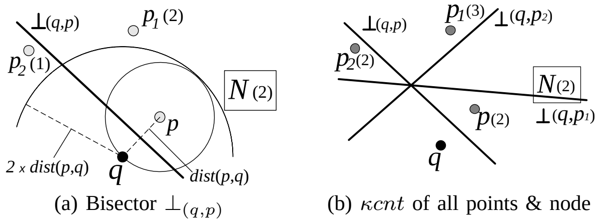
This section details the κ-Counting algorithm. We give an overview of the algorithm followed by the detail of how the algorithm operates for both monochromatic and bichromatic application scenarios. Finally we discuss its strength and weakness.

A. Overview

Based on an intuition that the κ of a data point p is somewhat related to the size of $cir(p, q)$ which centers at p with $dist(p, q)$ as the radius, the κ-Counting algorithm gradually expands the search space starting from a query point, q, outwards to visit the closest data points in $\mathcal{P}$. Additionally, we associate a κcnt, a counter for the number of NNs, with every single data point, initialized to *one*. While the search space is expanded, the κcnt's of some data points are incremented (if some other points are found to be closer to them than q) and/or finalized (if they are not affected any more by later examined data points). Those data points with the smallest finalized κcnt's (that equal to κ's) are collected as answer data points. The algorithm keeps expanding the search space and incrementing the κcnt's of data points until t answer data points are obtained.

We use half-planes as [17] to determine whose κcnt's need updating. When the search space expands to a data point, p, we divide the whole space along the perpendicular bisector, $\perp_{q,p}$ between p and q into two half-planes, denoted by $HP_q(q, p)$ and $HP_p(q, p)$. All the data points fall inside the half-plane containing p, i.e., $HP_p(q, p)$, must have p closer to them than q, so κcnt's of those data points are incremented by one. With the use of half-planes, the κcnt of a data point p equals the number of half-planes that cover p. For notational convenience, we use HP_q in place of $HP_q(q, p)$ hereafter.

To facilitate expanding the search space and counting κcnt's of individual data points or groups of data points, we adopt R-tree index. An example is illustrated in Figure 3(a), where the κcnt's of data points or MBBs are denoted in braces. When a perpendicular bisector $\perp_{q,p}$ is formed between the query point q and its first nearest point p, both data point p_1 and MBB N that represents all enclosed data points, falling inside HP_p, have their κcnt's incremented by one. On the other hand, p_2 is outside HP_p so its κcnt remains one.



(a) Bisector $\perp_{(q,p)}$ (b) κcnt of all points & node

Fig. 3. Basic idea of κ-Counting algorithm

As shown in Figure 3(b), after examining three data points, p, p_1 and p_2, κcnt's of p, p_1, p_2 and N are updated to 2, 3, 2, and 2, respectively. It is noteworthy that some of N's children N' may lie inside HP_{p_3}, though N is not entirely inside it. Consequently, the κcnt's associated with N's may be greater than, but definitely not less than, that of N.

Certainly, after examining all data points/half-planes, κcnt's of all data points can be finalized (converged and equal to κ's). However, since only t most influenced data points (i.e., those with smallest κ's) are needed, a comprehensive checking (that examines all the data points) incurring a large processing overhead is clearly unnecessary. To improve the search efficiency, *early κcnt finalization* is desirable. Following the non-decreasing distance order, κcnt's can be finalized earlier according to the following lemma.

Lemma 1: The κcnt of a data point p is finalized if $dist(p', q)$ of all unexamined data points p' to the query point q is greater than $2 \times dist(p, q)$. □

Proof: Since the perpendicular bisector, i.e., the boundary of a half-plane, formed between a query point q and any point p' must be at least $dist(p', q)/2$ away from q, the half-plane cannot cover any data point, p, whose $dist(p, q) < dist(p', q)/2$ (see Figure 3(a)). Thus, p's κcnt can be finalized and equals κ. ■

Though the κcnt's of some data points can be finalized earlier, it is not guaranteed that those points with early finalized κcnt must be the RRNN query answers. Until their κ's are certain to be the smallest, they will not be output as a part of the RRNN query result. In the following subsections, we present κ-Counting algorithm for monochromatic and bichromatic RRNN application scenarios.

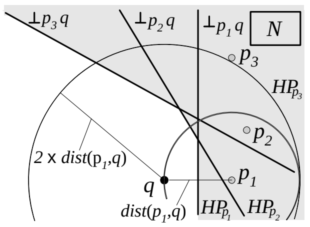
B. κ-Counting Algorithm for Monochromatic RRNN

The κ-Counting algorithm for monochromatic RRNN is based on distance browsing [7] as described in the pseudo-code in Figure 4. In this algorithm, data points have to be examined through three stages, namely, *queued* (pending for examination), *examined* (examined but with non-finalized κ's), and *finalized* (examined with finalized κ's), before they can be included as RRNN query results. A priority queue (P), a candidate set (C), and a finalized candidate set (F) are used to maintain data points in these respective stages. In addition, a half-plane set (H) maintains all the half-planes of examined data points. We also adopt a histogram to facilitate the decision on whether the finalized κcnt of a data point p is the smallest. For each value of κcnt, we record the number of data points/index nodes $p \in P \cup C$ with $\kappa cnt_p = \kappa cnt$. When a data point or an index node changes its κcnt from κ_{old} to κ_{new}, the number associated with value κ_{old} is reduced by one while that with value κ_{new} is increased by one. When the numbers for all κcnt's in the histogram smaller than p's κcnt reach zeroes, p's κcnt is guaranteed to be the smallest. Moreover, the histogram is very update efficient.

The algorithm starts with P filled with the root of the index and C, F and H set to empty. Thereafter, it iteratively takes out the head entry of P, that is, the closest unexamined entry ϵ (either a data point or an index node) to the query point. In each round, ϵ is checked (lines 2-23). If ϵ is a data point, a half-plane HP_ϵ is created and preserved in H (line 14). Next, all pending data points and index nodes in P and all data points in C falling inside this half-plane increase their κcnt by one (lines 15-18). Finally, ϵ is kept in C as a candidate (line 19). Otherwise, ϵ must be an index node. It is explored and all its children c are placed back to P. The κcnt of each newly inserted entry c is counted by comparing c against all half-planes in H (lines 8-12). Besides, the *mindist* of ϵ is compared against the distances between q and all data points in C. The data points in C with their κcnt's finalized according to Lemma 1 are moved to F (lines 4-6). Further, those data points in F with smallest κcnt are output as the

```
Algorithm κ-Counting($q, root, t$)
Input:   a query point (q), the root index of $\mathcal{P}$ ($root$),
         the number of answer data points ($t$)
Local:   a priority queue ($P$), a candidate set ($C$),
         a finalized candidate set ($F$) and a half-plane set ($H$);
Output:  $t$ RRNN result points;
Begin
 1.  enqueue ($root, 1$) to $P$ /* where 1 is the initial $\kappa cnt$ */
 2.  while ($P$ is not empty AND $t > 0$) do
 3.    ($\epsilon, \kappa cnt$) ← dequeue($P$);
/* identify those data point with finalized $\kappa cnt$ */
 4.    foreach $(p, \kappa cnt) \in C$
 5.      if ($dist(p, q) \leq mindist(q, \epsilon)/2$) then
 6.        $C \leftarrow C - \{(p, \kappa cnt)\}$; $F \leftarrow F \cup \{(p, \kappa cnt)\}$;
/* explore the entry */
 7.    if ($\epsilon$ is an index node) then
 8.      foreach $c \in \epsilon$'s children /* explore index node */
 9.        $\kappa cnt \leftarrow 1$;
10.        foreach $h \in H$
11.          if ($c$ inside $h$) then $\kappa cnt \leftarrow \kappa cnt + 1$
12.        enqueue ($c, \kappa cnt$) to $P$;
13.    else /* $\epsilon$ is a point */
14.      $H \leftarrow H \cup \{HP_\epsilon(\epsilon, q)\}$;
15.      foreach ($p, \kappa cnt_p$) in $P$ /* update $\kappa cnt$'s of others */
16.        if ($p$ inside $HP_\epsilon(\epsilon, q)$) then $\kappa cnt_p \leftarrow \kappa cnt_p + 1$;
17.      foreach ($p, \kappa cnt_p$) in $C$
18.        if ($p$ inside $HP_\epsilon(\epsilon, q)$) then $\kappa cnt_p \leftarrow \kappa cnt_p + 1$;
19.      $C \leftarrow C \cup \{(\epsilon, \kappa cnt)\}$;
/* output data point(s) with smallest finalized $\kappa$ */
20.    determine $m = MIN(\kappa cnt_p)$ with $p \in P \cup C$;
21.    foreach $(p, \kappa cnt) \in F$
22.      if ($\kappa cnt \leq m$) then
23.        $F \leftarrow F - (p, \kappa cnt)$; output ($p, \kappa cnt$);$t \leftarrow t - 1$;
End.
```

Fig. 4. κ-Counting Algorithm for Monochromatic RRNN



Exam	κcnt			
	p_1	p_2	p_3	N
p_1	1	2	2	2
p_2	2	2	3	3
p_3	2	3	3	4

Fig. 5. Example of κ-Counting for Monochromatic RRNN

partial RRNN query result *immediately* (lines 20-23). As long as t answer data points are collected, the algorithm terminates.

Figure 5 shows an example run of the κ-Counting algorithm. Suppose an RRNN query (with $t = 1$) issued at q searches for *one* data point with the smallest κ. The queue P currently contains three points, p_1, p_2, and p_3, and one index node, N, after some steps of index traversal, and the sets H, F and C are empty. First, p_1, the head of P and having $\kappa cnt = 1$, is examined. A half-plane HP_{p_1}, formed based on $\perp_{(q,p_1)}$, is inserted into H. Since p_2, p_3 and N in P fall inside HP_{p_1}, their corresponding κcnt's are increased by 1. p_1 is thereafter moved to C. Next, p_2 is examined and its half-plane HP_{p_2} covers p_1, p_3 and N. Thus, the κcnt of p_1, p_3 and N are changed to 2, 3 and 3, respectively. p_2 and HP_{p_2} are inserted into sets C and H accordingly to complete the second round.

