A Unified Method for Multi-dimensional NN, RNN, and Matching Queries

Tan Apaydin
Department of Computer Science & Engineering
The Ohio State University
apaydin@cse.ohio-state.edu

Hakan Ferhatosmanoglu
Department of Computer Science & Engineering
The Ohio State University
hakan@cse.ohio-state.edu

Amit Singh
J.P. Morgan Chase
amit.z.singh@jpmorgan.com

Ali Saman Tosun
Department of Computer Science
The University of Texas at San Antonio
tosun@cs.utsa.edu

Abstract

In this paper, we define a new type of query, i.e. *matching query*, which requires the answers of both NN and RNN queries. While NN is a well studied problem, RNN is relatively new. The earlier approaches to solve RNN problem mostly deal with two dimensional data. However, most of the applications such as document databases inherently involve solving RNN queries in high dimensions. We propose an approximate solution to answer RNN queries in any number of dimensions. Our approach is based on the strong correlation in practice between k-NN and RNN. It works in two phases. In the first phase, the k nearest neighbors of a query point are found. Out of these k points, most are eliminated by making use of bisector or distance optimization. In the next phase, the points obtained after applying optimization are analyzed using novel types of queries, called *Boolean/Count Range Query* (*BRQ/CRQ*) to find true RNNs. We also show methods to prevent false misses in this search process. Experimental results show that *BRQ*s and *CRQ*s are much more efficient than both NN and range queries, and can be effectively used to answer RNN queries. Performance is further enhanced by running multiple *BRQ*s simultaneously. Our approach is adaptable to any indexing structure including clustering and do not require any preprocessing. The proposed approach answers NN, RNN and matching queries in approximately the same number of I/O and time as running a k-NN query.

Keywords: Nearest Neighbor, Reverse Nearest Neighbor, Matching Query, Boolean Range Query, Algorithms, Performance.

1 Introduction

The goal of the Reverse Nearest Neighbor (RNN) problem is to find the set of points that have the query point as the nearest neighbor. Although RNN is a complement of Nearest Neighbor (NN) problem it is more complex than NN even for two dimensional data. The relationship between NN/RNN is not symmetric and the number of RNNs are not known in advance.

There are a wide range of applications for RNN queries. For example, a two-dimensional RNN query may ask the set of customers affected by the opening of a new store in order to inform the relevant customers. This query can be used to identify a good location which maximizes the number of potential customers. There are also many applications for multi-dimensional RNN, such as decision support systems, continuous referral systems, resource allocation, profile-based marketing, maintaining document repositories, etc. For example, a high dimensional RNN query may ask the subset of subscribers to a digital library who will find a newly added document most relevant. Each subscriber is described by a high dimensional feature vector consisting of her/his interests and background, and each document is similarly described by a corresponding feature vector describing its content. Many other applications are provided in the literature [14, 15, 22, 25]. Most of the concerned applications require an implementation of RNN for high dimensions, since the data in such applications (for example, documents) are typically represented by high dimensional vectors.

In addition to RNN query, a more general query, *RNN of order k*, is defined as the set of all points whose one of the k nearest neighbors is the query point. RNN problem is a special case of *RNN of order k* where $k = 1$. For example, one may want to inform the customers affected by the opening of a new store location: customers who have this store as one of their k closest. Our experiments with real data sets have shown that the traditional RNN problem returns a zero result in a significant number of queries (around $30\% - 40\%$). This percentage drops down to $5\% - 10\%$ for RNN of order 5. For many applications it is more useful to keep the query general, i.e., being second (or kth) closest to a data point.

We also define a novel query, i.e., *matching query* which has many applications of its own and can be answered efficiently using *RNN of order k* queries. A matching query asks all points in the i neighborhood (i.e., in the top i NNs) of the query which have the query point in their j neighborhood. It has many applications that involve the matching of the data such as profiles. For example, let's consider a job hunting website. An NN query can be posed, asking the top k companies that fit to a given user's qualifications. Or inversely, an RNN query can be posed to find the companies whose top choice is the

given user (which may return an empty set as the query result). However, a matching query would be more appropriate in this scenario, since both the company and the user must be mutually interested in each other. Similarly in a stock market database, a useful query is to ask companies that have similar stock price movements to each other, i.e., that are affected by each other. A big company will clearly affect the stocks of many other small ones, therefore an RNN query alone won't be that useful. Similarly an NN query by itself may not be very interesting either since a small company will easily be affected by the big ones. In order to capture the movements that are mutually correlated (such as two small companies doing business together), a matching query is a natural choice. In order to process a matching query, one needs to run both *k-NN* and *RNN of order k* queries. In this paper, we propose a unified algorithm to simultaneously process NN, RNN, and inherently also matching queries.

Our algorithm[1] works in two steps. The first step of the proposed technique runs a k-NN query on the data set and can employ available index structures on the database. Some of the k candidate points are eliminated from RNN set by using perpendicular bisector or distance calculation optimization (filtering step I). Perpendicular bisector optimization is experimentally found to be more efficient than distance calculation optimization in eliminating candidate points. In the second step, the remaining candidates after filtering I are analyzed by using a new kind of query *Boolean Range Query* (*BRQ*) to test for true RNN. *BRQ* is a special kind of range query, which terminates immediately if a single point is found or the whole edge of an MBR is inside the search region. For *RNN of order k*, we also define *Count Range Queries*, which again returns if there are less than k number of points inside the region. Experimental results show that *BRQ*s and *CRQs* are more efficient than range queries. Note that, *BRQ* is interested in neither the data objects nor the total number of data objects in the range. For most cases, it does not require any I/O operations and takes negligible amount of time. Therefore, we can answer RNN and matching queries in approximately the same number of I/O and time as running a k-NN query.

The problems of multi-dimensional indexing [10, 11, 18, 19], [4, 17], [6, 7, 24] and NN query processing [12,20,21] have been extensively studied in the literature. The proposed technique in this paper can make use of any of the existing access structures and nearest neighbor algorithms. No additional data structures, or additional space is needed. In our paper, we implement RNN algorithm on an R-tree and clustering based schemes. In contrast to earlier approaches, the proposed technique does not require any additional preprocessing of the data (such as precomputing and storing NN information for each point), and is not limited to two dimensions.

Using our proposed technique, it is possible to answer RNN without any false misses. False misses can be prevented by either using a cone based approach, which we discuss in subsequent sections, or by choosing a high value of starting k for running k-NN queries.

[1] An earlier version of this paper appeared in CIKM'03.

For the latter case, experimental results show that the time for running a k-NN query does not vary much with k. Therefore, we can choose a high value of starting k so as to avoid any false misses.

The rest of the paper is organized as follows. We first discuss the previous work on RNN based queries in Section 2. Section 3 discusses the challenges involved in answering high dimensional RNN and provides some interesting observations to serve as a theoretical foundation for the RNN search in high dimensions. Section 3.3 deals with how false misses can be avoided. Section 4 presents the proposed algorithm. In Section 5, the results of the experiments are given with the implementation of our algorithm. We conclude our work in Section 6.

