Ranked Reverse Nearest Neighbor Search

Ken C.K. Lee, Baihua Zheng, *Member*, *IEEE*, and Wang-Chien Lee, *Member*, *IEEE*

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 (NNs) are q. Reverse k-NN ($\mathrm{R}k\mathrm{NN}$) query (where $k \geq 1$) generalizes RNN query to find data points whose $k\mathrm{NNs}$ *include* q. For $\mathrm{R}k\mathrm{NN}$ query semantics, q is said to have an influence on 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 $\kappa_p\mathrm{th}$ NN of p. We introduce a new variant of RNN query, namely, *Ranked RNN* (RRNN) query, that retrieves t data points most influenced by q, i.e., the t data points having the smallest κs with respect to q. To answer this RRNN query efficiently, we propose two novel algorithms, κ-$\mathrm{Counting}$ and κ-$\mathrm{Browsing}$ 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 $\mathrm{R}k\mathrm{NN}$ query.

Index Terms—Reverse Nearest Neighbor query, ranking, search algorithm.

1 Introduction

1.1 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-NN ($\mathrm{R}k\mathrm{NN}$) with $k \geq 1$ generalizes RNN to find data points whose kNN *include* q. $\mathrm{R}k\mathrm{NN}$ query is different from (and even more complicated than) kNN query because of asymmetric NN relationship between two data points in a data set. That means if a query point q has found the NN point p ($\in \mathcal{P}$), p may have other data points else (i.e., other than q) as its NNs.

The primary goal of $\mathrm{R}k\mathrm{NN}$ 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 NN 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 that 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 1. When q is the second NN of another data point p', q is the second most influential point to p' and $\kappa_{p'}$ is 2, and so on. Based on the definition of κ, $\mathrm{R}k\mathrm{NN}$ query can be interpreted as to retrieve data points with their κs not exceeding a given threshold parameter k as formally stated in Definition 2.

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

• *K.C.K. Lee and W.-C. Lee are with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802. E-mail: {cklee, wlee}@cse.psu.edu.*
• *B. Zheng is with the School of Information Systems, Singapore Management University, Singapore. E-mail: bhzheng@smu.edu.sg.*

Manuscript received 23 Aug. 2007; revised 25 Dec. 2007; accepted 17 Jan. 2008; published online 28 Jan. 2008.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2007-08-0429.
Digital Object Identifier no. 10.1109/TKDE.2008.36.

Definition 1: Degree of influence. *Given a data set $\mathcal{P}$ 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: $\mathrm{R}k\mathrm{NN}$ query. *Given a data set $\mathcal{P}$ (and $\mathcal{Q}$ when bichromatic $\mathrm{R}k\mathrm{NN}$ is considered) and a query point q, $\mathrm{R}k\mathrm{NN}$ query returns a set of data points whose κs do not exceed k, an influence threshold setting, i.e., $\mathrm{R}k\mathrm{NN}(q) = \{p \mid p \in \mathcal{P} \wedge \kappa_p \leq k\}$.*

$\mathrm{R}k\mathrm{NN}$ query has no control of the answer set size since the setting of k does not determine the answer set size. For example, as reported in [15], a monochromatic R1NN query in a 2D space may 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, $\mathrm{R}k\mathrm{NN}$ 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 κs provided) ordered by their κs. 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

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

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 whose preferences match well with the product, but the product may not receive high ranks to those customers due to the 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 the *Ranked RNN* (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 1. When $t = |\mathcal{P}|$ (i.e., the cardinality of the data set), RRNN renders a sorted list of all data points according to their degrees of influence. Since κs 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 data set $\mathcal{P}$ (and $\mathcal{Q}$ when bichromatic scenario is considered), a query point q, and a requested number of answer data points t, RRNN query returns t tuples (p, κ_p), where $p \in \mathcal{P}$, and κ_p is p's degree of influence. Formally, $RRNN_t(q) = \{(p, \kappa_p) \mid p \in \mathcal{P}' \wedge |\mathcal{P}'| = t \wedge \mathcal{P}' \subseteq \mathcal{P} \wedge \forall_{x \in (\mathcal{P}-\mathcal{P}')} \kappa_p < \kappa_x\}$.*

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 the calculated matching degree. RRNN query can identify a given number of top-matched candidates, along with their corresponding matching degrees.

1.2 Possible Solutions

Although RkNN query is also based on the degrees of influence (see Definitions 2 and 3), 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 κs equal to 2. This process repeats with incremented k at each run until t answer data points and their κs are obtained, which clearly suffers from redundant processing among different runs. A slight improvement can be made by exponentially increasing the ks in the series of RkNN invocations, e.g., setting k to $1, 2, 4, 8, \ldots$ and so forth. 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 is equal to the number of data points, including q, fallen inside the circle. Thus, a naive approach to processing an RRNN query is to count the number of NNs for all data points exhaustively as aforementioned. 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 κs 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-refine query processing paradigm commonly used by RkNN search algorithms (to be discussed in Section 2). It has two phases: 1) in the filter phase, it retrieves K NN data points ($t \leq K$) to a query point as result candidates and 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 κs 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. Setting K to a large value may avoid false miss, but this makes the search suffer a serious performance penalty.

1.3 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 obtain κs 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 κs intuitively have short distances to q (i.e., small circles formed), the κ-Counting algorithm examines data points based on *their distances to the query point q*. While we determine the κ of one data point at a time, the κs of many other data points are incrementally obtained based on the 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 κs and, hence, are excluded from the answer set. Thus, κ-Counting algorithm, based on distance order, needs to process more data points. The details about this algorithm will be discussed in Section 3.

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 the 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 4.

Both the κ-Counting and the κ-Browsing algorithms support multidimensional data sets. Other than R-tree/aR-tree maintenance, they do not incur any preprocessing 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/dimensionality of the data set and various values of t (the required number of answer points). The result indicates that the κ-Browsing algorithm generally performs the best in terms of I/O costs and elapsed time.

1.4 Organization of This Paper

The remainder of this paper is organized as follows: Section 2 reviews R-tree and existing RNN/RkNN search algorithms. Sections 3 and 4 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 5. Finally, Section 6 concludes this paper.

2 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.

2.1 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). 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 Fig. 1a, $mindist(q, N)$ 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 $minmaxdist(q, N)$.