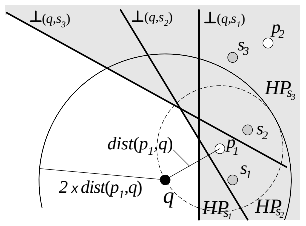
When p_3 is inspected, the κcnt of p_1 is finalized (based on Lemma 1) since $mindist(q, p_3)$ is twice more than $dist(p_1, q)$, and hence it is moved from C to F. As p_3's half-plane, HP_{p_3}, covers p_2 and N, κcnt associated with p_2 is increased to 3 and that with N is incremented to 4. As p_1's finalized κ is smaller than that of the rest data points (i.e., p_2, p_3 and N), it is output as the RRNN query result to complete the search.

C. κ-Counting Algorithm for Bichromatic RRNN

The κ-Counting algorithm for bichromatic RRNN query considers two datasets $\mathcal{P}$ and $\mathcal{Q}$. The answer data points are retrieved from $\mathcal{P}$ while their NNs are obtained from $\mathcal{Q}$. The logic is pretty much the same as that for monochromatic RRNN query. We associate κcnt's with all data points and index nodes from $\mathcal{P}$. Data points in $\mathcal{P}$ have to go through three stages, as described in Section III-B. Thus, a priority queue (P), a candidate set (C), and a finalized candidate set (F) are maintained. Examined data points in $\mathcal{Q}$ form half-planes, which are stored in H. As the examination follows the distance order, we put data points and index nodes from $\mathcal{Q}$ and $\mathcal{P}$ into P to provide a global distance order. Every time when an entry dequeued from P is being examined, one of the following operations is performed accordingly.

- **Case 1**. If the entry is an index node from $\mathcal{Q}$, it is explored and all its children nodes are pushed back to P for later examination.
- **Case 2**. If the entry is a data point s from $\mathcal{Q}$, it forms a half-plane HP_s based on perpendicular bisector $\perp_{(q,s)}$. Those entries in C and P (data points/index nodes of $\mathcal{P}$) falling inside HP_s increase their κcnt's by 1. The newly formed half-plane HP_s is then maintained in H.
- **Case 3**. If the entry is an index node from $\mathcal{P}$, it is explored. All its children are checked against all half-planes in H, update their κcnt's, and are enqueued to $\mathcal{P}$.
- **Case 4**. If the entry is a data point p from $\mathcal{P}$, it is put into C.

As previously discussed, the κcnt of a candidate data point p ($\in C$) can be finalized when the $mindist$ of the current head entry (and hence of all the other queued entries) to q is greater than the double of $dist(p, q)$ according to Lemma 1. Next, p with finalized κcnt is moved from C to F. Further, when its finalized κcnt is smaller than all others in P and C, p is removed from F and delivered as one of the query results. To efficiently determine whether the finalized κcnt's of some data points in F are the smallest, we maintain a histogram of κcnt's of data points in C and P. The algorithm terminates when t RRNN answer data points are collected.



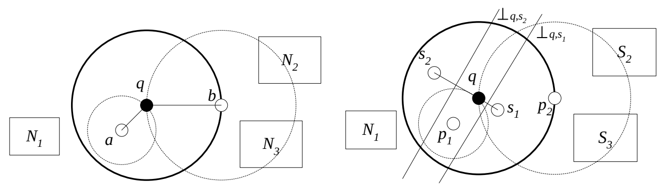
Exam	κcnt	
	p_1	p_2
s_1	2	2
s_2	3	3
s_3	3	4

Fig. 6. Example of κ-Counting for Bichromatic RRNN

As it is similar to that for monochromatic RRNN query, the pseudo-code of the κ-Counting algorithm for bichromatic RRNN query is omitted to save space. Figure 6 provides an illustrative example where t, the number of required data points, is set to 1. Assume that after certain traversal steps, P contains $[s_1, p_1, s_2, s_3, N_{\mathcal{S}}, p_2]$, with s_1 being the head and C, F and H being empty. Firstly, s_1 is examined, a half-plane, HP_{s_1}, is created with respect to s_1 and q, and the κcnt's of both p_1 and p_2 are incremented to 2. Secondly, p_1 is examined and buffered in C. Thirdly, s_2 is examined and its half-plane HP_{s_2} covers p_1 and p_2. As a result, the κcnt's of both p_1 and p_2 become 3. Then, s_3 is dequeued. Its *mindist* is twice greater than $dist(p_1, q)$, so p_1's κcnt is finalized and it is moved from C to F. Besides, HP_{s_3} covers p_2, and p_2's κcnt is therefore incremented to 4. At last, p_1 in F with the smallest κcnt is confirmed to be the final result. It is delivered and the search ends.

D. Discussion

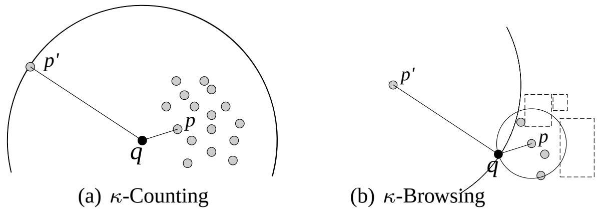
It is pretty straightforward to adapt the κ-Counting algorithm to support RkNN query by collecting data points with their finalized κcnt's (i.e., κ) not exceeding k and terminating the search when all the remaining data points (i.e., those in the priority queue and the candidate set) are confirmed to have κcnt's greater than k. By doing so, the κ-Counting algorithm can provide progressive result delivery that none of existing RkNN algorithm can offer. Besides, the κ-Counting algorithm can operate without specifying t, defaulting $t = \infty$. This actually sorts all the objects according to ascending order of κ, i.e., the degree of influence received from q.



(a) Monochromatic RRNN (b) Bichromatic RRNN

Fig. 7. Scenario about Filter-and-Rank

The κ-Counting algorithm can outperform those previously discussed solutions, namely, κ-Probing and FR. It does not repeatedly access the same data set as the κ-Probing; it does not need to access extra data points as FR; and it can guarantee the result correctness. Figure 7(a) shows a scenario, where a monochromatic RRNN query is issued at q and t is set to 1. FR examines 2NN points as its candidates. Here, a whose κ is 1 is the RRNN, but not b. In the rank phase, index nodes, N_2 and N_3, are visited as they intersect $cir(b, q)$, thereby incurring extra I/O costs. For bichromatic scenario, the κ-Counting algorithm can also perform reasonably better than FR. Figure 7(b) shows a scenario where an RRNN query is issued at q and p_1 is the answer point. FR retrieves both p_1 and p_2 as initial candidates. Based on their circles, other points like s_1 and index nodes S_2 and S_3 from $\mathcal{Q}$ are accessed. However, for the same situation, the κ-Counting algorithm does not need to explore that many index nodes of $\mathcal{Q}$ and even does not need to examine p_2. It accesses s_1, s_2, p_1 and then p_2 according to the global distance order. When p_2 is accessed, p_1's κcnt is finalized. At this time, no other data points have smaller κcnt than p_1's, thus the κ-Counting algorithm terminates earlier.



(a) κ-Counting (b) κ-Browsing

Fig. 8. Skewed Dataset

However, the κ-Counting algorithm, based on an intuition that the κ of a data point is related to its distance to a query point, could be less efficient for skewed datasets. As illustrated in Figure 8(a), p' is the answer data point but it is far away from q. According to the $mindist$ metric, the κ-Counting algorithm scans data points to form half-planes that are used to update the κcnt's of covered points. As a result, it has to scan all the data points on the right side of q before visiting p'. From this, we can see that processing RRNN query by means of distance ordering is not necessarily a good strategy. In the next section, we present κ-Browsing, our second algorithm using $min\kappa$'s to order the access of candidate data points/index nodes for RRNN query processing.

IV. κ-Browsing Algorithm

In this section, we detail the *κ-Browsing* algorithm which is based on a notion of $min\kappa$, an estimation of κ for a data point. To facilitate the calculation of $min\kappa$'s, we adopt aRtree [10] which is widely used to support aggregation query. For RRNN and RkNN, determining κ's that counts the number of NNs with respect to a data point is a sort of aggregation. aRtree is an Rtree variant with every index node associated with a *count* indicating the number of data points indexed beneath the node. Specifically, the count associated with a leaf node records the number of enclosed data points and the count associated with a non-leaf node equals the sum of counts of all its child (descendent) nodes. In the following, we discuss the basic idea of the κ-Browsing algorithm and then introduce the notion of $min\kappa$ and its properties, followed by description of the κ-Browsing algorithm for monochromatic and bichromatic RRNN scenarios.

A. Overview

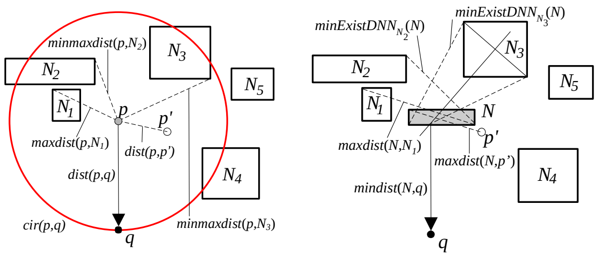
The key idea of the κ-Browsing algorithm is to order the access of data points/index points based on their likelihoods of being/containing the answer data points. To illustrate the idea of the algorithm, let us consider Figure 8(b) (which depicts the same scenario as Figure 8(a)). Considering two unexplored data points p and p'. It is reasonable to examine the data point p' before p, as we can visualize from the figure that $\kappa_{p'}$ is smaller than κ_p. However, the exact κ_p and $\kappa_{p'}$ is unknown without exploring other data points/index nodes around p and p'. Thus, a challenging issue that the κ-Browsing algorithm faces is how to determine the access order between p and p' (and other data points and index nodes) without exactly knowing their κ's. To tackle this problem, we introduce a notion of $min\kappa$, associated with every data point and index node, to represent the minimal number of data points being closer to its associated data point or index node (i.e., all the data points inside the MBB of the index node) than q, estimated based on available but limited knowledge of the data point distribution. In other words, it is the lower bound of κ of a data point or all data points inside an index node. A data point/index node with a relatively large $min\kappa$ is obviously less likely to be/contain the most influenced data point(s), and thus the access priority should be given to those with smaller $min\kappa$'s. With $min\kappa$'s, the κ-Browsing algorithm can also efficiently prune the search space. As illustrated in Figure 8(b), we can figure out, based on knowledge discovered during the query processing, that p's $min\kappa$ is larger than that of p'. Then, the search can decide to process p' before p and all its neighboring data points and index nodes, thus alleviating the processing overhead. Further, as $\kappa_{p'}$ is smaller than the $min\kappa$'s of p and its surrounding data points, p' can immediately be output and the search terminates.