2 Discussion of Related Work

The earlier approaches to find the RNN mostly deal with two dimensions. The problem of RNN was first introduced by Korn et al. [14] which used an R-tree based index structure. For a static database, they propose to use a special R-tree, called RNN-tree, to answer RNN queries. For a dynamic database (where updates frequently occur) two separate trees (NN-tree and RNN-tree) are used. RNN-tree stores the nearest neighbor (NN) of each data object and NN-tree stores a set of objects consisting of the data points and their corresponding nearest neighbor. RNN queries are then answered by point enclosure queries over the RNN-tree. In summary, for each point in the database the corresponding NN is determined and a circle is generated with the point as the center and its distance to NN as the radius. Then for a given query point q, all the circles that contain q is determined to answer RNN query. This approach is inefficient especially for dynamic databases because NN-tree and RNN-tree structures have to be modified each time an update occurs either by insertion or deletion. Yang et al. [25] improved the solution of Korn et al. [14] by introducing a single indexing structure (Rdnn-tree) instead of multiple indices, which makes it possible to answer both RNN and NN queries using a single tree. Rdnn-tree (R-tree containing Distance of Nearest Neighbors) differs from the standard R-tree by storing extra information about NNs of the points for each node, therefore requires prior computation of NN for each point. In addition, Maheshwari et al. [3] proposed static and dynamic data structures for answering RNN queries but it is also restricted to two dimensions.

All of the above mentioned techniques are based on preprocessing of the data. So, they cannot deal efficiently with dynamic databases where there are frequent updates. To solve the problem for dynamic databases, Stanoi et al. [22] used a geometric approach to answer RNN queries in two dimensions. Their approach does not require any preprocessing. RNN queries are answered by executing multiple conditional nearest neighbor queries. The algorithm makes use of the fact that in two dimensions there are at most six RNNs. The plane is divided into six equal regions around the query point. If there are two RNNs

in a region then they will be on the lines dividing the regions, otherwise each region would have at most one RNN. The approach involves two steps. In the first step, NNs in each of the regions are found and in the next step each point is checked for RNN. In addition, Tao et al. [23] proposed a two step algorithm that give exact solutions. Their approach is based on the perpendicular bisector between the query point q and an arbitrary data point p. The rationale is that any data point in the half plane that contains p cannot be an RNN of q because it is closer to p than q. They truncate the search space using the bisectors introduced by new arbitrary points. The performance of the node access of their approach heavily depends on the relative positions of the candidates and the resulting half planes. Furthermore, their search space truncation is computationally expensive in high dimensions and that is the reason they only present experimental results up to 5 dimensions. In addition, their algorithms are specific to the RNN problem, i.e., the NN and matching query problems cannot be handled by their technique.

3 High Dimensional RNN

Section 3.1 talks about some of the challenges in the high dimensional RNN problem. Section 3.2 states some interesting observations which will allow us to develop an efficient solution for the RNN problem even in high dimensions. Section 3.3 deals with how false misses can be avoided.

3.1 Challenges Involved in High Dimensions

Let's analyze the possible number of RNNs of a point in a given dimensionality. This problem is analogous to the sphere packing problem or the kissing number problem where one has to find the greatest number of equivalent hyperspheres in d dimensions, all of same size, that can be arranged around another hypersphere without any intersections [8]. In two dimensions, the answer is six, i.e., six circles around another circle. This property was used in [22] to develop a two-dimensional RNN search algorithm. In three dimensions this number becomes 12 and it increases exponentially with increase in dimensionality. The exact values of the kissing number are known for d=1 *to* 9 and for d=24. Exact kissing number in high dimensions is an open research problem, however a range for the actual solution is given in Figure 1. Lower bound denotes arrangements that can be achieved and upper bound gives the theoretical upper bound [8]. Also note that, in other distance metrics such as L_∞, the cardinality of RNN(q) is at most $3^d - 1$ in d dimensions.

The exponential increase in the number of RNNs with the increase in dimensions makes the geometric algorithm in [22] practically impossible for higher dimensions because the number of regions would be very high (equal to the number of possible RNNs) and we have to search each region to find whether an RNN exists or not. So, any similar approach would be infeasible for high dimensions. The other two approaches in the litera-

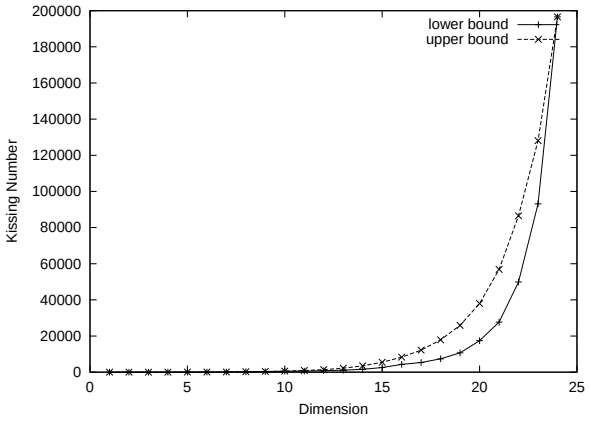


Figure 1: Kissing Number in High Dimensions

ture [14, 25] used prior NN distance information for each point to build the specific index structures. The pruning power, based on the precomputed NN distance will degrade significantly as the dimensionality increases because of the typical high overlap problem. For example, the NN circles (and the corresponding rectangular approximations) used in [14] will highly overlap for high dimensions, and the NN distance used for pruning in [25] will intersect most of the other MBRs and cause them not to be pruned during the RNN query.

3.2 Observations in High Dimensions

Number of RNNs : Although the number of RNNs of a point in a d-dimensional vector space exponentially depends on d, the expected number of RNNs in a data set is independent of dimensionality based on the following observation. Consider a set S of points in d-dimensional space and RNN queries $q \in S$ to the set S-{q}. For each point, take that point out and run the RNN query on the remaining set. The number of RNNs for q is the number of points which has q as NN by definition of RNN. If each node in the set has a unique NN then number of RNNs for all queries is equal to the number of points and the average is 1. If some points have 2 NNs, then we add 2 for those points and divide the total number of NNs by set size to find average number of RNNs. Although this type of queries does not capture the worst case performance, it takes into account several factors of the data set. Queries depend on density of data, more queries are run on dense regions. In most practical data sets the number of RNNs of a point will be quite low compared to the theoretical maximum. The average number of RNNs being 1 has important implications for high dimensional RNN queries in databases. The average number is independent of the dimensions. This suggests huge potential for improvement in high dimensions using approximate schemes.

We experimentally verified the result using several real-life data sets that are described in Section 5. Figure 2 shows the plot of percentage of queries versus the number of RNNs.

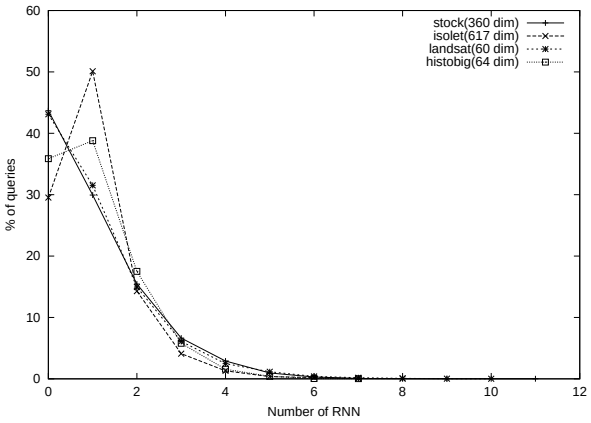


Figure 2: Percentage of Queries versus Number of RNN

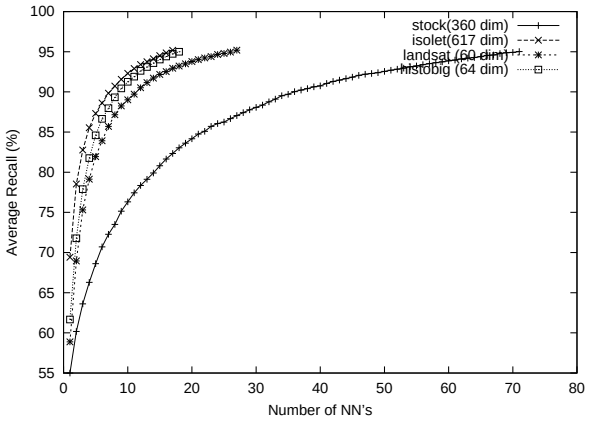


Figure 3: Plot showing average recall as a function of Number of NNs

Although the number of RNNs theoretically increases exponentially with the increase in dimension we observed that the number of RNNs is usually small (0-4) .