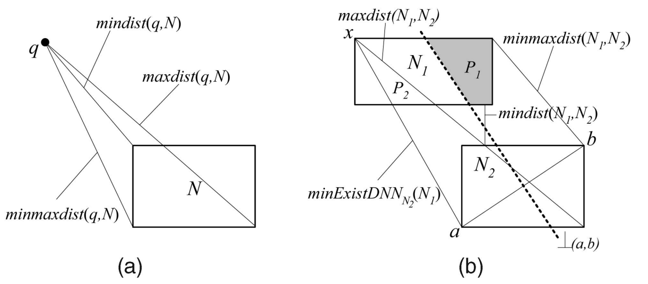


Fig. 1. Distance metrics. (a) Between a data point and an MBB. (b) Between two MBBs.

Besides, another set of distance metrics is defined between two MBBs. With the same terminologies, *mindist*, *minmaxdist*, and *maxdist* [4] are exemplified in Fig. 1b. $mindist(N_1, N_2)$ 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 Fig. 1b, 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.

2.2 RNN/RkNN Search Algorithms

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

2.2.1 *Precomputation-Based* RNN/RkNN *Search Algorithms*

Precomputation-based approaches preexecute 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 point 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

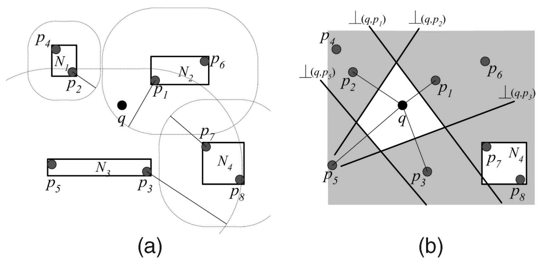


Fig. 2. RNN algorithms. (a) RdNN-tree. (b) TPL algorithm.

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 searches simultaneously. Fig. 2a 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 $\mathrm{R}k\mathrm{NN}$ queries for a fixed k and they incur a very high index construction and update overhead [12]. To support various k, Achtert et al. [1] and Xia et al. [18] suggested to estimate kNN distance at the run time instead of maintaining actual possible kNN distances.

2.2.2 *Dynamic* $\mathrm{RNN}/\mathrm{R}k\mathrm{NN}$ *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 $\mathrm{RNN}/\mathrm{R}k\mathrm{NN}$ 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 $\mathrm{RNN}/\mathrm{R}k\mathrm{NN}$ answers are identified as candidates from the entire data set, 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 $\mathrm{R}k\mathrm{NN}$ 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 FR algorithm (discussed in Section 1.2) borrows this idea.

To guarantee the completeness of results, Tao et al. [17] proposed the TPL algorithm that exploits a half-plane property in space to locate $\mathrm{R}k\mathrm{NN}$ candidates. The algorithm examines data points based on distance browsing [7]. Every time when an unexplored NN 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 $\mathrm{R}k\mathrm{NN}$ 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 Fig. 2b, 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 $\{p_1, p_5\}$.

2.3 Other RNN Algorithms

Various $\mathrm{RNN}/\mathrm{R}k\mathrm{NN}$ 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 that 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 bichromatic RNN scenarios, aiming at finding a rank list of data points from a set of query points, $\mathcal{Q}$, that influence most of the data points in $\mathcal{P}$. In other words, this work intends to find the most *influential query points*. Different from this work, our work finds *the most influenced data points to a single query point* and ranks them.

3 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.

3.1 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, outward 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 κcnts 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 κcnts (that equal to κs) are collected as answer data points. The algorithm keeps expanding the search space and incrementing the κcnts of data points until t answer data points are obtained.

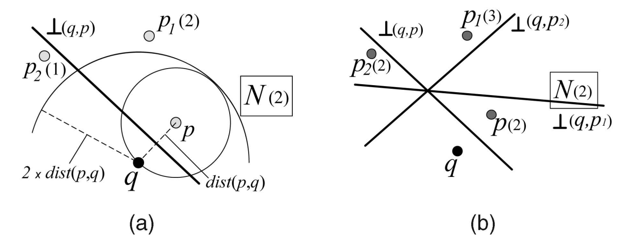


Fig. 3. Basic idea of κ-Counting algorithm. (a) Bisector $\perp_{(q,p)}$. (b) κcnt of all points and node.

We use half-planes as [17] to determine whose κcnts 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 the κcnts of those data points are incremented by one. With the use of half-planes, the κcnt of a data point p is equal to 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 κcnts of individual data points or groups of data points, we adopt the R-tree index. An example is illustrated in Fig. 3a, where the κcnts 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 κcnts incremented by one. On the other hand, p_2 is outside HP_p so its κcnt remains one.

As shown in Fig. 3b, after examining three data points, p, p_1, and p_2, the κcnts 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 κcnts associated with N's may be greater than, but definitely not less than, that of N.

Certainly, after examining all data points/half-planes, the κcnts of all data points can be finalized (converged and equal to κs). However, since only the t most influenced data points (i.e., those with smallest κs) 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 nondecreasing distance order, the 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 Fig. 3a). Thus, p's κcnt can be finalized and equals κ. □