B. Notion of $min\kappa$

The estimation of $min\kappa$'s is proceeded along with the index traversal. The state of the index under examination can be represented by a set of index nodes and data points (denoted by V). The data points and index nodes in V, logically constituting the whole dataset are not nested. The initial state of V contains only the root node of the index. As the κ-Browsing algorithm traverses and expands index nodes, V evolves into new sets of data points and index nodes that provide more precise knowledge of data distribution in the space.

Given a data point p, its $min\kappa$ indicates the minimum possible number of data points closer to p itself than a query point q, based on the knowledge embedded in the current state of V. Figure 9(a) shows an illustrative example of how $min\kappa$ of p is estimated. Rooted at p, a circle $cir(p, q)$ that covers some MBBs and data points is drawn. First, a data point p', inside the circle, is guaranteed to be closer to p than q and hence is counted towards $min\kappa$ of p. Similarly, N_1

is fully covered by the circle (i.e., $maxdist(p, N_1) \leq dist(p, q)$), that means all the data points enclosed by N_1 are definitely closer to p than q. We count $N_1.cnt$ (that denotes the *count* associated with N_1) towards $min\kappa$. Conversely, N_2, N_3 and N_4 are partially covered. With *minmaxdist*, we can assure each of N_2 and N_3 can contribute at least 1 towards $min\kappa$ of p because each of them has at least one side completely inside $cir(p, q)$. Finally, only a corner (rather than a complete side) of N_4 is covered and conservatively no contribution from N_4 to $min\kappa$ is assumed. As a result, the $min\kappa$ is estimated as $(1 + 3 + N_1.cnt)$. Notice that we need to add 1 for q to the $min\kappa$.



(a) $min\kappa(q, p, \mathcal{V})$ (b) $min\kappa(q, N, \mathcal{V})$

Fig. 9. Examples of $min\kappa$

Hence, based on a given V (i.e., the current state of explored data space in terms of index nodes and data points), Equation (1) calculates the $min\kappa$ of a data point p and Lemma 2 states the condition where the $min\kappa$ can be finalized and converged to κ.

$$min\kappa(q, p, V) = 1 + \sum_{v \in V} count(p, v) \tag{1}$$

where $count(p, v) =$

$$\begin{cases} 1 & \text{if } v \text{ is a data point } \wedge dist(p, v) \leq dist(p, q); \\ v.cnt & \text{if } v \text{ is an index node } \wedge maxdist(p, v) \leq dist(p, q); \\ 1 & \text{if } v \text{ is an index node } \wedge \\ & \quad minmaxdist(p, v) \leq dist(p, q) < maxdist(p, v); \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 2: Given a data point p, a query point q, and a set of index nodes and data points maintained in V, $min\kappa(q, p, V)$ equals κ_p if all touched index node N ($\in V$) are fully covered by $cir(p, q)$, i.e., $\forall_{N \in V | mindist(p,N) \leq dist(p,q)}\ maxdist(p, N) \leq dist(p, q)$. □

Proof. As no index node is partially covered by $cir(p, q)$, data points and index nodes enclosed completely contribute their counts to the $min\kappa(q, p, V)$. Thus the $min\kappa$ of p is finalized and equal to κ_p. ■

Given an index node N, the $min\kappa$ of N indicates the minimum possible number of data points that must be closer to all the data points inside N than q wherever they are located inside N. Compared with that for a single data point, the calculation of $min\kappa$ for an index node is more complicated. Since the exact positions of data points inside an index node are unknown but they are certainly bounded by MBBs, we estimate $min\kappa$'s based on heuristics derived from MBB distance metrics as discussed in Section II. There are three possible cases that all the data points inside MBB can find another point closer to them than q as stated in the following lemmas.

Lemma 3: A data point p' is not farther to all data points inside an index node N than q if $maxdist(N, p') \leq mindist(N, q)$. □

Proof. Let p be a data point anywhere inside N. Since $dist(p, p') \leq maxdist(N, p')$ and $mindist(N, q) \leq dist(p, q)$, $maxdist(N, p') \leq mindist(N, q)$ guarantees $dist(p, p') \leq dist(p, q)$. ■

Lemma 4: An entire index node N' (i.e, all data points inside N') is not farther to another index node N (i.e., any data point inside N) than q if $maxdist(N, N') \leq mindist(N, q)$. □

Proof. Assume that p is a data point located inside N. Due to the fact that $dist(p, N') \leq maxdist(N, N')$ and $mindist(N, q) \leq dist(p, q)$, $maxdist(N, N') \leq mindist(N, q)$ ensures $dist\ (p, N') \leq dist(p, q)$. ■

Lemma 5: At least one data point in an index node N' is not farther to all data points inside an index node N than q if $minExistDNN_{N'}(N) \leq mindist(N, q)$. □

Proof. Consider that a data point p is inside N, its NN point p' is in N' and their distance is $dist(p, p')$. Since $dist(p, p') \leq minExistDNN_{N'}(N)$, $dist(p, p') \leq dist(p, q)$ is ensured, according to the stated condition: $minExistDNN_{N'}(N) \leq mindist(N, q)$ and $mindist(N, q) \leq dist(p, q)$. ■

Based on Lemma 3, 4 and 5, Equation (2) can be obtained to determine the $min\kappa$ of an index node N (given a query point q and a set of data points/index nodes V).

$$min\kappa(q, N, V) = 1 + \sum_{v \in V} count(N, v) \tag{2}$$

where $count(N, v) =$

$$\begin{cases} 1 & \text{if } v \text{ is a data point } \wedge maxdist(N, v) \leq mindist(N, q); \\ v.cnt & \text{if } v \text{ is an index node } \wedge maxdist(N, v) \leq mindist(N, q); \\ 1 & \text{if } v \text{ is an index node } \wedge \\ & minExistDNN_v(N) \leq mindist(N, q) < maxdist(N, v); \\ 0 & \text{otherwise.} \end{cases}$$

Figure 9(b) depicts the $min\kappa$ of an index node N. In the figure, the data point p' is closer to the entire N than q and hence contributes 1 to N's $min\kappa$. N_1, thanks for the smaller $maxdist(N, N_1)$ ($< mindist\ (N, q)$), contributes $N_1.cnt$ to the $min\kappa$. Besides, both N_2 and N_3 certainly have at least one point each closer to any point inside N than q, since $minExistDNN_{N_2}(N)$ and $minExistDNN_{N_3}(N)$ are smaller than $mindist(N, q)$. In brief, the corresponding $min\kappa$ is $1 + 3 + N_1.cnt$. Further, we explore monotone properties of $min\kappa$ (defined in Lemma 6 and Lemma 7) that is useful to the κ-Browsing algorithm.

Lemma 6: Given a set of data points/index nodes in V, and a query point q, if N_c is a child (or descendent) of N, $min\kappa(q, N_c, V) \geq min\kappa(q, N, V)$. □

Proof. As N_c is a child (descendent) of N, $mindist(N_c, q) \geq mindist(N, q)$. Also, other upper distance metrics (i.e., *maxdist* and *minExistDNN*) of N_c to another data point/index node could be smaller (definitely not greater) than that of N. By Equation (1) and (2), N_c will cover either a same set of data points/index nodes as N does, or more data points/index nodes or larger portions of index nodes than N. Hence, same or greater $min\kappa$ is expected. ■

Lemma 7: Given a query point, q, a set of data points/index nodes in V, an index node N' in V is explored and replaced with its n descendants $N'_1, \cdots N'_n$, resulting in V'. For any data point p or index nodes N, the following two statements should be true:

1) $min\kappa(q, N, V') \geq min\kappa(q, N, V)$, and
2) $min\kappa(q, p, V') \geq min\kappa(q, p, V)$ □

Proof. Without losing generality, we assume $V' = V - \{N'\} \cup \{\cup_{i=1}^{n} N'_i\}$. Let δ denote the difference between $min\kappa(q, N, V')$ and $min\kappa(q, N, V)$, i.e., $\delta = \sum_{i=1}^{n} count(N, N'_i) - count(N, N')$. There are four possible conditions that N may contribute to $min\kappa(q, N, V)$:

1) $maxdist(N, N') \leq mindist(N, q)$. The $count(N, N')$ is $N.cnt$. Since all $N'_i \subset N'$, all $maxdist(N, N'_i)$ must not be greater than $maxdist(N, N')$ and the total counts of all children must be equal to $N'.cnt$. Hence, $\sum_{i=1}^{n} count(N, N'_i) = count(N, N')$ and $\delta = 0$;
2) $minExistDNN_{N'}(N) \leq mindist(N, q) < maxdist\ (N, N')$. The $count(N, N')$ is 1. Since there is no N'_i such

that $minExistDNN_{N'_i}(N) > minExistDNN_{N'}(N)$. By Equation (2), N_i can provide at least 1 to $min\kappa$ as N does. Hence, $\delta \geq 0$.

3) $mindist(N, q) < minExistDNN_{N'}(N)$. $count(N, N')$ is 0. Because there would be N'_i whose $minExistDNN_{N'_i}(N)$ is not greater than $minExistDNN_{N'}(N)$ and may be smaller than $mindist(N, q)$, so at least 1 is counted. Even more, some N_i may have $maxdist(N, N'_i) < mindist(N, q)$, resulting in the contribution of $N'_i.cnt$ to $min\kappa$. Hence, $\delta \geq 0$.
4) $mindist(N, q) < mindist(N, N')$. In this case, N' and all its children would not contribute to the $min\kappa$ of N, thus $\delta = 0$.

As a result, the difference δ is guaranteed to be nonnegative for all possible conditions and hence $min\kappa(q, N, \mathcal{V}') \geq min\kappa(q, N, \mathcal{V})$. A similar proof can be conducted for a data point p. To save space, the proof is omitted. ■

C. κ-Browsing Algorithm for Monochromatic RRNN

The κ-Browsing algorithm for monochromatic RRNN considers only one dataset, $\mathcal{P}$. In the algorithm, a priority queue P is adopted to keep unexamined data points/index nodes according to the non-decreasing order of their $min\kappa$'s. In the case of a tie that two or more data points/index nodes have same $min\kappa$'s, the one with the smallest *mindist* is ordered first. Besides, a set V is maintained to capture the current knowledge of the data point ($\in \mathcal{P}$) distribution via a set of data points/index nodes, based on which the $min\kappa$ of each entry ($\in P$) is estimated.