Recall : The second important observation for real high dimensional data sets is that although NN and RNN are not symmetric, they are strongly correlated. Efficient processing of NN queries has been extensively studied in the past, and those techniques can be employed for approximate searching of RNNs, if there is really a correlation in practice between the two. For this purpose, we performed experiments by executing k-NN queries to answer RNN queries approximately. Figure 3 shows the variation of average recall with number of NNs. Average recall is defined as the percentage of queries where RNN is contained in the k-nearest neighbor set. We observe that with only finding a small number of NNs, i.e., with low values of k, we can achieve very high recall, e.g, 90% success with $k = 10$. The average recall will depend on the value of k. Higher the value of k, higher will be the recall. In other words, a large value of k decreases the false misses.

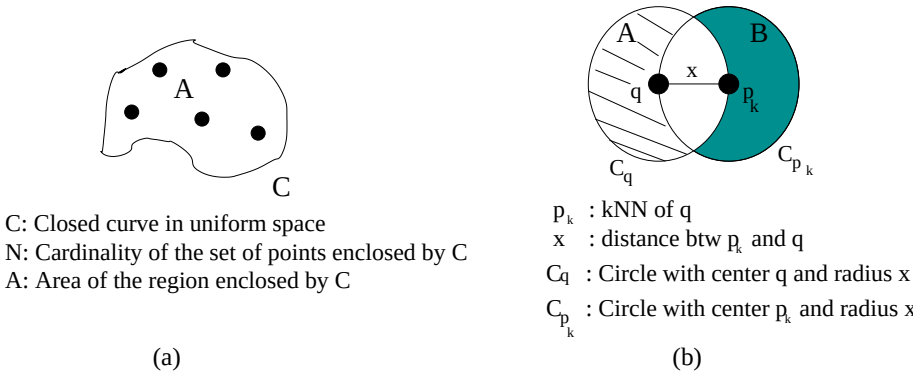


Figure 4: Probability of a k-NN to be an RNN

Probability of a k-NN to be an RNN : Another important observation is that the probability of *a k-NN to be an RNN* degrades as k increases. In order to focus on this, consider a uniformly distributed data space and a subset of that (i.e., C) as in Figure 4(a). For uniform distribution, the probability mass function (pmf), i.e. $P_N(n)$, of the random variable **N** (defined in Figure 4(a)) is a Poisson function given by:

$$P_{\mathbf{N}}(n) = \frac{(\sigma A)^n e^{-\sigma A}}{n!} \tag{1}$$

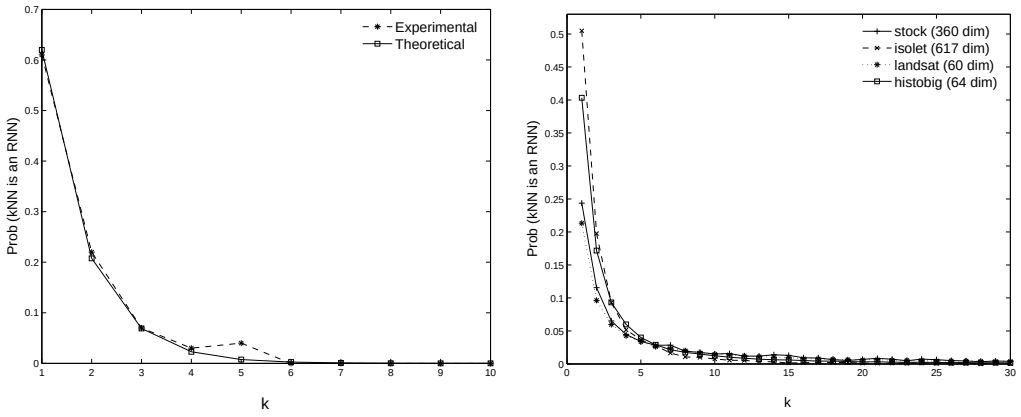
where σ denotes the density (data points per unit area). Thus, the probability that there are no points within C is given by:

$$Prob\{\mathbf{N} = 0\} = P_{\mathbf{N}}(0) = e^{-\sigma A} \tag{2}$$

We now analyze the probability of a k-NN to be an RNN. Assuming p_k to be k-NN of q, we know that there are $k-1$ points in C_q in Figure 4(b). For p_k to be an RNN, none of these points can be in the intersection region of C_q and C_{p_k}, i.e., they must be in region A. Furthermore, there can be no points in region B, otherwise the nearest neighbor of p_k would not be q. Therefore, the probability of p_k to be an RNN is given by:

$$\begin{aligned}
\text{Prob\{ k-NN is an RNN \}} &= \text{Prob\{ First (k-1) NNs are in region A \} . Prob\{No points in region B\}} \\
&= \left[\frac{Area(A)}{Area(C_q)}\right]^{k-1} e^{-\sigma Area(B)} \\
&= \left[\frac{(\frac{\pi}{3} + \frac{\sqrt{3}}{2})x^2}{\pi x^2}\right]^{k-1} e^{-\sigma(\frac{\pi}{3}+\frac{\sqrt{3}}{2})x^2} \\
&= \left(\frac{1}{3} + \frac{\sqrt{3}}{2\pi}\right)^{k-1} e^{-\sigma(\frac{\pi}{3}+\frac{\sqrt{3}}{2})x^2}
\end{aligned} \tag{3}$$

where x denotes the distance between q and p_k.



(a) Theoretical and experimental results based on uniform distribution

(b) Real data results

Figure 5: Probability that k-NN is an RNN

For a uniform distribution, the *pdf* of x is given by:

$$P_{\mathbf{X}}(x) = \frac{2(\pi\sigma)^k x^{2k-1} e^{-\pi\sigma x^2}}{(k-1)!} \tag{4}$$

The expected value of x is:

$$\overline{x} = E[x] = \int_0^\infty \frac{2(\pi\sigma)^k x^{2k} e^{-\pi\sigma x^2}}{(k-1)!} dx \tag{5}$$

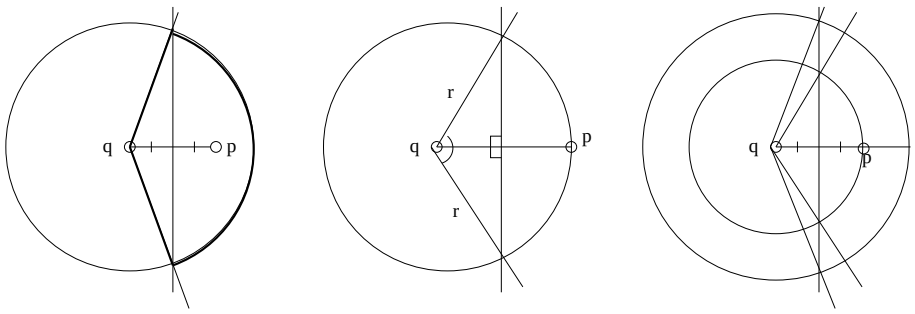
For each value of k, Equation 5 is numerically computed and the result is substituted into Equation 3. Finally, *the probability of a k-NN to be an RNN* is plotted in Figure 5(a) as a function of k for uniform distribution. The Figure illustrates that the probability decreases to almost zero after $k = 6$. Furthermore, we experimentally verified the above theoretical results by using real data sets, which are shown in Figure 5(b). The RNNs of a query point are very likely to be found in the k-NN set for a reasonable value of k. Thus, it is reasonable find the k-NNs of the query point first and then look for the RNNs within the k-NNs.