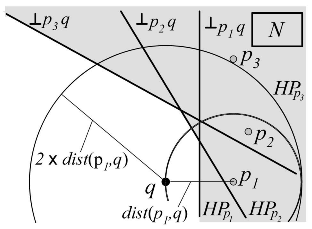
```
Algorithm $\kappa$-Counting($q, root, t$)
Input:   a query point (q), the root index of $\mathcal{P}$ ($root$),
         the number of answer data points ($t$)
Local:   a priority queue ($P$), a candidate set ($C$),
         a finalized candidate set ($F$) and a half-plane set ($H$);
Output:  $t$ 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 data points with finalized $\kappa cnt$'s */
 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 points with the smallest finalized $\kappa cnt$'s */
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.

Though the κcnts of some data points can be finalized earlier, it is not guaranteed that those points with early finalized κcnts must be the RRNN query answers. Until their κs are certain to be the smallest, they will not be output as a part of the RRNN query result. In Sections 3.2 and 3.3, we present κ-Counting algorithm for monochromatic and bichromatic RRNN application scenarios.

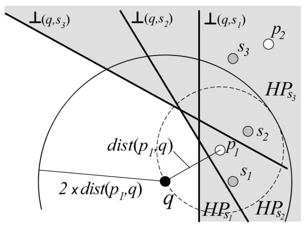
3.2 κ-Counting Algorithm for Monochromatic RRNN

The κ-Counting algorithm for monochromatic RRNN is based on distance browsing [7], as described in the pseudocode in Fig. 4. In this algorithm, data points have to be examined through three stages, namely, *queued* (pending for examination), *examined* (examined but with nonfinalized κs), and *finalized* (examined with finalized κs), 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 κcnts in the histogram smaller than p's κcnt reach zero, p's κcnt is guaranteed to be the smallest. Moreover, the histogram is very update efficient.



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.



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.

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 κcnts finalized according to Lemma 1 are moved to F (lines 4-6). Further, those data points in F with the smallest κcnts are output as the partial RRNN query result *immediately* (lines 20-23). As long as t answer data points are collected, the algorithm terminates.

Fig. 5 shows a sample 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_1s finalized κ is smaller than that of the rest of the data points (i.e., p_2, p_3, and N), it is output as the RRNN query result to complete the search.

3.3 κ-Counting Algorithm for Bichromatic RRNN

The κ-Counting algorithm for bichromatic RRNN query considers two data sets $\mathcal{P}$ 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 κcnts 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 3.2. 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 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 κcnts 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 κcnts, 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, as its κcnt is finalized, 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 κcnts of some data points in F are the smallest, we maintain a histogram of κcnts of data points in C and P. The algorithm terminates when t RRNN answer data points are collected.

As it is similar to that for monochromatic RRNN query, the pseudocode of the κ-Counting algorithm for bichromatic RRNN query is omitted to save space. Fig. 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. First, s_1 is examined, a half-plane, HP_{s_1}, is created with respect to s_1 and q, and the κcnts of both p_1 and p_2 are incremented to 2. Second, p_1 is examined and buffered in C. Third, s_2 is examined and its half-plane HP_{s_2} covers p_1 and p_2. As a result, the κcnts 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

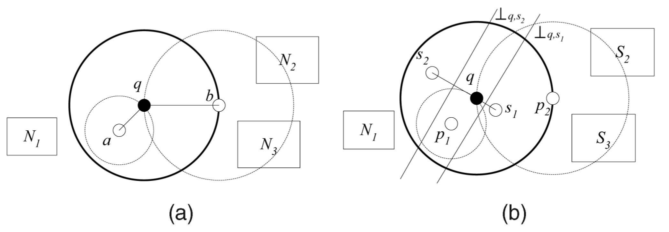


Fig. 7. Scenario about FR. (a) Monochromatic RRNN. (b) Bichromatic RRNN.

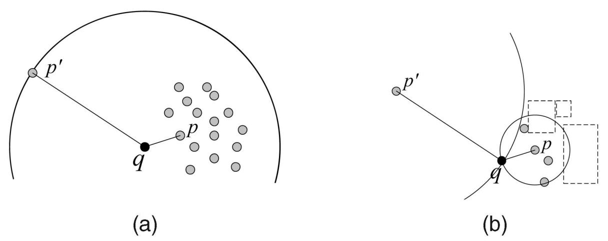


Fig. 8. Skewed data set. (a) κ-Counting. (b) κ-Browsing.

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.

3.4 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 the existing RkNN algorithms can offer. Besides, the κ-Counting algorithm can operate without specifying t, defaulting $t = \infty$. This actually sorts all the points according to ascending order of κ, i.e., the degree of influence received from q.

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. Fig. 7a 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. Fig. 7b 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 Q are accessed. However, for the same situation, the κ-Counting algorithm does not need to explore that many index nodes of Q and does not even 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_1s, thus the κ-Counting algorithm terminates earlier.

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 data sets. As illustrated in Fig. 8a, 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 κcnts 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.

4 κ-Browsing Algorithm

In this section, we detail the κ-Browsing algorithm, which is based on a notion of $min\kappa$, a lower bound estimation of κ for a data point. To facilitate the calculation of $min\kappa$s, we adopt aR-tree [10], which is widely used to support aggregation query. For RRNN and RkNN, determining κs that counts the number of NNs with respect to a data point is a sort of aggregation. aR-tree is an R-tree 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 nonleaf node is equal to 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 the description of the κ-Browsing algorithm for monochromatic and bichromatic RRNN scenarios.

4.1 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 Fig. 8b (which depicts the same scenario as Fig. 8a). Based on 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 $\kappa_{p'}$ are 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 κs. 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

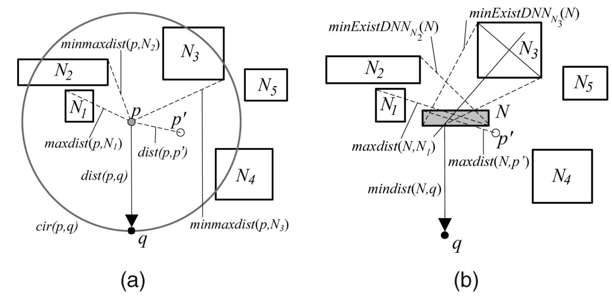


Fig. 9. Examples of $min\kappa$. (a) $min\kappa(q,p,\mathcal{V})$. (b) $min\kappa(q,N,\mathcal{V})$.

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 Fig. 8b, 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.

4.2 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 data set, 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 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. Fig. 9a 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 toward $min\kappa$ of p. Similarly, N_1 is fully covered by the circle (i.e., $maxdist(p,N_1) \leq dist(p,q)$), which means that 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) toward $min\kappa$. 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 toward $min\kappa$ 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, $min\kappa$ is estimated as $(1+3+N_1.cnt)$. Notice that we need to add 1 for q to $min\kappa$.

Hence, based on a given V (i.e., the current state of explored data space in terms of index nodes and data points), the following expression 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

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

Lemma 2. *Given a data point p, a query point q, and a set of index nodes and data points maintained in V, $min\kappa(q,p,V)$ is equal to 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 2. 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

$$dist(p,p') \leq maxdist(N,p') \text{ and } mindist(N,q) \leq dist(p,q),$$

$maxdist\ (N,p') \leq mindist(N,q)$ 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

$$dist(p,N') \leq maxdist(N,N') \text{ and } mindist(N,q) \leq dist(p,q),$$

$maxdist(N,N') \leq mindist(N,q)$ 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 Lemmas 3, 4, and 5, the following expression 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):

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

where

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

Fig. 9b 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, due to 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 the monotone properties of $min\kappa$ (defined in Lemmas 6 and 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, $mindist(N_c,q) \geq mindist(N,q)$. In addition, 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 (1) and (2), N_c will cover either the 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, the 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 $N'_1,\ldots,N'_n$, resulting in V'. For any data point p or index node N, the following two statements should be true:*

1. $min\kappa(q,N,V') \geq min\kappa(q,N,V)$,
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 (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

   $$minExistDNN_{N'_i}(N)$$

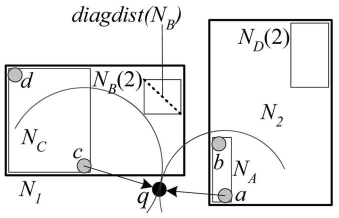
   is not greater than $minExistDNN_{N'}(N)$ and may be smaller than $mindist(N,q)$, so at least 1 is counted. Even more, some N_i may have $maxdist(N,N'_i) < mindist(N,q)$, resulting in the contribution of $N'_i.cnt$ to $min\kappa$. Hence, $\delta \geq 0$.
4. $mindist(N,q) < mindist(N,N')$. In this case, 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. □

4.3 κ-Browsing Algorithm for Monochromatic RRNN

The κ-Browsing algorithm for monochromatic RRNN considers only one data set, $\mathcal{P}$. In the algorithm, a priority queue P is adopted to keep unexamined data points/index nodes according to the nondecreasing order of their $min\kappa$s. In the case of a tie that two or more data points/index nodes have the 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 κs 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,



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.

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

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

Let us see how the κ-Browsing algorithm runs as exemplified in Fig. 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, the 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 Fig. 10. When c is dequeued, its $min\kappa$ is equal to 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 Lemmas 6 and 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 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 of the entries in the queue. According to Lemmas 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.