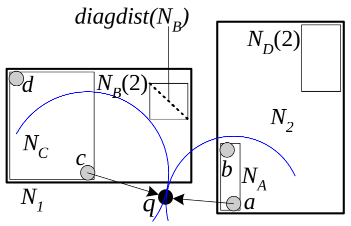
The algorithm always dequeues the head entry for examination until t data points with the smallest κ's are retrieved. For monochromatic RRNN, all the data points/index nodes in V (except p itself) contribute to the $min\kappa$ of point p. As a result, the $min\kappa$ of a data point p based on the set V is represented by $min\kappa(q, p, V - \{p\})$. The $min\kappa$ of an index node N based on the set V is represented by $min\kappa(q, N, V - \{N\}) + sc(q, N)$ where $sc(q, N)$ counts the number of other data points p' inside N that are closer to a point p than q, with $p', p \in N$ and $p \neq p'$. More specifically,

$$sc(q, N) = \begin{cases} N.cnt - 1 & \text{if } diagdist(N) \leq mindist(N, q) \\ 0 & \text{otherwise.} \end{cases}$$

where $diagdist(N)$ is the diagonal distance of N.

Let us see how the κ-Browsing algorithm runs as exemplified in Figure 10. Suppose an RRNN (with $t = 1$) is issued at a query point q. Initially, the root of P, $root$, is enqueued into a priority queue P with its associated $min\kappa$ set to 1. A set V also takes $\{root\}$ as the starting content. The algorithm begins. It retrieves $root$ from P and explores its children N_1 and N_2. Then, set V is updated to $\{N_1, N_2\}$ accordingly. The explored N_1 and N_2 with associated $min\kappa$'s (=1) are enqueued to P. Due to its smaller *mindist*, N_1 is dequeued and its children N_B and N_C are retrieved. Consequently, V is updated to $\{N_2, N_B, N_C\}$. Again, N_B with updated $min\kappa = 2$ and N_C with unchanged $min\kappa = 1$ are inserted back to P. N_2 becomes the head as it has the smallest $min\kappa$ and shortest $mindist$ to q. N_2 is dequeued, followed by N_C and then point c, with V and corresponding $min\kappa$ updated accordingly as shown in Figure 10. When c is dequeued, its $min\kappa$ equals 1 which is the smallest among all the queued entries. Because there is no other data point inside $cir(c, q)$, the $min\kappa$ of c is finalized (i.e., actually κ_c), according to Lemma 2. Furthermore, based on Lemma 6 and Lemma 7, the $min\kappa$'s associated with other entries might increase, but certainly not decrease. Therefore, c is safely reported as the RRNN query result to complete the search.

The $min\kappa$'s of entries in P might be changed whenever V is updated, resulting in a high $min\kappa$ update (processing) cost, especially



Eax.	$P_{\mathcal{P}}$	$V_{\mathcal{Q}}$
	$[(N_1, 1), (N_2, 1)]$	$\{N_1, N_2\}$
N_1	$[(N_2, 1), (N_C, 1), (N_B, \mathbf{2})]$	$\{N_2, N_B, N_C\}$
N_2	$[(N_C, 1), (N_A, 2), (N_B, 2), (N_D, 2)]$	$\{N_A, N_B, N_C, N_D\}$
N_C	$[(c, 1), (N_A, 2), (N_B, 2), (N_D, 2), (d, \mathbf{4})]$	$\{c, d, N_A, N_B, N_D\}$
c	Take c as RRNN query result.	

Fig. 10. Example of κ-Browsing for Monochromatic RRNN

when the priority queue is long, the number of entries maintained in the view is large, and/or view update rate is high. In order to reduce the update cost for all queued entries and to maintain the efficiency of the κ-Browsing algorithm, we propose an on-demand $min\kappa$ update scheme. This scheme is motivated by an observation that many entries in the queue will not be examined in detail. It tries to defer the $min\kappa$ update of the queued entry until it is needed. In support of the κ-Browsing algorithm, the head entry of the priority queue must have the smallest $min\kappa$. Therefore, when an entry is dequeued, its $min\kappa$ is updated based on the current V content and compared against that of the second entry whose $min\kappa$ is the smallest among the rest entries in the queue. According to Lemma 6 and 7, the updated V does not reduce $min\kappa$'s of the queued entries. As a result, the first entry with updated $min\kappa$ smaller than the second entry is guaranteed to have the smallest $min\kappa$ and hence it can be dispatched safely. Otherwise, the head entry with the new $min\kappa$ is pushed back to the queue, and the new head is examined. This may iterate until a head entry with the smallest updated $min\kappa$ is found. The function *DequeueWithUpdate* that makes use of basic queue operations is defined in Figure 11.

```
Function DequeueWithUpdate(P, q, V)
Input:   a priority queue (P), a query point (q),
         a collection of index nodes and data points (V);
Output:  a queue entry in form of (ε, k) with the smallest k;
Begin
1.  while(true)
2.    $(\epsilon, k) \leftarrow dequeue(P)$;
3.    if ($\epsilon$ is a data point) then $k \leftarrow min\kappa(q, \epsilon, V - \{\epsilon\})$ ;
4.    else $k \leftarrow min\kappa(q, \epsilon, V - \{\epsilon\}) + sc(q, N)$;
5.    $(\epsilon', k') \leftarrow head(P)$; /* get the head entry of $P$ */
6.    if ($k \leq k'$) return $(\epsilon, k)$;
7.    enqueue $(\epsilon, k)$ to $P$;
End.
```

Fig. 11. Function *DequeueWithUpdate*

Beside the update of $min\kappa$'s, index traversal is another important issue. When a data point p is explored, its $min\kappa$ cannot be finalized unless all touched MBBs are fully covered by $cir(p, q)$. In this case, partially covered MBBs need to be explored. Exploring *all* of those partially covered MBBs at the time p is explored is certainly not a good strategy, especially if p is not the answer data point. Instead, we select one of those partially covered MBBs to explore at a time. If p is an answer data point, all those MBBs are eventually explored any way. On the other hand, if p is not the answer, exploring all the partially covered nodes only causes extra I/O costs. Here, our selection strategy explores the index node with the largest overlap with $cir(p, q)$. This index node has a higher potential to contribute more data points to $min\kappa$, thus narrowing the difference between $min\kappa$ and actual κ. After a node is explored, V is updated and p is re-inserted into P with updated $min\kappa$ for next examination.

We depict the pseudo code of the κ-Browsing algorithm for the monochromatic RRNN in Figure 12. It takes a query point, q, the root node, $root$, of the aR-tree of a dataset $\mathcal{P}$, and the number of requested answer data points, t, as the inputs. It first initializes V with $root$ and the priority queue P with $root$ associated with initial $min\kappa$ (lines 1-2). Then, it explores the head entry which has the smallest $min\kappa$ until t answer data points are reported (lines 3-20). If the entry is an index node, the corresponding entry in V is first replaced by all its children nodes and then the children are inserted into P (lines 6-9). Otherwise, a point p is explored. Among all MBBs are partially covered by $cir(p,q)$ if any, the one with the largest overlap area is selected to explore (lines 12-17). Or, the $min\kappa$ is finalized to κ and guaranteed to be the smallest. Finally, the data point is output as one answer point (line 19).

```
Algorithm $\kappa$-Browsing($q$,$root$,$p$)
Input:   a query point ($q$), the root of $\mathcal{P}$ ($root$)
         the no. of requested result points ($p$)
Local:   a priority queue ($P$), a set of index nodes ($V$)
Output:  $t$ RRNN answer data points;
Begin
 1.  $\mathcal{V} \leftarrow \{root\}$;
 2.  enqueue $(root, min\kappa(q, root, V))$ to $P$;
 3.  while ($P$ is not empty AND $t > 0$) do
 4.    $(\epsilon, min\kappa) \leftarrow DequeueWithUpdate(P, q, V)$;
 5.    if ($\epsilon$ is an index node) then
 6.      suppose $\epsilon$ has $n$ children $N_i$ ($i \in [1, n]$);
 7.      $V \leftarrow V - \{\epsilon\} \cup (\cup_{i=1}^{n}\{N_i\})$;
 8.      foreach $N_i$ ($1 \leq i \leq n$) do
 9.        enqueue $(N_i, min\kappa(q, N_i, V))$ to $P$;
10.    else                                  /* $\epsilon$ is a point */
11.      if $\exists N$ partially covered by $cir(\epsilon, q)$, then
12.        suppose $N$ has $n$ children $N_i$ ($i \in [1, n]$);
13.        $V \leftarrow V - \{N\} \cup (\cup_{i=1}^{n}\{N_i\})$;
14.        remove $N$ from $P$;
15.        foreach $N_i \subset N$ ($1 \leq i \leq n$) do
16.          enqueue $(N_i, min\kappa(q, N_i, V))$ to $P$;
17.        enqueue $(\epsilon, min\kappa(q, \epsilon, V))$ to $P$;
18.      else      /* there is no partially covered node */
19.        output $(\epsilon, min\kappa)$;
20.        $t \leftarrow t - 1$;
End.
```

Fig. 12. κ-Browsing for Monochromatic RRNN

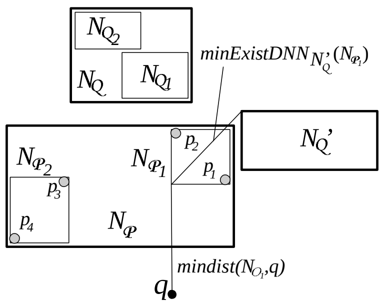
D. κ-Browsing Algorithm for Bichromatic RRNN

The logic of the κ-Browsing algorithm for bichromatic RRNN scenario is similar to that for monochromatic RRNN scenario. We omitted the pseudo-code for this algorithm to save space. In a high-level description, it uses a priority queue $P_{\mathcal{P}}$ to keep track of data points/index nodes in the non-decreasing order of $min\kappa$'s and a set $V_{\mathcal{Q}}$ to preserve the knowledge regarding another set of data points, $\mathcal{Q}$, to support determination of $min\kappa$.

Though two datasets are involved and they interact with each other in the estimation of $min\kappa$, the algorithm still examines the head entry ϵ of the priority queue $P_{\mathcal{P}}$ in every round. An expansion of $V_{\mathcal{Q}}$ is triggered upon examination of ϵ. If ϵ is a data point, index nodes maintained in $V_{\mathcal{Q}}$ that are partially covered by $cir(\epsilon, q)$ need to be explored. As explained previously, instead of exploring all of them at a time, our node exploring strategy explores the one with largest overlap with $cir(\epsilon, q)$. Thereafter, $V_{\mathcal{Q}}$ is updated and the entry ϵ is put back to $P_{\mathcal{P}}$ for later examination. If there is no index node in $V_{\mathcal{Q}}$ that is partially covered by the vicinity circle, the $min\kappa$ of ϵ actually equals κ. Consequently, ϵ is returned as one of the final results since it has the smallest finalized $min\kappa$ value.