3.3 Prevention of False Misses

In this subsection, we will discuss how false misses can be avoided either by using a cone based approach or high starting value of k for running k-NN queries.

Cone Based Approach

Using the perpendicular bisectors between the query point and a subset of k points, we can find out whether the set of candidates we have suffice to prevent false misses or not.



(a) Example of a cone (b) Cone for a single point (c) Wider Cone

Figure 6: Definition of a cone

Definition: Given a circle C with center q, radius r and a point p, a cone is the region between the center of the circle and intersection points of the bisector between q and p (refer to Figure 6(a)).

Theorem: Let the set $P = \{p_1, ..., p_k\}$ be the result of k-NNs in increasing order of distance from the point q. Consider the circle C with the center q and radius $d(q, p_k)$. No RNN is missed if the union of the cones covers the whole circle.

Proof: Assume that the union of the cones covers the circle. A point t, not in the set S, would be outside the circle. Since cones are formed using intersections of bisectors, t would be on the other side of a bisector and be closer to one of the points in S, therefore t is not an RNN of q.

lemma 1 *A single point (NN) covers an angle of $\frac{2\pi}{3}$.*

lemma 2 *As the radius of a circle increases, the cone for a fixed point becomes wider. Wider cone is given in Figure 6(c).*

lemma 3 *Cone of a point always covers an angle $< \pi$.*

lemma 4 *In 2-dimensions at least 3 points are needed to cover the circle using cones.*

Proof: Since each point covers $< \pi$, at least 3 points are needed to cover 2π.

In order to decide if the cones cover the circle, we represent each cone as intervals $[l, h]$ where l and h are the angles made with the x axis. Cone is specified in counter-clockwise order. For example, the cone in Figure 6(a) is represented by the interval $[\frac{10\pi}{6}\frac{2\pi}{6}]$. The intervals cover the circle if the union of the intervals is $[0, 2\pi]$.

Choosing Larger k

The false misses can also be avoided by using a high value of starting k. Figure 7 shows the variation of time with number of nearest neighbors for isolet and landsat data. As we can see from the figure there is not much difference in the time with the increase in the value of k. So, we can start with a high value of k so as to avoid false misses. The starting value of k to avoid any false misses can be found out by running experiments similar to Figure 3.

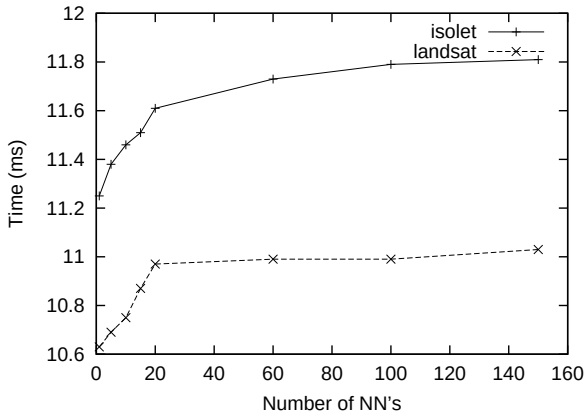


Figure 7: Average time (ms) versus Number of NNs for isolet and landsat data

4 RNN Algorithm

Figure 8 summarizes our general approach for answering RNN queries. The algorithm works in two steps: In the first step, a k-NN search is performed returning the k closest points to the query and in the next step these k objects are efficiently analyzed to answer RNN. The value of k that should be used in the first step depends on the data set and can be estimated by the experiments similar to Figure 3. Based on the values of average recall we can choose the starting value of k. For avoiding any false misses, the value of k can be chosen by applying the cone approach or by running k-NN query for high value of k.

The following subsections explain the filtering step I and II in detail with necessary optimizations that can be used.

Input: Query object q, query parameter k

1−) To find candidates for RNN(q), run kNN query on the index structure

2−) Filtering I: Eliminate candidates by using distance or perpendicular bisector optimization

3−) Filtering II: To determine actual RNN, run Boolean Range Query on the remaining candidates

Result: Final RNN

Figure 8: Proposed Algorithm for RNN

4.1 Filtering step I

After the first step of the RNN algorithm we have k nearest neighbors of a query point. However, some of them can be eliminated from RNN set by either local distance calculations (optimization 1) performed within the k, i.e., the candidate set, objects or using the property of perpendicular bisector (optimization 2). We refer to this step as *filtering step I*.

Optimization I : It is based on the fact that a point that does not have the query point as the nearest neighbor among the candidate set can never be the RNN of the query point in the whole data set. Therefore a point that has another candidate as its closest in those k-NN points is eliminated from the search (note that only k objects are involved in this test). Our experiments show that this step can efficiently filter out a significant number of candidates. For example, for 60-NN queries on our stock market time-series data, on the average 50% of the 60 candidates are eliminated. In this step, some of the distance calculations (out of a possible $\binom{k}{2}$) can be avoided by using an optimization based on comparison of the already computed distances. This is formalized by the following lemma.

lemma 5 *Let x and y be two nearest neighbors of the query point q and $d(x, y)$ denotes the distance between data points x and y. If $d(x, y) \leq d(x, q)$ and $d(x, q) \leq d(y, q)$, then it follows that both x and y cannot be RNN.*

Optimization II : It makes use of the property of perpendicular bisector in eliminating candidate points. Let q be the point of interest. Let $P = \{p_1, ..., p_k\}$ be the k points returned by the k-NN step. Pick any point in P, and construct the perpendicular bisector between q and this point as shown in Figure 9. Any point r that is on the same side of bisector as p can not be an RNN of q since it is closer to p than q. The most logical choice for p is the NN of q. Since the bisector will be closer to q, the likelihood of eliminating nodes is increased.

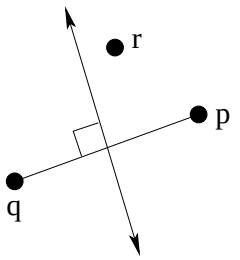


Figure 9: Bisector Optimization

In order to figure out whether a point is on one side of the hyperplane or not, let $a_1x_1 + a_2x_2 + ... + a_dx_d = b$ be the equation of the hyperplane in d dimensions. We just need to find the equation of the hyperplane (bisector), evaluate the equation at a point and compare the result with b to figure out which side it is on.

Table 1: Comparison of optimization I with II (# of eliminated candidate points)

Data set	Optimization II			Optimization I
	1st pass	(1st+2nd) pass	(1st+2nd+3rd) pass	
Landsat	31.72	45	58.95	15.77
Histo	46.87	74.13	74.16	29.73
Stock	62.05	80.98	81.09	51.1
Isolet	43.55	72.25	72.38	19.69

It is important to note that Tao et al. [23] also utilized the perpendicular bisectors, which is a well known technique in computational geometry. However, they apply the bisectors to the entire data set and that makes their algorithms inefficient for higher dimensions. On the other hand, we use bisectors for only the k number of candidate points and that makes our filtering step computationally inexpensive.