```
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 $(\epsilon, k)$ 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*.

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 Fig. 11.

In addition to 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 reinserted into P with updated $min\kappa$ for next examination.

We depict the pseudocode of the κ-Browsing algorithm for the monochromatic RRNN in Fig. 12. It takes a query point, q, the root node, *root*, of the aR-tree of a data set $\mathcal{P}$, 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 that 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).

4.4 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 pseudocode 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 nondecreasing 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 $min\kappa$.

```
Algorithm κ-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.

Though two data sets 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 the 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 the 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 (2), if

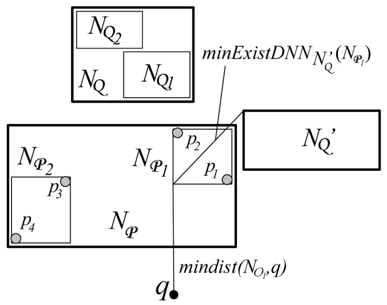
$$maxdist(N_{\mathcal{P}}, N'_{\mathcal{Q}}) \leq mindist(N_{\mathcal{P}}, q),$$

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 $min\kappa$ to be 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 Fig. 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 data sets, namely, $\mathcal{P}$ and $\mathcal{Q}$, are considered, with the answer



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.

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.

4.5 Discussion

The κ-Browsing algorithm can efficiently process RRNN query because the use of $min\kappa$ helps 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 Fig. 8. Suppose t is greater than 1, the κ-Browsing algorithm first identifies p' and outputs it and then looks for the second most influenced point p. However, the κ-Counting algorithm 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

TABLE 1
Data Set Settings

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

the same $min\kappa$ are eventually accessed. We study all those factors in our experiments.

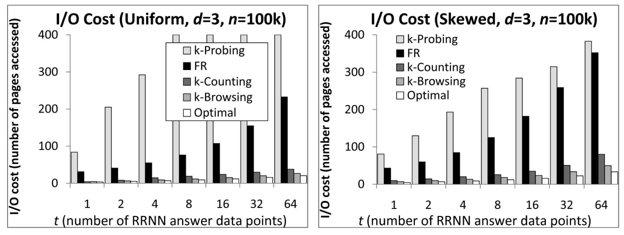
5 Performance Evaluation

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 FR described in Section 1. 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 *data set cardinality* (n), and the *data dimensionality* (d). The I/O cost (in units 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 units 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 data sets in this evaluation, as summarized in Table 1. The data spaces for all data sets are normalized to $[0,1)^d$. Synthetic data sets 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 data set cardinality is varied from 10k, 50k, 100k, 500k, 1,000k, 5,000k, and 10,000k (i.e., 10 million) and it is defaulted at 100k; and the data set dimensionality ranged from 2 to 8 and defaulted at 3. Real data sets include Church, School, Wave3A, Wave3B, Wave4A, and Wave4B. Church and School are the 2D geographical coordinates of churches and schools in US, respectively, obtained from the US Census Bureau;[3] Wave3A and Wave3B (Wave4A and Wave4B) are three (four) wave directions sampled hourly, which are obtained from the National Data Buoy Center.[4]

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. In addition, 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 the 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, which 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 4 Kbytes. 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.



(a) (b)

Fig. 14. The evaluation of the number of answer data points (t) on I/O cost. (a) Uniform data set. (b) Skewed data set.

We implemented all these algorithms with GNU C++ and conducted all experiments on Linux 2.6.9 on Intel Xeon 3.2-GHz computers with 4-Gbyte RAM. Each experimental 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.