On the other hand, if ϵ is an index node $N_{\mathcal{P}}$, some index nodes $N_{\mathcal{Q}}$ in $V_{\mathcal{Q}}$ may contribute to its $min\kappa$. A question raised is which index node ($N_{\mathcal{P}}$ or $N_{\mathcal{Q}}$) is good to explore next. Recall that in Equation (2), if $maxdist(N_{\mathcal{P}}, N'_{\mathcal{Q}}) \leq mindist(N_{\mathcal{P}}, q)$, node $N'_{\mathcal{Q}}$ is contributing all its $N'_{\mathcal{Q}}.cnt$ points to $min\kappa$. Otherwise, $N'_{\mathcal{Q}}$ is considered to contribute at most 1 to $min\kappa$, which causes $min\kappa$ underestimated a lot even if a large portion of $N_{\mathcal{Q}}$ is closer to $N_{\mathcal{P}}$ than q. To decide which node ($N_{\mathcal{Q}}$ or $N_{\mathcal{P}}$) to explore, we compare $minExistDNN_{N_{\mathcal{Q}}}(N_{\mathcal{P}})$ and $mindist(N_{\mathcal{P}}, q)$. If $minExistDNN_{N_{\mathcal{Q}}}(N_{\mathcal{P}})$ is greater than $mindist(N_{\mathcal{P}}, q)$, $N_{\mathcal{P}}$ is explored. Otherwise, $N_{\mathcal{Q}}$ is explored since $N_{\mathcal{Q}}$ would have a few data points closer to $N_{\mathcal{P}}$ than q. If multiple partially covered nodes are involved, we select one with the largest *minExistDNN* to compare with $N_{\mathcal{P}}$.

The detailed search algorithm is illustrated through a running example in Figure 13. For simplicity, the details of some nodes are omitted. Suppose that a bichromatic RRNN query with $t = 1$ is issued at a query point q, and two datasets, namely $\mathcal{P}$ and $\mathcal{Q}$, are considered, with the answer point from $\mathcal{P}$. In the first place, $P_{\mathcal{P}}$ contains $[(N_{\mathcal{P}}, 1)]$ and $V_{\mathcal{Q}}$ contains $\{N_{\mathcal{Q}}, N'_{\mathcal{Q}}\}$. By dequeuing $P_{\mathcal{P}}$, $N_{\mathcal{P}}$ is being examined. It is replaced by its two children $N_{\mathcal{P}1}$ and $N_{\mathcal{P}2}$. Since $N_{\mathcal{P}2}$ has $minExistDNN_{N'_{\mathcal{Q}}}(N_{\mathcal{P}1})$ smaller than $mindist(N_{\mathcal{P}1}, q)$, there must be at least one data point in $N'_{\mathcal{Q}}$ closer to any point in $N_{\mathcal{P}1}$ than q. Therefore, its $min\kappa$ is 2. Now in $P_{\mathcal{P}}$, $N_{\mathcal{P}2}$ is the head entry and its $min\kappa$ is 1. Exploring $N_{\mathcal{P}2}$ obtains p_3 and p_4. Again, since p_3's $minmaxdist(p_3, N_{\mathcal{Q}})$ is smaller than $mindist(p_3, q)$, p_3's $min\kappa$ is 2. p_4's $min\kappa$ retains 1 as $cir(p_4, q)$ covers a small portion of $N_{\mathcal{Q}}$. They both are pushed back to $P_{\mathcal{P}}$. Next, p_4 whose $min\kappa$ is the smallest is retrieved. As $N_{\mathcal{Q}}$ is the only node covered by p_4, $N_{\mathcal{Q}}$ is explored into $N_{\mathcal{Q}1}$ and $N_{\mathcal{Q}2}$, which in turn are placed back to $V_{\mathcal{Q}}$. Again, p_4 with smallest $min\kappa$ is dequeued and now no node is partially covered by $cir(p_4, q)$. p_4's $min\kappa$ is finalized and it is the final result.



Eax.	$P_{\mathcal{O}}$	$V_{\mathcal{S}}$
	$[(N_{\mathcal{P}}, 1)]$	$\{N_{\mathcal{Q}}, N'_{\mathcal{Q}}\}$
$N_{\mathcal{P}}$	$[(N_{\mathcal{P}2}, 1), (N_{\mathcal{P}1}, 2)]$	ditto
$N_{\mathcal{P}2}$	$[(p_4, 1), (N_{\mathcal{P}1}, 2), (p_3, 2)]$	ditto
p_4	$[(p_4, 1), (N_{\mathcal{P}1}, 2), (p_3, 2)]$	$\{\underline{N_{\mathcal{Q}1}}, \underline{N_{\mathcal{Q}2}}, N'_{\mathcal{Q}}\}$

Fig. 13. Example of κ-Browsing for Bichromatic RRNN

E. Discussion

The κ-Browsing algorithm can efficiently process RRNN query because the use of $min\kappa$ helps to guide the algorithm to explore the index nodes that are more likely to contain answer data points. In addition, it is expected to perform better than the κ-Counting algorithm in terms of result delivery progressiveness. Consider Figure 8. Suppose t is greater than one, the κ-Browsing algorithm first identifies p' and outputs it and then looks for the second most influenced point p. However, κ-Counting has to visit p and other nearest points prior to reaching p'. Until p' is found, both p and p' are returned.

On the other hand, the efficiency of the κ-Browsing algorithm relative to the κ-Counting algorithm would degrade if data points are uniformly distributed and/or in high data dimensionality. When data point distribution is uniform, $min\kappa$ of p (or N) will be closely proportional to $mindist(p, q)$ or $(mindist(p, N))$. As a result, the access order based on $min\kappa$'s makes no significant difference from that based on *mindist* as adopted by the κ-Counting algorithm. Even worse, additional $min\kappa$ calculation at every index level consumes considerable processing overhead. Besides, for high data dimensionality, $min\kappa$ based on MBB distance metric heuristics may provide overly conservative estimation. As a result, many index nodes with more or less the same $min\kappa$ are eventually accessed. We study all those factors in our experiments.

V. Performance Evaluation

In this section, we evaluate our proposed RRNN search algorithms, namely, the κ-Counting and κ-Browsing algorithms in comparison with the κ-Probing algorithm and the filter-and-rank (FR) described in Section I. We measure the performance of all the algorithms based on two commonly used metrics, *I/O cost* and *elapsed time*, with respect to three factors, namely, the *number of requested answer data points* (t), the *dataset cardinality* (n) and the *data dimensionality* (d). The I/O cost (in unit of *number of pages accessed*) is measured as the number of index nodes accessed from the disk. The elapsed time is measured as the time duration (in unit of *seconds*) from the query initiation to query completion that all answer data points are collected. In our experiments, we also estimate the *optimal performance* by traversing indices only for answer data points (obtained by the other evaluated algorithms) and their NNs.

We employ synthetic and real datasets in this evaluation as summarized in Table I. The data spaces for all datasets are normalized to $[0, 1)^d$. Synthetic datasets are generated following uniform (labeled as Uniform) and Gaussian distributions (labeled as Skewed). The mean and standard deviation of Gaussian distribution are fixed at 0.5 and 0.2, respectively. The dataset cardinality is varied from 10k, 50k, 100k, 500k, 1000k, 5000k and 10,000k (i.e., 10 million) and it is defaulted at 100k; and the dataset dimensionality is ranged from 2 to 8 and defaulted at 3. Real datasets include Church, School, Wave3A, Wave3B, Wave4A and Wave4B. Church and School are the 2D geographical coordinates of churches and schools in United States, respectively, obtained from the US Census Bureau[3]; Wave3A and Wave3B (Wave4A and Wave4B) are 3 (4) wave directions sampled hourly obtained from National Data Buoy Center[4].

Dataset	cardinality (n)	dimensionality (d)
Uniform/Skewed	10k through $10,000k$,	2 through 8
Church	109k	2
School	46k	2
Wave3A/Wave3B	60k	3
Wave4A/Wave4B	60k	4

TABLE I
DATASET SETTINGS

We build R*-tree [2] to support the κ-Counting algorithm, the FR and the κ-Probing algorithm, and aR-tree [10] to support the κ-Browsing algorithm. For the κ-Probing algorithm, we increase k by a factor of 2 in each round. When more than t answer data points are found, we gradually decrease k until t answer data points are collected. Also, we implement TPL [17], the currently known efficient RkNN algorithm as its underlying RkNN algorithm. For FR, KNN is selected as initial RRNN candidates. To decide K that has to be large enough to prevent a false miss, we, based on [17], adopt $10 \times d \times MAX_\kappa$ where d is data dimensionality and MAX_κ is the largest κ among all answer data points obtained from other algorithms for the same experiment settings. Besides, we adopt an aggregated count query [18] to determine the number of points inside individual circular ranges, that counts enclosed data points for multiple candidates' circular ranges in one index scan. In our experiments, the size of a page (i.e., an index node) is fixed at 4KB. We implement an index cache of 50 pages that uses LRU as the cache replacement policy. This cache alleviates some I/O costs for the κ-Probing algorithm and FR that access indices multiple times. Every run starts with a cold cache. Since both the κ-Counting and κ-Browsing algorithms need one index lookup, they are not impacted by the cache size at all.

[3]http://www.census.gov/geo/www/tiger.

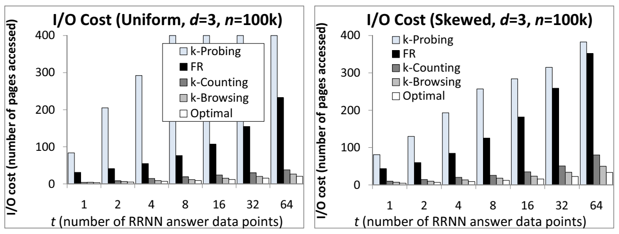
[4]http://www.ndbc.noaa.gov/historical_data.shtml.