We ran experiments on different data sets to eliminate some of the candidate points using optimization II. In running our experiments, we first took a query point (q) and the first nearest neighbor of query point (p) and eliminate points by calculating the position of the point relative to the perpendicular bisector. We called this step as first pass. In the second pass, we took query point as q and nearest neighbor of q in the reduced set as point p and followed the same approach. We conducted our experiments for only three passes because there is not much gain after three passes. The number of points that can be eliminated will increase with the number of passes but it will result in higher processing cost. Table 1 shows the corresponding results with comparison of optimization II with optimization I. Here, we have taken 60 nearest neighbors i.e. candidate points for a given query point.

It is evident from the table that optimization II clearly outperforms optimization I even for a single pass. There is not much improvement after second pass. The time complexity for optimization I is $O(k^2)$ where k is the number of candidate points. On the other hand, time complexity for optimization II is $O(kd)$ where d is the number of dimensions. Timing comparison results depend on the number of passes involved in optimization II. In our subsequent experiments we will only consider two passes of optimization II. Overall, optimization II is far better than optimization I because it eliminates more candidate points which are further analyzed in filtering step II. Another advantage of optimization II is that even if we start with a high value of k we can eliminate a lot of candidate points in one or two passes. This is very important if user wants to avoid any false misses or wants to calculate exact RNN.

4.2 Boolean Range Queries (Filtering step II)

After filtering step I, we have a set of candidate data points, each of which lies in the hyperplane containing the query point q. For a point to be an RNN, we need to verify

```
Procedure BooleanRangeQuery(Noden, pointq, radius)
BEGIN
    Input:
        Node n to start the search
        Point q is query point
        radius is equal to distance between
        query point and candidate set point
    Output:
        True ⟹ At least one point inside search region
        False ⟹ No point inside the search region
    If n is a leaf node do:
        check if there exists a point p such that
        dist(p, q) ≤ radius then return TRUE;
    If n is an internal node, then for each branch do:
        If any node(MBR) intersects such that at least
        one edge of MBR is inside search region, i.e.,
        minmaxdist ≤ radius then return TRUE;
        else
        Sort the intersecting MBRs wrt. some criterion
        such as mindist and traverse the MBRs wrt. it.
        If any of the MBRs has a point then return TRUE;
        else return FALSE;
END
```

Figure 10: Boolean Range Query

that the query point is the actual NN considering the whole data set. Two different approaches can be followed here. One crude approach to solve the above problem is to find the global NN of each point and check whether it is the query point. If this is the case, then that point is an RNN otherwise it is not an RNN. Another approach, which we refer to as *filtering step II* is running a new kind of query called the *boolean range query.*

Definition 1 *A boolean range query* $\mathcal{BRQ}(q, r)$ *returns true if the set* $\{t \in S | d(q, t) < r\}$ *is not empty and returns false otherwise.*

Boolean range query is a special kind of range query which will return either true or false depending on whether there is any point inside the given range or not. Boolean Range Queries can be naturally handled more efficiently than range queries and this advantage is exploited in the proposed scheme. For each candidate point, we define a circular range with the candidate point as the center and the distance to the query point as the radius. If this boolean range query returns false then the point is RNN, otherwise point is not an RNN. The range queries typically are more efficient than NN queries [16], and boolean range queries are much more efficient than range queries. Our explanation here will be based on an R-tree based index structure, but the general algorithm is not restricted to R-trees and it can be easily applied to other indexing structures. The pseudocode for *Boolean Range Query* is given in Figure 10.

Boolean Range Query versus Range Query

In a range query, typically multiple MBRs intersect the query region. When the number of dimensions is high, the number of MBRs intersecting the query region is also

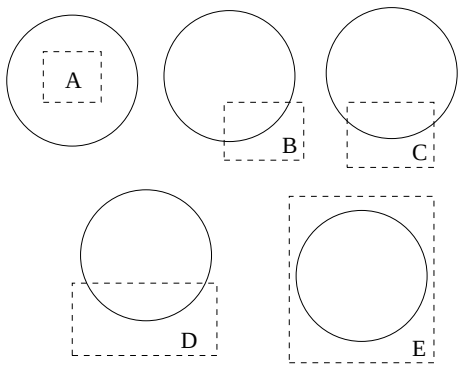


Figure 11: Cases for boolean range query intersecting with a circular region

high. [2]

For range queries we have to access all the MBRs that intersect with the search region. The main strength of boolean range query over traditional range query is that even if multiple MBRs intersect a search region we do not need to access all of the MBRs.

Figure 11 shows five possible cases where a circular search region can overlap partially or fully for range queries. Search region is denoted by circle and MBRs are denoted by *A*, *B*, *C*, *D* and *E*.

1. MBR *A*: fully contained in search region. This means that search region has at least one point.

2. MBR *B*: only one vertex of B is in search region.

3. MBR *C*: two of *C*'s vertices are in search region. This means that an entire edge of MBR is in search region, which guarantees that at least one data point is in search region.

4. MBR *D*: no vertices of *D* are in search region.

5. MBR *E*: Search region is fully contained in the MBR *E*.

For a boolean range query, if we can guarantee that at least one edge of MBR is inside the search region then we do not need any page access (case A and C in Figure 11). This can be easily checked by calculating the *minmaxdist*, which is defined as minimum of the maximum possible distances between a point (here candidate point or center of search region) to the face of intersecting MBR [20]. If *minmaxdist* is less than or equal to the radius of the search region than there is at least a point in the intersecting MBR so

[2]However, it is beneficial to note that the number of MBRs accessed in a high dimensional query is not as pessimistic as the theoretical results suggest, mainly because of the correlations of dimensions in real data. We will not discuss those results in detail here, since it is an orthogonal issue to the purpose of this paper.

we do not need any page access. For example, we found that for isolet data set, 35% of intersecting MBRs have this property. For other cases B, D, and E we cannot say whether there is a point contained both in search region and MBR. So, traversal of the corresponding branch may be necessary to answer the query if no relevant point is found previously.

When a range query intersects a number of MBRs, it has to retrieve all of them and make sure that there is a data point in them. The answer of the range query is all the points that lie in the range. Even if the query asks only the average of data objects in a given range, i.e. aggregation queries, it needs to read all the data points within the MBRs and check whether they lie within the given range. However a boolean range query can be answered without any page accesses as described above or with minimal number of page accesses. If a single relevant point is found in one of the MBRs, then we do not have to look for other points and we can safely say that the corresponding point is not an RNN. So, for the case of multiple MBRs intersecting the query region, a decision has to be made to maximize the chance of finding a point in the MBR so that the query can be answered with minimal number of MBRs accesses. In choosing the MBRs, there could be a number of possible choices:

1. Sort MBRs wrt. overlap with search region and then choose MBR with maximum overlap.
2. Sort MBRs wrt. *mindist* and then choose MBR with minimum *mindist*.
3. Choose randomly.

We tried experimentally all the above three possibilities and found that choice 1 and 2 are comparable and both are better than choice 3. Most of the time the set of MBRs that are retrieved to the memory in the first phase of the algorithm is enough to guarantee that a point in a candidate set is not an RNN. In this case, no additional I/O is needed in the second phase. Therefore, almost with no additional overhead on the k-NN algorithm we can simply answer RNN queries. This result is justified by the experiments in Section 5.

Multiple Boolean Range Queries

In the filtering step II we need to run multiple boolean range queries since there are multiple points that remain after the filtering step I. Since these are candidates that are very close to each other, multiple query optimization becomes very natural and effective. Multiple queries are defined as the set of queries issued simultaneously as opposed to single queries which are issued independently. There have been a significant amount of research that investigated this approach for other kind of queries [5]. In our algorithm, a boolean range query needs to be executed for each point in the candidate set. Since the radius of the queries are expected to be very close to each other (because they are all in the k-NN of the query), executing multiple queries simultaneously reduces the I/O

cost significantly. First we read a single page for the whole set of queries. The criteria for deciding which page to access first is to retrieve the page that has the most number of intersections with the queries. The performance gain of this approach is discussed in the experimental section.