5.1 Experiments for Monochromatic RRNN

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

5.1.1 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 data sets are fixed at $100k$ and 3, respectively. The results are plotted in Figs. 14 and 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.

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 κs 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

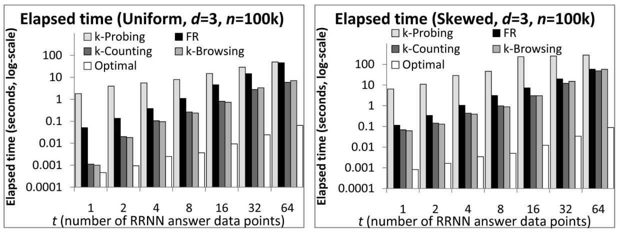


Fig. 15. The evaluation of the number of answer data points (t) on elapsed time. (a) Uniform data set. (b) Skewed data set.

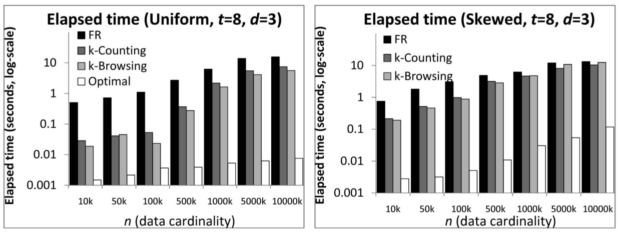


Fig. 17. The evaluation of data cardinality (n) on elapsed time. (a) Uniform data set. (b) Skewed data set.

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 data sets. 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 a near optimal access order of candidates. However, it would take slightly longer time than the κ-Counting algorithm for both Uniform and Skewed data sets due to expensive $min\kappa$ computation. In contrast, the κ-Probing algorithm is 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.

5.1.2 Evaluation of the Data Set Cardinality (n)

The second part of this experiment evaluates the performance under different data set 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 data set cardinality affects the density of data points (i.e., the number of data points in a unit volume), which in turn has an impact on the expected κs of data points. Thus, the κ-Counting and κ-Browsing algorithms have to examine more data points/index nodes before they can finalize the κs of answer points. In the meantime, the FR includes a larger pool of candidates.

The results, depicted in Figs. 16 and 17, show that the I/O cost and the elapsed time grow as the data set 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 data sets, an increase of the data set cardinality affects the density of data points, thus increasing the κs of data points. Consequently, the κ-Counting and κ-Browsing algorithms explore larger search spaces to find RRNN candidates and finalize κs of RRNN candidates. On the other hand, the increased data set cardinality in the Skewed data sets 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 data sets, the extents of index nodes are usually smaller than that in smaller data sets. As a result, the κ-Browsing algorithm can considerably save I/O costs by exploiting the counts associated with aR-tree index nodes rather than exploring all those index nodes.

5.1.3 Evaluation of the Data Set Dimensionality (d)

The third part examines the sensitivity of the algorithms to data set 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 data set 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 Figs. 18 and 19, respectively.

5.1.4 Evaluation on Real Data Sets

Next, we examine the practicality of our algorithms by using real data sets that include Church, School, Wave3A, Wave3B, Wave4A, and Wave4B data sets with t varied from 1 to 64. Figs. 20 and 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 could be slightly longer than that of the κ-Counting algorithm. These observations are consistent to what we obtained from synthetic data sets.

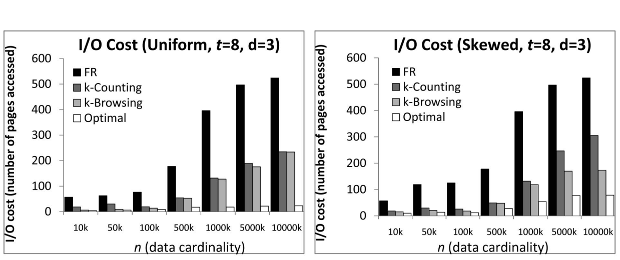


Fig. 16. The evaluation of data cardinality (n) on I/O cost. (a) Uniform data set. (b) Skewed data set.

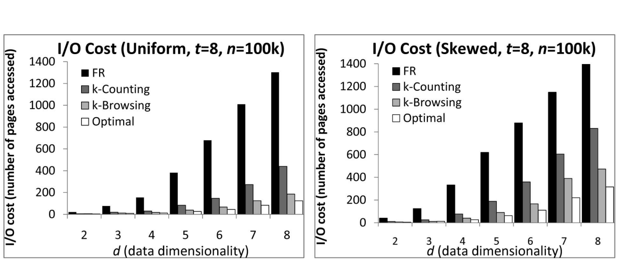
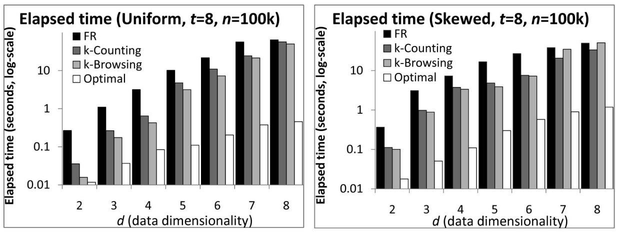


Fig. 18. The evaluation of data dimensionality (d) on I/O cost. (a) Uniform data set. (b) Skewed data set.



(a) (b)

Fig. 19. The evaluation of data dimensionality (d) on elapsed time. (a) Uniform data set. (b) Skewed data set.