We implemented all these algorithms with GNU C++ and conduct all experiments on Linux 2.6.9 on Intel Xeon 3.2GHz computers with 4GB RAM. Each experiment result to be presented is the average of 100 runs on query points uniformly distributed in the data space. In what follows, we present the experiment settings, results and our findings for monochromatic and bichromatic RRNN scenarios.

A. Experiments for Monochromatic RRNN

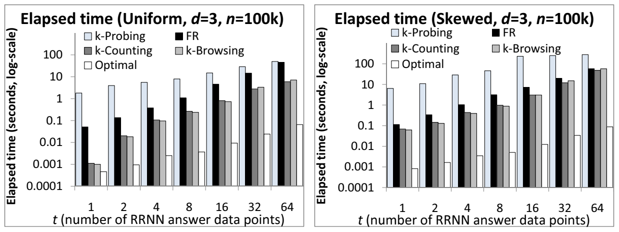
Our first experiment set focuses on monochromatic RRNN scenarios where answer data points and their NNs are from one dataset. First, we evaluate the algorithms with synthetic datasets under various number of requested answer points (t), dataset cardinality (n) and dataset dimensionality (d). Next, we evaluate their practicality using the real datasets.

Evaluation of the number of answer data points (t). We first evaluate all the algorithms by varying the number of requested answer points, t (ranged from 1 to 64). The data cardinality (n) and dimensionality (d) of datasets are fixed at $100k$ and 3, respectively. The results are plotted in Figure 14 and Figure 15. We observe from the figures that both the I/O cost and elapsed time (in log scale) for all the algorithms increase with t. This is because of the expanded search range in the data space.



(a) Uniform dataset (b) Skewed dataset

Fig. 14. The evaluation of the no. of answer data points (t) on I/O cost

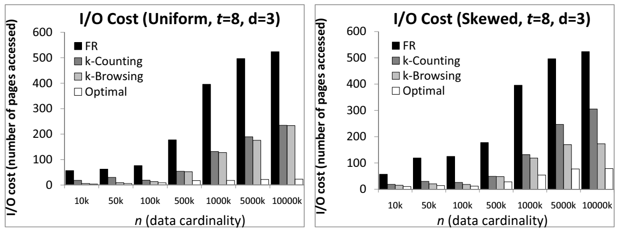


(a) Uniform dataset (b) Skewed dataset

Fig. 15. The evaluation of the no. of answer data points (t) on elapsed time

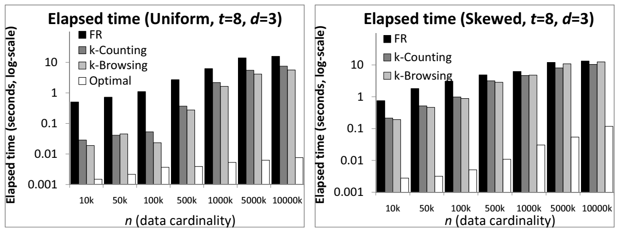
Among all the evaluated algorithms, both the κ-Counting and κ-Browsing algorithms are observed to be more efficient than FR and κ-Probing algorithms in terms of the I/O cost and the elapsed time. They access fewer pages to retrieve candidates and finalize their κ's with one index lookup. They terminate as soon as t RRNN answer points are determined. However, the FR would access some index nodes twice for candidates and their NNs, and it terminates only when all index nodes covered by the circular ranges of all candidate points are visited. This makes the FR consume a longer elapsed time and access more pages than the κ-Counting and κ-Browsing algorithms. These observations are consistent for both Uniform and Skewed datasets. We can also see that the I/O cost of the κ-Browsing algorithm performs the closest to the optimal one because $min\kappa$ estimation provides an near optimal access order of candidates. However, it takes slightly longer time than the κ-Counting algorithm for both Uniform and Skewed datasets due to expensive $min\kappa$ computation. In contrast, the κ-Probing algorithm is the incomparably worst among all the algorithms for both metrics due to repeated invocations of underlying RkNN algorithms. We omit it to save space in the rest of the following discussion.

Evaluation of the dataset cardinality (n). The second part of this experiment evaluates the performance under different dataset cardinalities (from $10k$ up to $10,000k$), while d and t are fixed at 3 and 8, respectively. As the size of the data space is fixed, the change of dataset cardinality affects the density of data points (i.e, the number of data points in an unit volume), which in turn impacts on the expected κ's of data points. Thus, the κ-Counting and κ-Browsing algorithms have to examine more data points/index nodes before they can finalize the κ's of answer points. In the mean time, the FR includes a larger pool of candidates.



(a) Uniform dataset (b) Skewed dataset

Fig. 16. The evaluation of data cardinality (n) on I/O cost

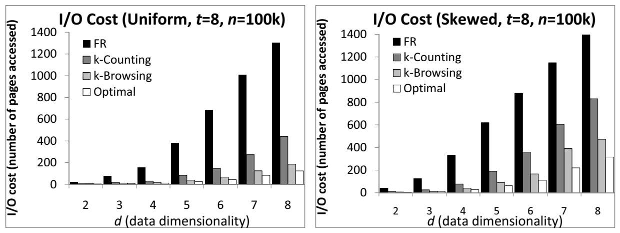


(a) Uniform dataset (b) Skewed dataset

Fig. 17. The evaluation of data cardinality (n) on elapsed time

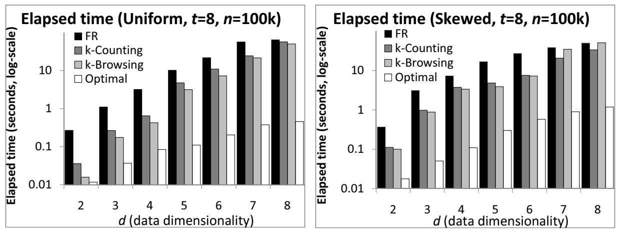
The results, depicted in Figure 16 and Figure 17, show that the I/O cost and elapsed time grow as the dataset sizes increase. For Skewed distribution, the improvement of the κ-Counting and κ-Browsing algorithms over the FR in terms of both I/O cost and elapsed time is more significant than that for Uniform distribution. It is because in Uniform datasets, an increase of the dataset cardinality affects the density of data points, thus increasing the κ's of data points. Consequently, the κ-Counting and κ-Browsing algorithms explore larger search spaces to find RRNN candidates and finalize κ's of RRNN candidates. On the other hand, the increased dataset cardinality in the Skewed datasets has a smaller impact on the density of data points around query points if they are far away from the cluster of data points. In larger datasets, the extents of index nodes are usually smaller than that in smaller datasets. As a result, the κ-Browsing algorithm can considerably save I/O costs by exploiting the counts associated with aRtree index nodes, rather than exploring all those index nodes.

Evaluation of the dataset dimensionality (d). The third part examines the sensitivity of the algorithms to dataset dimensionality. In this experiment, we fix n and t at $100k$ and 8, respectively. An increase of dimensionality expands the data space volume. While the dataset size is unchanged, the density of data points is reduced, according to our previous argument. However, as the dimensionality grows, the underlying R-tree/aR-tree becomes less efficient (this phenomenon is known as the curse of dimensionality) and it would result in more false hits that index nodes appear closer to a query point but not their enclosed data points. The performances, in terms of I/O cost and elapse time, are depicted in Figure 18 and Figure 19, respectively.



(a) Uniform dataset (b) Skewed dataset

Fig. 18. The evaluation of data dimensionality (d) on I/O cost



(a) Uniform dataset (b) Skewed dataset

Fig. 19. The evaluation of data dimensionality (d) on elapsed time

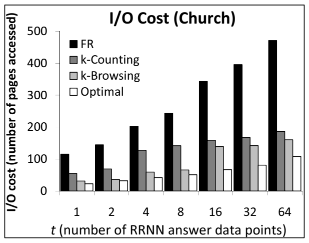
Evaluation on real datasets. Next, we examine the practicality of our algorithms by using real datasets that include Church, School, Wave3A, Wave3B, Wave4A and Wave4B datasets with t varied from 1 to 64. Figure 20 and Figure 21 show that both the κ-Counting and κ-Browsing algorithms achieve desirably good performance, and both are consistently better than the FR. The κ-Browsing algorithm is again the most I/O efficient but its elapsed time is slightly longer than that of the κ-Counting algorithm. These observations are consistent to what we obtained from synthetic datasets.

B. Experiments for Bichromatic RRNN

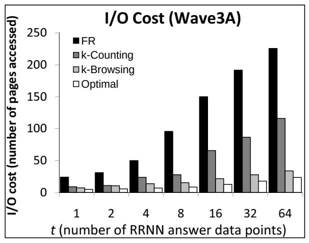
The second experiment set evaluates the performance of our algorithms for bichromatic RRNN query where answer data points are retrieved from a dataset, $\mathcal{P}$, while their NNs are obtained in another dataset, $\mathcal{Q}$. Our evaluation studies the performance of the algorithms over synthetic datasets, followed by real datasets. In this experiment, the κ-Probing algorithm is omitted since it works on TPL that is only applicable for monochromatic RkNN. The FR, as a baseline algorithm, is included for comparison in this evaluation. It first retrieves a number of candidates from $\mathcal{P}$, independent of $\mathcal{Q}$, and then performs aggregated counting queries upon $\mathcal{Q}$. We also measure the optimal performance by traversing an index of $\mathcal{P}$ for answer data points obtained by other algorithms and traversing another index of $\mathcal{Q}$ for answer data points' NN points.

For synthetic datasets, two Uniform datasets (and two Skewed datasets) are used. One is used as $\mathcal{P}$ and the other as $\mathcal{Q}$. They are generated independently. We study the performance of the algorithms against the number of answer data points (t), dataset cardinality (n) and dataset dimensionality (d).

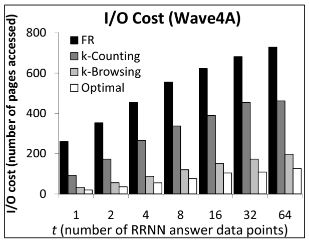
Evaluation of number of answer data points (t). Figure 22 and Figure 23 show the results obtained from synthetic datasets with various t while the cardinality and dimensionality are fixed at 100k and 3, respectively. The κ-Counting and κ-Browsing algorithms considerably outperform the FR, mainly because they carefully examine the two queried datasets in a synchronized fashion so that they can effectively retrieve answer data points and can terminate earlier. However, the FR filters candidates from $\mathcal{P}$ independently of $\mathcal{Q}$, resulting in redundant index node accesses.