4.3 Clustering

We also adapted our approach on a clustering based index which is more effective than R-trees on higher dimensions [9]. We used a modified version of k-means clustering [1]. The basic RNN algorithm for clustering is as follows:

1. Find the cluster with representative, i.e., mean, closest to the query point.
2. Depending on the number of data points in the cluster find the k-NN in that cluster.
3. Eliminate some of the candidates by using the approach we discussed earlier (Filtering Step I).
4. Run boolean range query on the remaining points (Filtering step II).

Boolean range queries can be implemented in the same way as discussed before and multiple boolean range queries can retrieve the clusters again in the order of the number of intersected queries. As usual, if a cluster has a point in the given range then we can stop immediately and return TRUE for the corresponding boolean range query, i.e., the candidate point is not an RNN, otherwise the algorithm can choose the next cluster which is intersected by most queries and so on. As we will see in the section 5.5, most of the time a single cluster is enough to answer RNN queries.

5 Experimental Results

We carried out experiments using several real data sets to explore the RNN problem in high dimensions. We also compared the performance of boolean range query with range query, and also examined the performance of multiple boolean range queries. For performance comparisons we used SVD reduced data sets and chose 500 query points at random from it.

The data sets used in our experiments are real data sets from different application domains (please contact author for the data sets). We used the following data sets:

- Stock Time series, is a time series data set which contains 360 days (dimensions) stock price movement of 6500 companies (8.93 MB).
- Isolet (Isolated letter speech recognition) data consists of speech samples of 26 English alphabets with 617 dimensions and 7797 samples (18.35 MB).

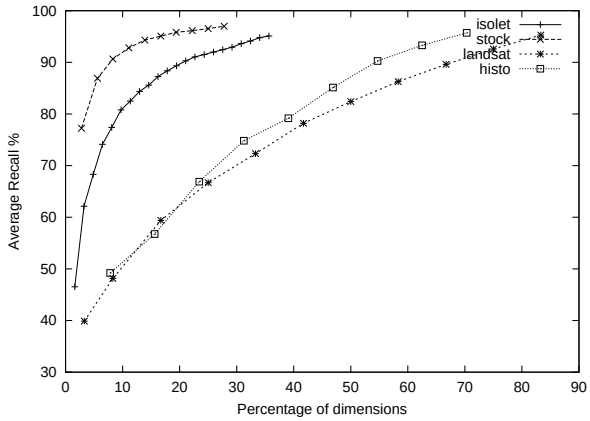


Figure 12: Plot showing average recall as a function of % of dimensions

- Satellite Image Texture (Landsat), is of size 21,000 with 60-dimensional vectors representing texture features of Landsat images (4.85 MB).
- Color Histogram, is a 64 dimensional data set of size 12,000. The vectors represent color histogram computed from a commercial CD-ROM (2.94 MB).

The code for our experiments was implemented in C++ on a Solaris UNIX workstation. We modified the R-tree code provided by GIST [2] for performance comparisons. The page size was set to 8K.

5.1 Dimensionality Reduction for RNN

When the dimensionality of data is high, transformation functions such as SVD [13] are generally used to reduce the dimensionality. SVD is widely and successfully used for traditional range and NN queries. We performed experiments to investigate the effectiveness of transformation functions such as SVD for RNN problem. Figure 12 shows the comparison of the various data sets for different percentage of dimensions. These results show that SVD is effective to reduce the dimensionality also in the domain of RNN queries. For example, using only 7% of the SVD dimensions an average recall of 90% is achieved for stock price time-series data.

5.2 Boolean Range Query and Multiple Query Optimization

In this section we compare the performance of range query vs. boolean range query, a single boolean range query vs. multiple boolean range query. We calculated the average I/O cost, i.e., the number of page accesses divided by the number of queries.

Table 2 shows the I/O performance of the boolean range query compared to range query. There is a significant improvement in the performance. Here we only show results on range query and boolean range query corresponding to the region for the first nearest

Table 2: Comparison of Boolean Range Query with Range Query (# of page accesses)

Dim.	5		10		15	
	RQ	BRQ	RQ	BRQ	RQ	BRQ
Landsat	60.57	27.18	188.29	19.15	299.87	9.45
Histo	7.56	3.94	38.21	8.55	91.8	13.28
Stock	46.18	20.55	91.63	17.43	125.22	14.20
Isolet	88.86	36.49	160.96	3.11	225.79	1.07

Table 3: Comparison of Single vs Multiple Boolean Range queries (# of page accesses)

Dim.	5		10		15	
	Single	Mult	Single	Mult	Single	Mult
Landsat	351.8	44.15	158.12	27.03	111.21	14.53
Histo	98.26	9.23	148.56	17.39	186.38	24.14
Stock	179.64	25.87	207.4	22.11	178.26	21.08
Isolet	300.83	16.19	64.68	5.84	60.55	1.17

neighbor only. When we perform the same experiments for more neighbors, the difference in page accesses of the two is very high, mainly because as the search region increases range query retrieves more pages.

Table 3 shows the I/O performance of multiple boolean range queries compared to single boolean range queries. The values here refer to the total number of page accesses for 60 boolean range queries. Since all the queries are close to each other there is a high overlap between query regions, and all queries access almost the same set of pages. We observed a speedup of 7 to 60 over the single query case.

5.3 Overall Performance of Proposed Technique

In our RNN search we find RNN in two phases. In the first phase, we find k-NN of a query point and in the second phase we run boolean range query on reduced set. Most of the time both phases use the same pages. Therefore, if we use the pages that are retrieved in the first phase, there will be a significant improvement in performance for the second phase. So we ran experiments to measure the costs for NN queries, multiple boolean range queries, and the combined NN-BRQ. Table 4 shows the corresponding results. From the table, it is evident that the cost of combined queries is essentially the same as that of k-NN query and is small compared to separate k-NN and BRQ. This shows that the proposed approach can effectively and efficiently answer RNN query as well as k-NN queries. Also, note here that in this table we are not using any type of filtering I.

Analysis of each step

Table 4: Performance of combined NN and Boolean range queries

Dim.	5			10		
	k-NN	Combined	Separate	k-NN	Combined	Separate
Landsat	107.91	111.49	148.72	257.71	259.1	284.74
Histo	20.47	23.44	29.70	78.28	79.37	95.66
Stock	58.58	58.84	84.46	102.6	102.86	124.72
Isolet	102.69	102.69	146.84	171.24	171.29	177.1

The time complexity of our algorithm is comprised of three factors: 1) Time to calculate k-NN for a given query point. 2) Time taken for filtering step I. 3) Time to run boolean range queries (filtering step II).

To analyze the effect of each step, we looked at five possible scenarios of answering RNN queries.

1. k-NN + multiple NN for k (no filtering is used here and we are running NN queries on k points).

2. k-NN + multiple NN with filter I (here we are reducing the number of multiple NN queries from k to m by using distance optimization I).