5.2 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 data set, $\mathcal{P}$, while their NNs are obtained in another data set, $\mathcal{Q}$. Our evaluation studies the performance of the algorithms over synthetic data sets, followed by real data sets. 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 data sets, two Uniform data sets (and two Skewed data sets) 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), data set cardinality (n), and data set dimensionality (d).

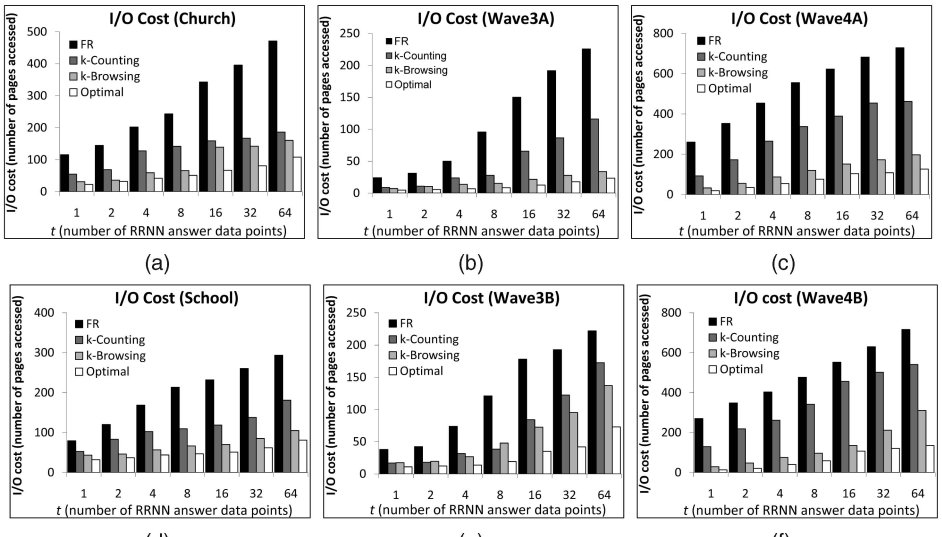
5.2.1 *Evaluation of Number of Answer Data Points* (t)

Figs. 22 and 23 show the results obtained from synthetic data sets 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 data sets 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.

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, aR-tree provides counts associated with index nodes to facilitate the κ-Browsing algorithm to estimate $min\kappa$s instead of direct point counting. The exploring of certain index nodes in $\mathcal{Q}$ can be saved for determining the κs 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.

5.2.2 *Evaluation of Data Set 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 10,000 K while keeping the data dimensionality (d) and the number of answer data points (t) fixed at 3 and 8, respectively. The results are plotted in Figs. 24 and 25. The third experiment investigates the effect of data dimensionality by varying the dimensionality from 2 to 8 and fixing t and n at 8 and $100k$, respectively. Figs. 26 and 27 show the experimental 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) (b) (c)

(d) (e) (f)

Fig. 20. The evaluation of real data sets on I/O cost. (a) Church (2D). (b) Wave3A (3D). (c) Wave4A (4D). (d) School (2D). (e) Wave3B (3D). (f) Wave4B (4D).

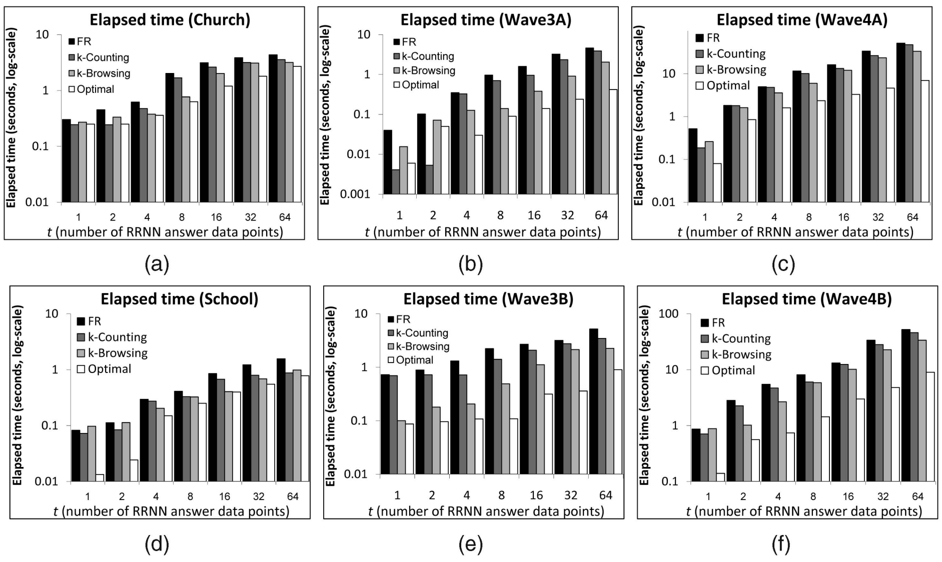


Fig. 21. The evaluation of real data sets on elapsed time. (a) Church (2D). (b) Wave3A (3D). (c) Wave4A (4D). (d) School (2D). (e) Wave3B (3D). (f) Wave4B (4D).

5.2.3 Evaluation on Real Data Sets

At last, we evaluate the performance of algorithms over real data sets. Here, we evaluate a pair of data sets for each setting. For instance, for 2D data set, 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 Figs. 28 and 29. For all the evaluation cases, the κ-Browsing algorithm consistently performs the best.

As concluded from this evaluation, the κ-Browsing algorithm is the best algorithm for both monochromatic and bichromatic RRNN searches 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/aR-tree performance deteriorates as data set dimensionality increases, all the algorithms developed on it also deteriorate. We shall study alternative indices and algorithms for RRNN query for high-dimensional data sets as our future work.

6 Conclusion

RNN and its direct variant RkNN have received considerable interests from the database research community in the

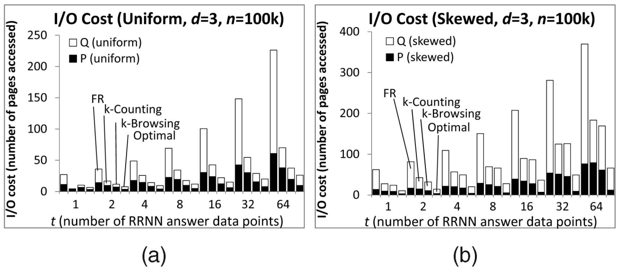


Fig. 22. The evaluation of the number of answer data points (t) on I/O cost. (a) Uniform data sets. (b) Skewed data sets.

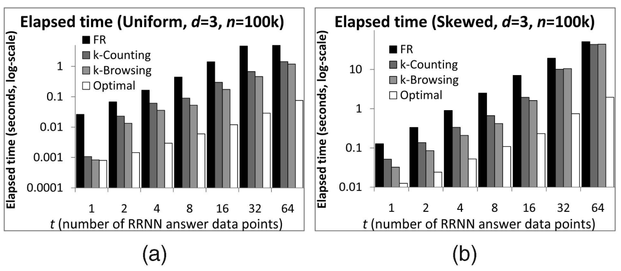


Fig. 23. The evaluation of the number of answer data points (t) on elapsed time. (a) Uniform data sets. (b) Skewed data sets.

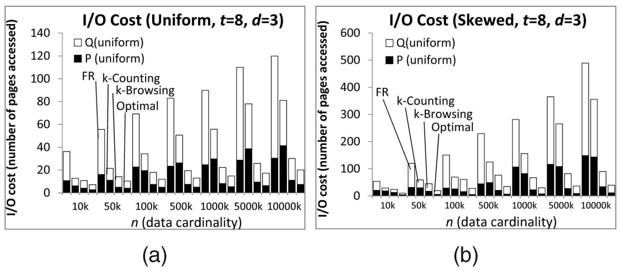


Fig. 24. The evaluation of data cardinality (n) on I/O cost. (a) Uniform data set. (b) Skewed data set.

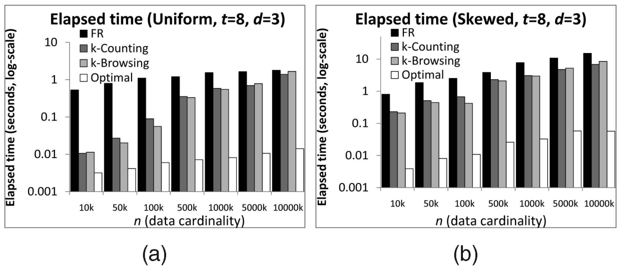
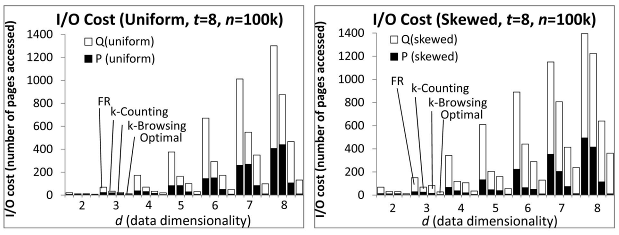
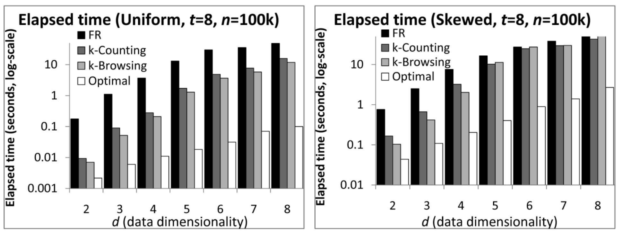


Fig. 25. The evaluation of data cardinality (n) on elapsed time. (a) Uniform data set. (b) Skewed data set.



(a) (b)

Fig. 26. The evaluation of data dimensionality (d) on I/O cost. (a) Uniform data sets. (b) Skewed data sets.



(a) (b)

Fig. 27. The evaluation of data dimensionality (d) on elapsed time. (a) Uniform data sets. (b) Skewed data sets.

past few years. In this paper, we have examined some unexplored aspects of RNN/RkNN and make the following contributions:

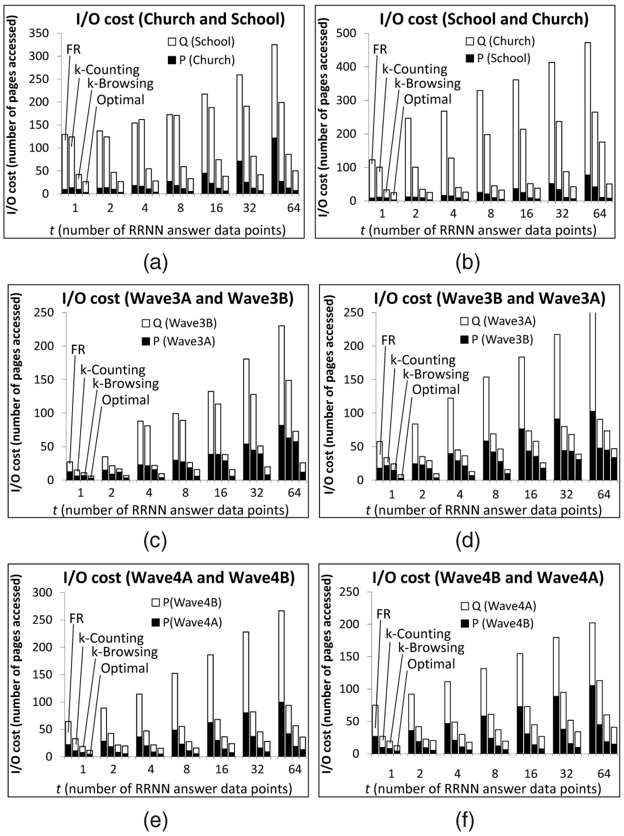