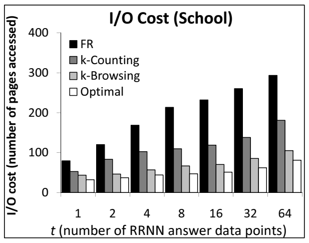
(a) Church (2D)



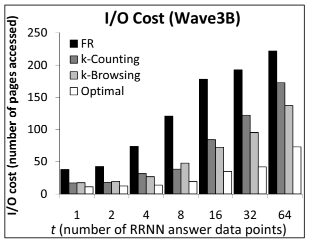
(b) Wave3A (3D)



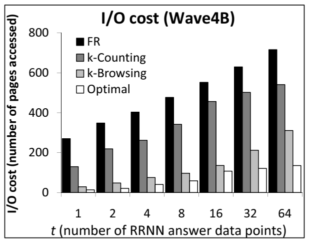
(c) Wave4A (4D)



(d) School (2D)

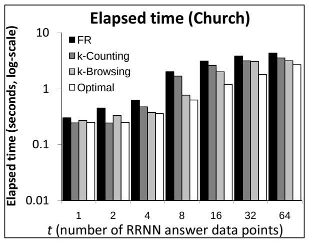


(e) Wave3B (3D)

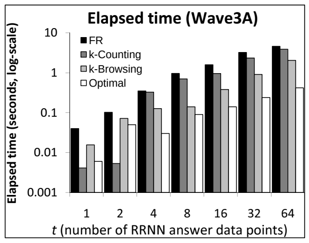


(f) Wave4B (4D)

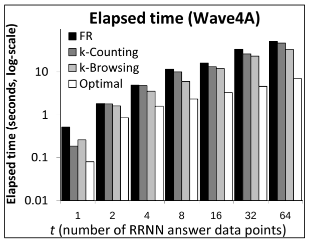
Fig. 20. The evaluation of real datasets on I/O cost



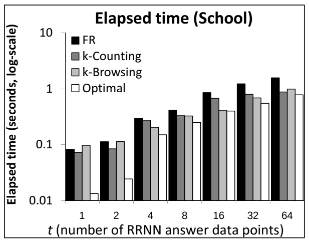
(a) Church (2D)



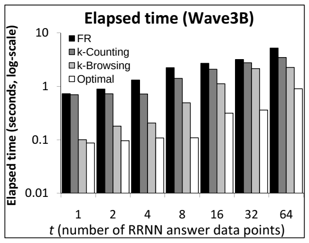
(b) Wave3A (3D)



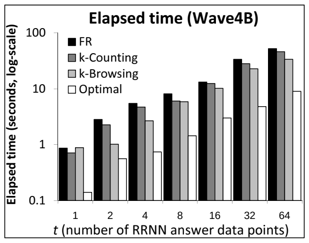
(c) Wave4A (4D)



(d) School (2D)

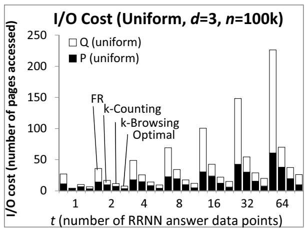


(e) Wave3B (3D)

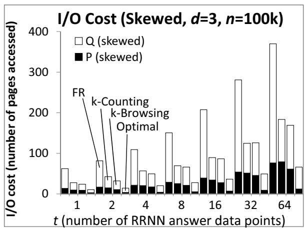


(f) Wave4B (4D)

Fig. 21. The evaluation of real datasets on elapsed time

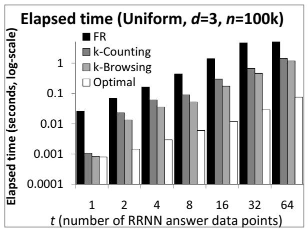


(a) Uniform datasets

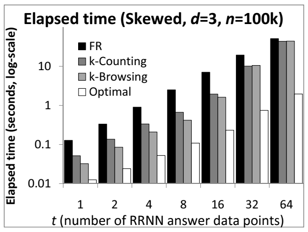


(b) Skewed datasets

Fig. 22. The evaluation of the no. of answer data points (t) on I/O cost



(a) Uniform datasets



(b) Skewed datasets

Fig. 23. The evaluation of the no. of answer data points (t) on elapsed time

As shown in the figure, the κ-Browsing algorithm performs considerably better than the κ-Counting algorithm in terms of I/O costs due to two reasons. First, aRtree provides counts associated with index nodes to facilitate the κ-Browsing algorithm to estimate $min\kappa$'s instead of direct object counting. The exploring of certain index nodes in $\mathcal{Q}$ can be saved for determining the κ's of answer data points, which alleviates some I/O costs. Second, the κ-Browsing algorithm selectively explores index nodes of $\mathcal{Q}$ when they are partially covered by a candidate data point with the smallest $min\kappa$. However, the κ-Counting algorithm has to completely expand the search space around answer data points.

Evaluation of dataset cardinality (n) and dimensionality (d). The second experiment set examines the impact of data cardinality (n). We vary the data size from 10k up to 10,000k while keeping the data dimensionality (d) and the number of anser data points (t) fixed at 3 and 8, respectively. The results are plotted in Figure 24 and Figure 25. The third experiment investigates the effect of data dimensionality by varying dimensionality from 2 to 8 and fixing t and n at 8 and 100k, respectively. Figure 26 and Figure 27 show the experiment results. In all these experiments, the FR is the weakest candidate among all the evaluated algorithms; while the κ-Browsing algorithm outperforms the κ-Counting algorithm in terms of I/O costs but reverse in terms of elapsed time due to the reasons explained previously in monochromatic RRNN scenarios.

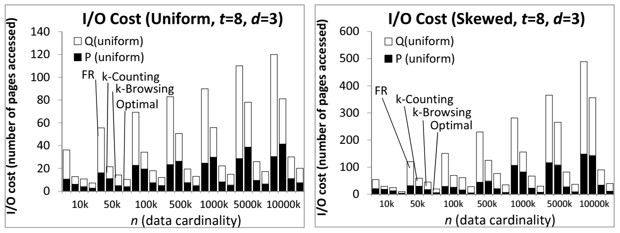

(a) Uniform dataset (b) Skewed dataset

Fig. 24. The evaluation of data cardinality (n) on I/O cost

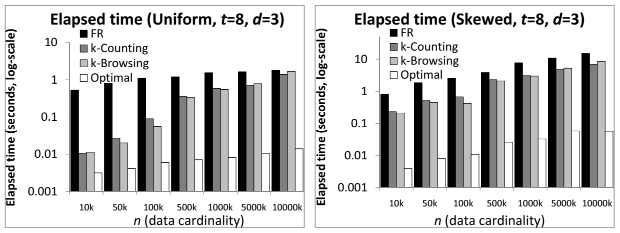

(a) Uniform dataset (b) Skewed dataset

Fig. 25. The evaluation of data cardinality (n) on elapsed time

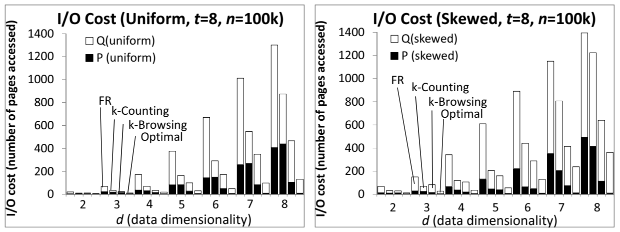

(a) Uniform datasets (b) Skewed datasets

Fig. 26. The evaluation of data dimensionality (d) on I/O cost

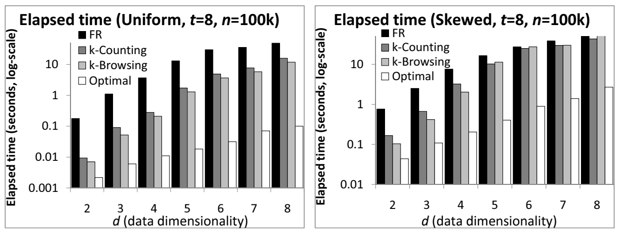
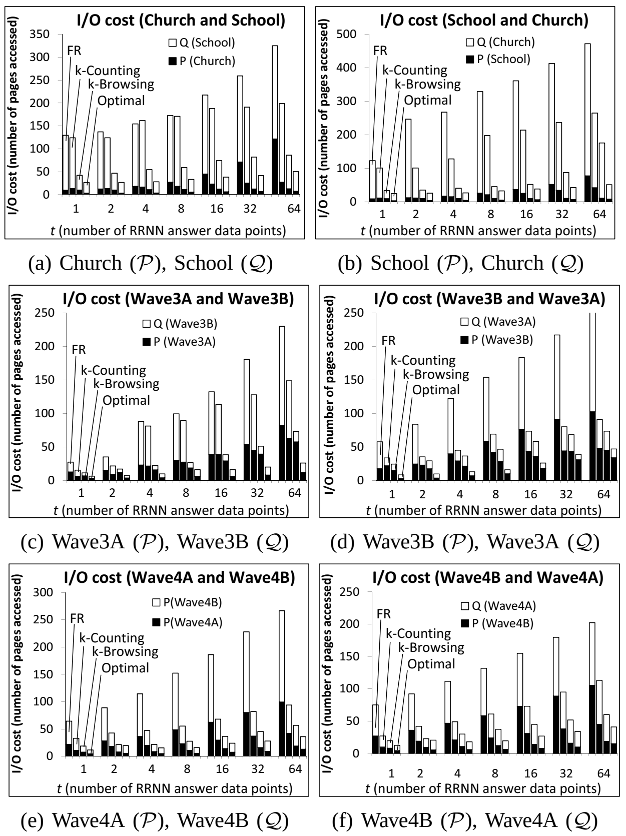

(a) Uniform datasets (b) Skewed datasets

Fig. 27. The evaluation of data dimensionality (d) on elapsed time

Evaluation on real datasets. At last, we evaluate the performance of algorithms over real datasets. Here we evaluate a pair of datasets for each setting. For instance, for 2D dataset, we use School as $\mathcal{P}$ and Church as $\mathcal{Q}$ for one setting and reverse for another. Similarly, we evaluate Wave3A and Wave3B for 3D cases and Wave4A and Wave4B for 4D cases. The results with varied t (from 1 to 64) are shown in Figure 28 and Figure 29. For all the evaluation cases, the κ-Browsing algorithm performs best consistently.