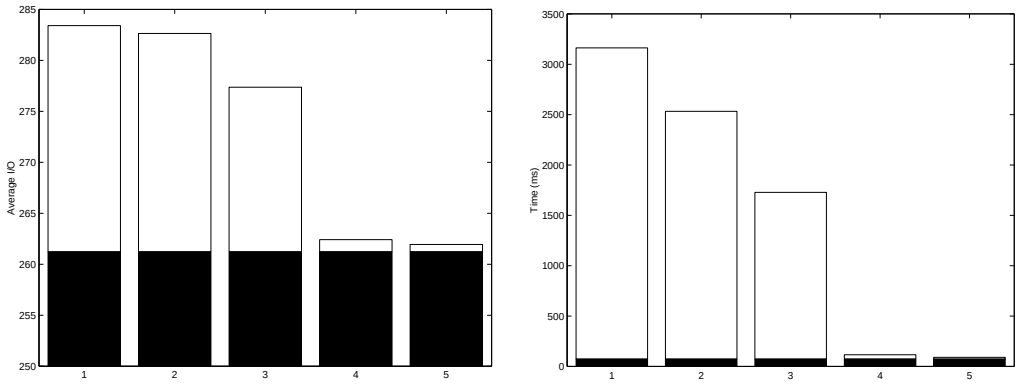
3. k-NN + multiple NN with filter I (here we are reducing the number of multiple NN queries from k to m by using bisector optimization II).

4. k-NN + filter I (optimization I) + filter II.

5. k-NN + filter I (optimization II) + filter II (the overall algorithm).

In each of the above cases first step is same, i.e., running a k-NN query. Therefore all scenarios mentioned above are taking advantage of the fact that most of the time the pages that are retrieved in the first step are used in the second step. We ran our experiments on SVD reduced data sets with different dimensions but because of the lack of space we are showing the results for landsat data with 10 dimensions only.

Figure 13(a) shows the average I/O for landsat data and Figure 13(b) shows the time for a single query for landsat data. The labels on x-axis refer to the five considered scenarios. Note here that timing results are real and shows the time taken for executing a single RNN query. However, average I/O takes advantage of the fact that the points that are retrieved in the first step, i.e., k-NN search does not need to be accessed again. Lower bar in each of these bar charts correspond to first step i.e. running a k-NN query. The number of page accesses and the time required by the overall algorithm is minimum of all the considered choices. Moreover, the number of I/O's required is almost same as required by k-NN query, so we can effectively and efficiently answer RNN query by running only k-NN query. The time taken and the number of page accesses will definitely



(a) Average I/O (b) Average time

Figure 13: I/O and time for landsat data

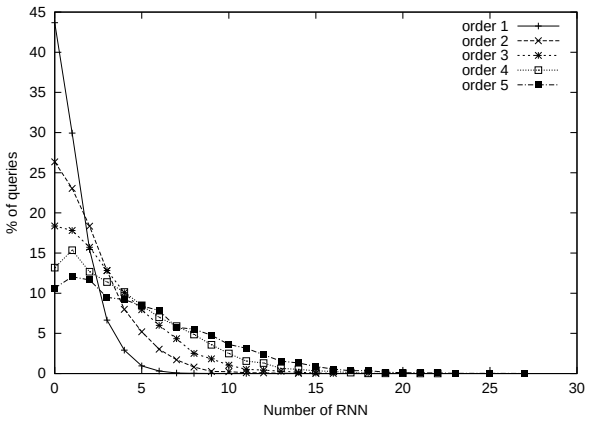


Figure 14: Percentage of Queries versus Number of RNN

depend on the number of data points and both will increase as the size of the data increases.

5.4 RNN of Order k

We extended our proposed approach to other variants of RNN such as *RNN of order k* queries. Figure 14 shows the plot of percentage of queries versus number of RNN for stock data set. It is clear from the figure that the traditional RNN problem (RNN of order 1) returns a zero result in a significant number of queries (around $30\% - 40\%$). This percentage drops down to $5\% - 10\%$ for RNN of order 5.

We performed experiments by executing k-NN queries to answer RNN of order k queries. Figure 15 shows the variation of average recall with number of NNs for RNN of order k with k=1-5. Average recall is defined as the percentage of queries where RNN is contained in the k-nearest neighbor set. We observe that with only finding a small

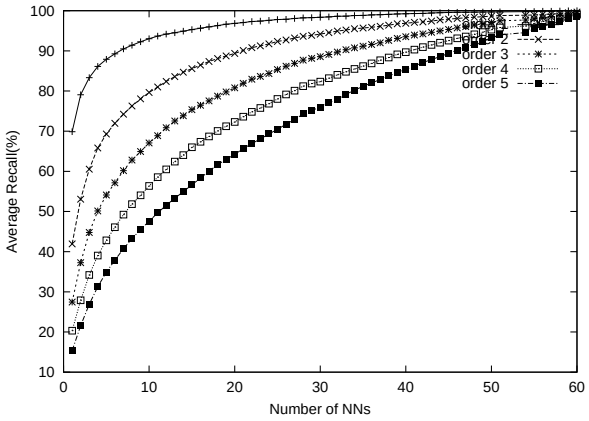


Figure 15: Plot showing average recall as a function of Number of NNs

number of NNs, we can achieve very high recall. The average recall increases with the increase in the number of NNs.

For answering RNN of order k queries we define *count range queries*.

Definition 2 *A count range query $\mathcal{CRQ}(q, r, k)$ returns true if the set $\{t \in S | d(q, t) < r\}$ has cardinality $\leq k$ and false otherwise.*

Count range query for RNN of order k checks whether the corresponding range query has k points inside it or not. Boolean range query is a count range query with count equals to 1. We can answer RNN of order k by running count range query instead of boolean range query in the last step of RNN algorithm.

Figure 16 shows the average I/O and time for landsat data for RNN of order k. Here we are only showing the case for k-NN + filtering I(optimization I)+ filtering II. Filtering I especially with optimization II becomes more complicated if the order of RNN increases. Optimization I can be used to eliminate some of the candidates but it is not very useful as the order of RNN increases since the processing cost involved in eliminating the candidate points increases. Lower bar in each of these bar charts correspond to running the k-NN query. The average number of I/Os required is almost as same as running a k-NN query.

5.5 Clustering

We also adapted our proposed algorithm on clustering based schemes. We performed experiments to see the effectiveness of the k-means algorithm for RNN problem. For a clustering based algorithm to be effective, the clustering needs to assign RNN of each point to the same cluster as the potential query point. In our experiments, we observed that retrieving a single cluster will mostly be enough to answer the RNN query. Figure 17 illustrates the result of our experiments. It shows the success rate for various data sets wrt. total number of clusters used to cluster the data set. Here we have defined the success rate as percentage of queries that the RNN is in the same cluster as query point.