Fig. 28. The evaluation of real data sets on I/O cost. (a) Church ($\mathcal{P}$), School ($\mathcal{Q}$). (b) School ($\mathcal{P}$), Church ($\mathcal{Q}$). (c) Wave3A ($\mathcal{P}$), Wave3B ($\mathcal{Q}$). (d) Wave3B ($\mathcal{P}$), Wave3A ($\mathcal{Q}$). (e) Wave4A ($\mathcal{P}$), Wave4B ($\mathcal{Q}$). (f) Wave4B ($\mathcal{P}$), Wave4A ($\mathcal{Q}$).

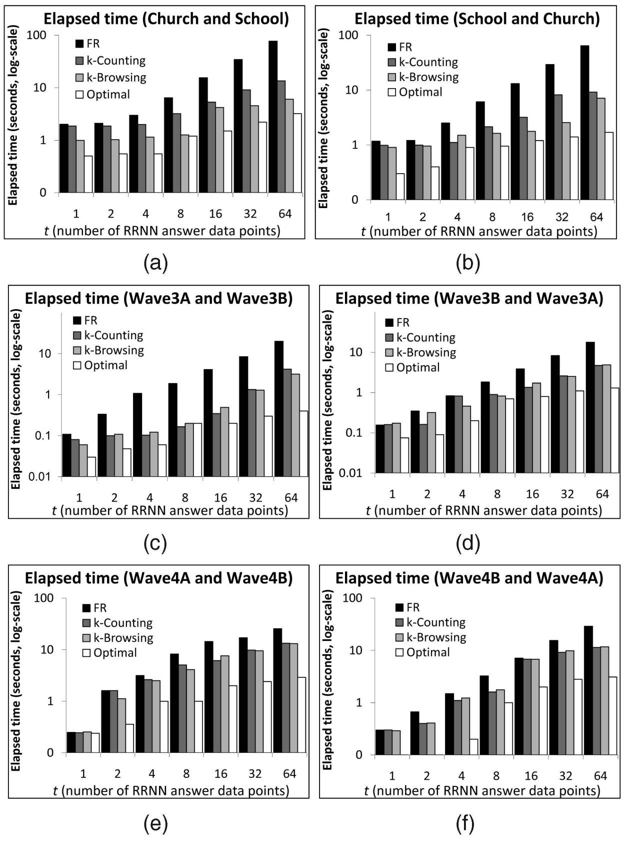


Fig. 29. The evaluation of real data sets on elapsed time. (a) Church ($\mathcal{P}$), School ($\mathcal{Q}$). (b) School ($\mathcal{P}$), Church ($\mathcal{Q}$). (c) Wave3A ($\mathcal{P}$), Wave3B ($\mathcal{Q}$). (d) Wave3B ($\mathcal{P}$), Wave3A ($\mathcal{Q}$). (e) Wave4A ($\mathcal{P}$), Wave4B ($\mathcal{Q}$). (f) Wave4B ($\mathcal{P}$), Wave4A ($\mathcal{Q}$).

- We present a new RNN variant, namely, 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 data sets, 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 proposals can achieve.
- Through extensive experiments on various synthetic and real data sets, 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.

Acknowledgments

Wang-Chien Lee and Ken C.K. Lee were supported in part by the US National Science Foundation under Grant IIS-0328881, Grant IIS-0534343, and Grant CNS-0626709.

References

[1] E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz, "Efficient Reverse k-Nearest Neighbor Search in Arbitrary Metric Spaces," *Proc. ACM SIGMOD '06,* pp. 515-526, June 2006.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R^*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD '90,* pp. 322-331, May 1990.

[3] R. Benetis, C.S. Jensen, G. Karciauskas, and S. Saltenis, "Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects," *Proc. Int'l Database Eng. and Applications Symp. (IDEAS '02),* pp. 44-53, July 2002.

[4] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Closest Pair Queries in Spatial Databases," *Proc. ACM SIGMOD '00,* pp. 189-200, May 2000.

[5] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. El Abbadi, "Constrained Nearest Neighbor Queries," *Proc. Seventh Int'l Symp. Advances in Spatial and Temporal Databases (SSTD '01),* pp. 257-278, July 2001.

[6] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD '84,* pp. 47-57, June 1984.

[7] G.R. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," *ACM Trans. Database System (TODS),* vol. 24, no. 2, pp. 265-318, 1999.

[8] F. Korn and S. Muthukrishnan, "Influence Sets Based on Reverse Nearest Neighbor Queries," *Proc. ACM SIGMOD '00,* pp. 201-212, May 2000.

[9] F. Korn, S. Muthukrishnan, and D. Srivastava, "Reverse Nearest Neighbor Aggregates over Data Streams," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02),* pp. 814-825, Aug. 2002.

[10] I. Lazaridis and S. Mehrotra, "Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure," *Proc. ACM SIGMOD '01,* pp. 401-412, May 2001.

[11] M.-L. Lee, W. Hsu, C.S. Jensen, B. Cui, and K.L. Teo, "Supporting Frequent Updates in R-Trees: A Bottom-Up Approach," *Proc. 29th Int'l Conf. Very Large Data Bases (VLDB '03),* pp. 608-619, Sept. 2003.

[12] K.-I. Lin, M. Nolen, and C. Yang, "Applying Bulk Insertion Techniques for Dynamic Reverse Nearest Neighbor Problems," *Proc. Seventh Int'l Database Eng. and Applications Symp. (IDEAS '03),* pp. 290-297, July 2003.

[13] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest Neighbor Queries," *Proc. ACM SIGMOD '95,* pp. 71-79, May 1995.

[14] A. Singh, H. Ferhatosmanoglu, and A. Saman Tosun, "High Dimensional Reverse Nearest Neighbor Queries," *Proc. ACM Int'l Conf. Information and Knowledge Management (CIKM '03),* pp. 91-98, Nov. 2003.

[15] I. Stanoi, D. Agrawal, and A. El Abbadi, "Reverse Nearest Neighbor Queries for Dynamic Databases," *Proc. ACM SIGMOD Workshop Research Issues in Data Mining and Knowledge Discovery (DMKD),* 2000.

[16] I. Stanoi, M. Riedewald, D. Agrawal, and A. El Abbadi, "Discovery of Influence Sets in Frequently Updated Databases," *Proc. 27th Int'l Conf. Very Large Data Bases (VLDB '01),* pp. 99-108, Sept. 2001.