As concluded from this evaluation, the κ-Browsing algorithm is the best algorithm for both monochromatic and bichromatic RRNN search for all evaluated settings. Despite the κ-Counting algorithm is generally not better than κ-Browsing, it performs substantially better than the other straightforward approaches, namely the FR and the κ-Probing algorithms. As R-tree/aRtree performance deteriorates as dataset dimensionality increases, all the algorithms developed on it also deteriorates. We shall study alternative indices and algorithms for RRNN query for high dimensional datasets as our future work.


(a) Church ($\mathcal{P}$), School ($\mathcal{Q}$) (b) School ($\mathcal{P}$), Church ($\mathcal{Q}$)

(c) Wave3A ($\mathcal{P}$), Wave3B ($\mathcal{Q}$) (d) Wave3B ($\mathcal{P}$), Wave3A ($\mathcal{Q}$)

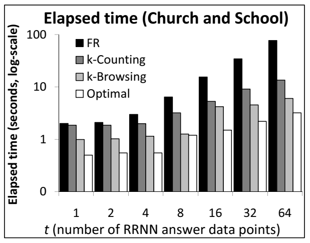
(e) Wave4A ($\mathcal{P}$), Wave4B ($\mathcal{Q}$) (f) Wave4B ($\mathcal{P}$), Wave4A ($\mathcal{Q}$)

Fig. 28. The evaluation of real datasets on I/O cost

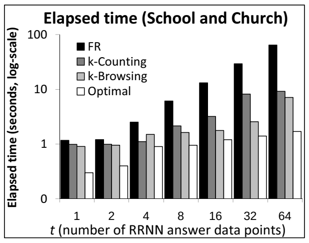
VI. Conclusion

Reverse Nearest Neighbor (RNN) and its direct variant Reverse k Nearest Neighbor (RkNN) have received considerable interests from the database research community in the past few years. In this paper, we examine some unexplored aspects of RNN/RkNN and make the following major contributions:

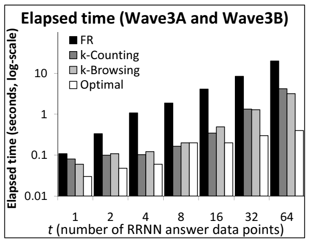
- We present a new RNN variant, namely, *Ranked Reverse Nearest Neighbor* (RRNN), that complements the conventional RNN query. RRNN distinguishes itself from the existing RNN/RkNN by: 1) discovering the influence of a query point to a specified number of data points; 2) rendering a ranked answer set based on the degrees of influence; and 3) returning the corresponding degrees of influence along with answer data points.
- We propose two innovative algorithms, κ-Counting and κ-Browsing, for efficient RRNN query processing. The κ-Counting algorithm processes data points in the order of their distances to the query point, and the κ-Browsing algorithm processes data points/index nodes based on their estimated degrees of influence. Both algorithms support multidimensional datasets, require single index lookup, provide progressive result delivery, and answer both monochromatic and bichromatic RRNN variants. In addition, with minor modification, our proposed algorithms can support RkNN with all above merits that none of existing proposal can achieve.
- Through extensive experiments on various synthetic and real datasets, the κ-Browsing and κ-Counting algorithms are shown to significantly outperform FR (the baseline approach) and the k-Probing algorithm (based on conventional RNN) in terms of I/O costs and elapsed time. Overall, the κ-Browsing is the best choice for processing RRNN query. Its I/O cost is the closest to the optimal among all the evaluated algorithms.



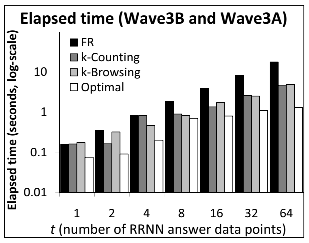
(a) Church ($\mathcal{P}$), School ($\mathcal{Q}$)



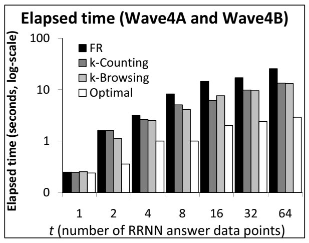
(b) School ($\mathcal{P}$), Church ($\mathcal{Q}$)



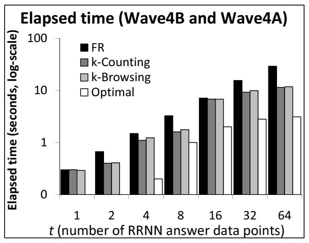
(c) Wave3A ($\mathcal{P}$), Wave3B ($\mathcal{Q}$)



(d) Wave3B ($\mathcal{P}$), Wave3A ($\mathcal{Q}$)



(e) Wave4A ($\mathcal{P}$), Wave4B ($\mathcal{Q}$)



(f) Wave4B ($\mathcal{P}$), Wave4A ($\mathcal{Q}$)

Fig. 29. The evaluation of real datasets on elapsed time

Acknowledgement

Wang-Chien Lee and Ken C. K. Lee were supported in part by the National Science Foundation under Grant no. IIS-0328881, IIS-0534343 and CNS-0626709.

References

[1] Elke Achtert, Christian Böhm, Peer Kröger, Peter Kunath, Alexey Pryakhin, and Matthias Renz. Efficient Reverse k-Nearest Neighbor Search in Arbitrary Metric Spaces. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, Chicago, IL, USA, Jun 26-29*, pages 515–526, 2006.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, USA, May 23-25*, pages 322–331, 1990.

[3] Rimantas Benetis, Christian S. Jensen, Gytis Karciauskas, and Simonas Saltenis. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. In *Proceedings of International Database Engineering & Applications Symposium, (IDEAS), Edmonton, Canada, Jul 17-19*, pages 44–53, 2002.

[4] Antonio Corral, Yannis Manolopoulos, Yannis Theodoridis, and Michael Vassilakopoulos. Closest Pair Queries in Spatial Databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, May 16-18*, pages 189–200, 2000.

[5] Hakan Ferhatosmanoglu, Ioana Stanoi, Divyakant Agrawal, and Amr El Abbadi. Constrained Nearest Neighbor Queries. In *Proceedings of the 7th International Symposium of Advances in Spatial and Temporal Databases (SSTD), Redondo Beach, CA, USA, Jul 12-15*, pages 257–278, 2001.

[6] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, Boston, MA, USA, Jun 18-21*, pages 47–57, 1984.

[7] Gísli R. Hjaltason and Hanan Samet. Distance Browsing in Spatial Databases. *ACM Transactions on Database System (TODS)*, 24(2):265–318, 1999.

[8] Flip Korn and S. Muthukrishnan. Influence Sets Based on Reverse Nearest Neighbor Queries. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, May 16-18*, pages 201–212, 2000.

[9] Flip Korn, S. Muthukrishnan, and Divesh Srivastava. Reverse Nearest Neighbor Aggregates Over Data Streams. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), Toronto, Canada, Aug 20 - 23*, pages 814–825, 2002.

[10] Iosif Lazaridis and Sharad Mehrotra. Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, CA, USA, May 21-24*, pages 401–412, 2001.

[11] Mong-Li Lee, Wynne Hsu, Christian S. Jensen, Bin Cui, and Keng Lik Teo. Supporting Frequent Updates in R-Trees: A Bottom-Up Approach. In *Proceedings of 29th International Conference on Very Large Data Bases (VLDB), Berlin, Germany, Sep 9-12*, pages 608–619, 2003.

[12] King-Ip Lin, Michael Nolen, and Congjun Yang. Applying Bulk Insertion Techniques for Dynamic Reverse Nearest Neighbor Problems. In *Proceedings of the 7th International Database Engineering and Applications Symposium (IDEAS), Hong Kong, China, Jul 16-18*, pages 290–297, 2003.

[13] Nick Roussopoulos, Stephen Kelley, and Frédéic Vincent. Nearest Neighbor Queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, CA, USA, May 22-25*, pages 71–79, 1995.

[14] Amit Singh, Hakan Ferhatosmanoglu, and Ali Saman Tosun. High Dimensional Reverse Nearest Neighbor Queries. In *Proceedings of the 2003 ACM International Conference on Information and Knowledge Management (CIKM), New Orleans, LA, USA, Nov 2-8*, pages 91–98, 2003.

[15] Ioana Stanoi, Divyakant Agrawal, and Amr El Abbadi. Reverse Nearest Neighbor Queries for Dynamic Databases. In *Proceedings of the 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000.

[16] Ioana Stanoi, Mirek Riedewald, Divyakant Agrawal, and Amr El Abbadi. Discovery of Influence Sets in Frequently Updated Databases. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB), Roma, Italy, Sep 11-14*, pages 99–108, 2001.

[17] Yufei Tao, Dimitris Papadias, and Xiang Lian. Reverse kNN Search in Arbitrary Dimensionality. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada, Aug 31 - Sep 3*, pages 744–755, 2004.

[18] Chenyi Xia, Wynne Hsu, and Mong-Li Lee. ERkNN: Efficient Reverse k-Nearest Neighbors Retrieval with Local kNN-Distance Estimation. In *Proceedings of the 2005 ACM International Conference on Information and Knowledge Management (CIKM), Bremen, Germany, Oct 31 - Nov 5*, pages 533–540, 2005.

[19] Tian Xia and Donghui Zhang. Continuous Reverse Nearest Neighbor Monitoring. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE), Altanta, GA, USA, Apr 3-8*, page 77, 2006.

[20] Tian Xia, Donghui Zhang, Evangelos Kanoulas, and Yang Du. On Computing Top-t Most Influential Spatial Sites. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), Trondheim, Norway, Aug 30-Sep 2*, pages 946–957, 2005.

[21] Xiaopeng Xiong and Walid G. Aref. R-trees with Update Memos. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE), Altanta, GA, USA, Apr 3-8*, page 22, 2006.

[22] Congjun Yang and King-Ip Lin. An Index Structure for Efficient Reverse Nearest Neighbor Queries. In *Proceedings of the 17th International Conference on Data Engineering (ICDE), Heidelberg, Germany, Apr 2-6*, pages 485–492, 2001.

[23] Man Lung Yiu and Nikos Mamoulis. Reverse Nearest Neighbors Search in Ad-hoc SubSpaces. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE), Altanta, GA, USA, Apr 3-8*, page 76, 2006.

[24] Man Lung Yiu, Dimitris Papadias, Nikos Mamoulis, and Yufei Tao. Reverse Nearest Neighbors in Large Graphs. In *Proceedings of the 21st International Conference on Data Engineering (ICDE), Tokyo, Japan, Apr 5-8*, pages 186–187, 2005.