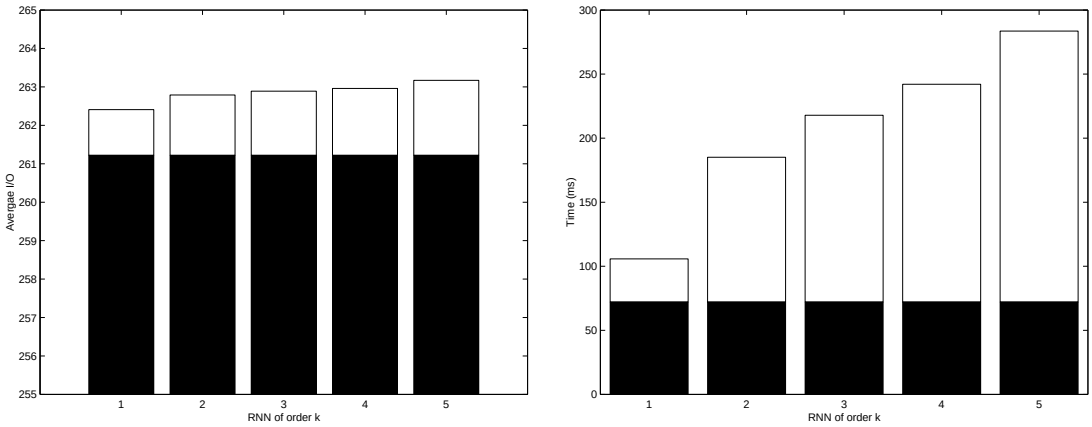


Figure 16: Average I/O and time for landsat data

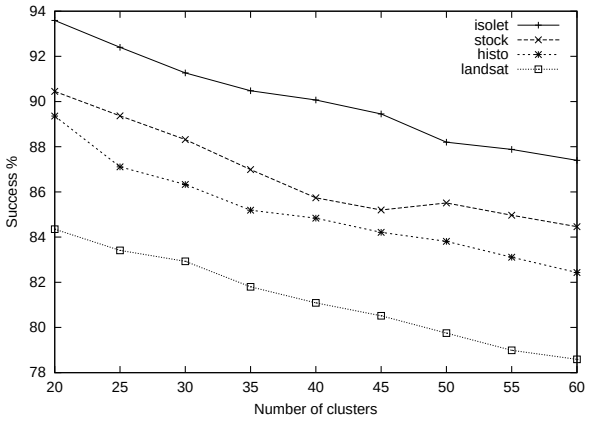


Figure 17: Success rate as a function of total number of clusters, when only a single cluster is retrieved

A point to note here is that we are reading only a single cluster. For example, for isolet data with 20 clusters 94% of the time RNN of a query is in the same cluster with the query. From the figure it is evident that clustering is an effective approach for answering high dimensional RNN queries. To improve the success rate a progressive search scheme can be used on top of the clustering. Progressively, user can search for nearest cluster then second most nearest cluster and so on. Figure 18 shows improvement in success rate if search is continued for neighboring clusters.

6 Conclusions

The problem of finding RNN in high dimensions is a challenging problem and we discussed some of the involved challenges. We proposed a solution to answer RNN queries in any dimensions. Our approach is based on the strong correlation in practice between k-NN

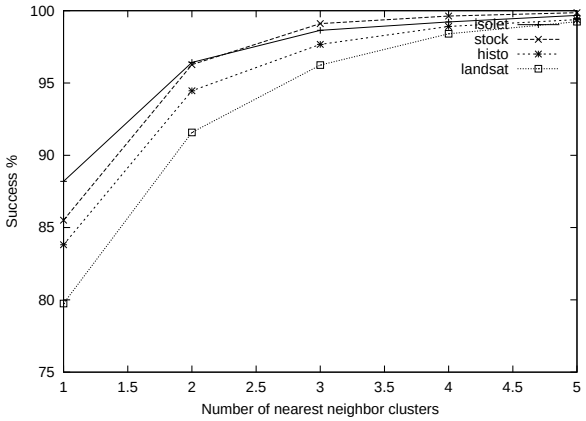


Figure 18: Success rate as a function of number of neighboring clusters

and RNN. The algorithm works in two phases. In the first phase, the k nearest neighbors of a query point are found. Out of these k points, some are eliminated by making use of bisector or distance optimization. In the next phase, the points obtained after applying optimization are analyzed using a novel type of query, called *Boolean Range Query* (*BRQ*) to find true RNN. Experiments on several real data sets showed that BRQ is much more efficient than range query, and can be effectively used to answer RNN queries. Performance is further improved by running multiple BRQ simultaneously.

We can find whether the set of points are sufficient to avoid any false misses by utilizing a cone based approach. The time for a k-NN query does not vary much with the increase in the value of k. So, alternatively user can choose a high value of starting k so as to avoid any false misses. Bisector optimization is found to be very efficient in eliminating candidate points. So, even for a large value of k, the number of points used in the second step (for running BRQ) would be a small set.

We also mention a novel kind of query, i.e. matching queries, which have many applications in the real world. Since we answer both k-NN and RNN queries at the same time, with slight modifications our algorithmic approach processes the matching queries efficiently. Using our approach, we can answer NN, RNN, and matching queries together almost in the same I/O and time as running a k-NN query.

Our approach is easily adaptable to any access structure without any changes on the current implementation. In this paper, we discussed effective way of implementing our algorithm on R-trees and clustering based structures. For dynamic data sets, no additional operation has to be performed except the routine insertion and deletion algorithms for the underlying index structure that has to take place anyway.

Acknowledgment

This research was partially supported by Department of Energy (DOE) grant DE-FG02-03ER25573 and National Science Foundation (NSF) grant CNS-0403342. We would like to thank Serdar Vural (The Ohio State University, ECE Department) for discussions of some of the ideas in this paper.

References

[1] http://www-users.cs.umn.edu/ karypis/cluto.

[2] http://gist.cs.berkeley.edu.

[3] J. V. Anil Maheshwari and N. Zeh. On reverse nearest neighbor queries. In *Proceedings of the 14th Canadian Conference on Computational Geometry, University of Lethbridge, Alberta, Canada*, Aug 2002.

[4] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, May 23-25 1990.

[5] B. Braunmuller, M. Ester, H. Kriegel, and J. Sander. Efficiently supporting multiple similarity queries for mining in metric databases. In *Proc of 16th IEEE Int. Conf. on Data Engineering (ICDE)*, San Diego, CA, March 2000.

[6] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *Proc. Int. Conf. Data Engineering*, pages 440–447, Sydney, Australia, 1999.

[7] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the Int. Conf. on Very Large Data Bases*, Athens, Greece, August 1997.

[8] J. Conway and N. Sloane. *Sphere packings, Lattices and Groups*. Springer-Verlag, 1988.

[9] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proc of 17th IEEE Int. Conf. on Data Engineering (ICDE)*, pages 503–511, Heidelberg, Germany, Apr. 2001.

[10] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998.

[11] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[12] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Proc. of 4th Int. Symp. on Large Spatial Databases*, pages 83–95, Portland,ME, 1995.

[13] K. V. R. Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 166–176, Seattle, Washington, June 1998.

[14] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 2000, Dallas, Texas, USA*, May 2000.

[15] F. Korn, S. Muthukrishnan, and D. Srivastava. Reverse nearest neighbor aggregates over data streams. In *Proceedings of the Int. Conf. on Very Large Data Bases*, Hong Kong, China, Aug. 2002.

[16] C. Lang and A. Singh. Accelerating high-dimensional nearest neighbor queries. In *SSDBM*, Edinburgh, Scotland, July 2002.

[17] D. B. Lomet and B. Salzberg. The hb-tree: A multi-attribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems*, 15(4):625–658, December 1990.

[18] J. Nievergelt, H. Hinterberger, and K.C.Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems 9*, 1:38–71, Mar. 1984.

[19] J. T. Robinson. The kdb-tree: A search structure for large multi-dimensional dynamic indexes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 10–18, 1981.

[20] N. Roussopoulos, S. Kelly, and F. Vincent. Nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 71–79, San Jose, California, May 1995.

[21] T. Seidl and H. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Chicago, Illinois, U.S.A., June 1998. ACM.

[22] I. Stanoi, D. Agrawal, and A. El Abbadi. Reverse nearest neighbor queries for dynamic databases. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 2000) in conjunction with SIGMOD 2000, Dallas, USA*, 2000.

[23] Y. Tao, D. Papadias, and X. Lian. Reverse knn search in arbitrary dimensionality. In *VLDB*, 2004.

[24] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 194–205, New York City, New York, Aug. 1998.

[25] C. Yang and K.-I. Lin. An index structure for efficient reverse nearest neighbor queries. In *Proceedings of the 17th International Conference on Data Engineering Databases, Heidelberg, Germany*, 2001.