[17] Y. Tao, D. Papadias, and X. Lian, "Reverse kNN Search in Arbitrary Dimensionality," *Proc. 30th Int'l Conf. Very Large Data Bases (VLDB '04),* pp. 744-755, Aug.-Sept. 2004.

[18] C. Xia, W. Hsu, and M.-L. Lee, "ERkNN: Efficient Reverse k-Nearest Neighbors Retrieval with Local kNN-Distance Estimation," *Proc. ACM Int'l Conf. Information and Knowledge Management (CIKM '05),* pp. 533-540, Oct.-Nov. 2005.

[19] T. Xia and D. Zhang, "Continuous Reverse Nearest Neighbor Monitoring," *Proc. 22nd Int'l Conf. Data Eng. (ICDE '06),* p. 77, Apr. 2006.

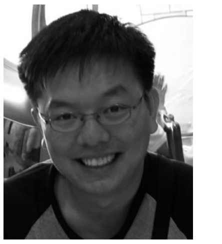
[20] T. Xia, D. Zhang, E. Kanoulas, and Y. Du, "On Computing Top-t Most Influential Spatial Sites," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05),* pp. 946-957, Aug.-Sept. 2005.

[21] X. Xiong and W.G. Aref, "R-Trees with Update Memos," *Proc. 22nd Int'l Conf. Data Eng. (ICDE '06),* p. 22, Apr. 2006.

[22] C. Yang and K.-I. Lin, "An Index Structure for Efficient Reverse Nearest Neighbor Queries," *Proc. 17th Int'l Conf. Data Eng. (ICDE '01),* pp. 485-492, Apr. 2001.

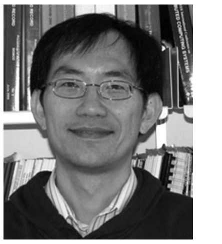
[23] M.L. Yiu and N. Mamoulis, "Reverse Nearest Neighbors Search in Ad-Hoc Subspaces," *Proc. 22nd Int'l Conf. Data Eng. (ICDE '06),* p. 76, Apr. 2006.

[24] M.L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao, "Reverse Nearest Neighbors in Large Graphs," *Proc. 21st Int'l Conf. Data Eng. (ICDE '05),* pp. 186-187, 2005.



Ken C.K. Lee received the BA and MPhil degrees in computing from Hong Kong Polytechnic University, Hong Kong. He is currently working toward the PhD degree in the Department of Computer Science and Engineering, Pennsylvania State University, University Park, where he is also a member of the PDA Group. His research interest includes spatial database, mobile and pervasive computing, and location-based services.

Baihua Zheng received the PhD degree in computer science from the Hong Kong University of Science and Technology, Hong Kong. She is currently an assistant professor in the School of Information Systems, Singapore Management University, Singapore. Her research interests include mobile and pervasive computing and spatial databases. She is a member of the IEEE and the ACM.



Wang-Chien Lee received the BS degree from the National Chiao Tung University, Hsinchu, Taiwan, R.O.C., the MS degree from the Indiana University, Bloomington, and the PhD degree from the Ohio State University, Columbus. He is an associate professor of computer science and engineering at Pennsylvania State University, University Park, where he also leads the Pervasive Data Access Research Group to perform cross-area research in database systems, pervasive/mobile computing, and networking. Prior to joining Pennsylvania State University, he was a principal member of the technical staff at Verizon/GTE Laboratories. He is particularly interested in developing data management techniques (including accessing, indexing, caching, aggregation, dissemination, and query processing) for supporting complex queries in a wide spectrum of networking and mobile environments such as peer-to-peer networks, mobile ad hoc networks, wireless sensor networks, and wireless broadcast systems. Meanwhile, he has worked on XML, security, information integration/retrieval, and object-oriented databases. His research has been supported by US National Science Foundation and industry grants. Most of his research results have been published in prestigious journals and conference proceedings in the fields of databases, mobile computing, and networking. He has served as a guest editor for several journal special issues on mobile database-related topics, including the *IEEE Transactions on Computer*, *IEEE Personal Communications Magazine*, *ACM MONET*, and *ACM WINET*. He was the founding program committee cochair for the International Conference on Mobile Data Management. He is a member of the IEEE and the ACM.