High Dimensional Reverse Nearest Neighbor Queries

Amit Singh
Department of Computer and Information Science
The Ohio State University
singha@cis.ohio-state.edu

Hakan Ferhatosmanoğlu
Department of Computer and Information Science
The Ohio State University
hakan@cis.ohio-state.edu

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

ABSTRACT

Reverse Nearest Neighbor (RNN) queries are of particular interest in a wide range of applications such as decision support systems, profile based marketing, data streaming, document databases, and bioinformatics. The earlier approaches to solve this problem mostly deal with two dimensional data. However most of the above applications inherently involve high dimensions and high dimensional RNN problem is still unexplored. In this paper, we propose an approximate solution to answer RNN queries in high 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-NN of a query point is found and in the next phase they are further analyzed using a novel type of query *Boolean Range Query* (*BRQ*). Experimental results show that *BRQ* is much more efficient than both NN and range queries, and can be effectively used to answer RNN queries. Performance is further improved by running multiple *BRQ* simultaneously. The proposed approach can also be used to answer other variants of RNN queries such as RNN of order k, bichromatic RNN, and *Matching Query* which has many applications of its own. Our technique can efficiently answer NN, RNN, and its variants with approximately same number of I/O as running a NN query.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Theory, Algorithm, Experimentation, Performance

Keywords

Nearest Neighbor, Reverse Nearest Neighbor, Boolean Range Query

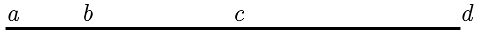
CIKM'03, November 3–8, 2003, New Orleans, Louisiana, USA.

1. INTRODUCTION

The goal of the Reverse Nearest Neighbor (RNN) problem is to find all points in a given data set whose nearest neighbor is a given query point. There are a wide range of applications that can benefit from an efficient implementation of RNN queries, such as decision support systems, continuous referral systems, profile-based marketing, maintaining document repositories, bioinformatics, etc.

A large number of applications for RNN queries is given in [12]. For example, a two-dimensional RNN query may ask the set of customers affected by the opening of a new store outlet location in order to inform the relevant customers. This query can also help to identify a good location which maximizes the number of potential customers. 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. Development of feature vectors and similarity metrics for document databases have been extensively studied in the information retrieval community. Similar to the digital library example, high dimensional RNN queries can be employed in profile-based marketing, where a company can start a new service based on the profiles of its customers interests. An RNN query can be posed to identify the customers who will find a new service the most suitable to them. The query will be based on the distance between profiles and the feature vector representing the new service. Many other applications are provided in the literature [12, 20, 23, 13]. Most of the concerned applications require an implementation of RNN for high dimensions, since the data in such applications (e.g., profiles, documents, etc.) are typically represented by high dimensional vectors.

Although RNN is a complement of NN problem it is more complex than NN problem even for two dimensional data. The relationship between NN/RNN is not symmetric and the number of RNNs are not known in advance. This can be illustrated using a simple example. Consider four points in a straight line a, b, c and d as shown below. The NN of a is b, b is a, c is b and d is c. In this example, although d has c as its NN it has no RNN (only possible candidate could be c but it is more closer to b than d).



The problems of multi-dimensional indexing [8, 16, 17, 9, 3, 15, 6, 22, 5] and NN query processing [18, 10, 19] have been extensively studied in the literature. The proposed technique in this paper can make use of any of the access structures and nearest neighbor algorithms existing in the literature. No additional data structures, or additional space is needed. 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. For large data sets, this step may need to perform I/O operations. In the second step we introduce a new kind of query *Boolean Range Query* (*BRQ*) which tests whether there is any point in a given region. Note that, the query is interested in neither the data objects nor the total number of data objects in the range. Since the query is restricted by nature, it can be implemented very efficiently compared to range or aggregation queries, where both the data objects or the total number of data objects are of interest. For most cases, it does not require any I/O operations, therefore takes negligible amount of time. Using our framework, we can answer both NN and RNN queries almost in the same time as running an NN query.

We implement the proposed approach based on a R-tree. In contrast to current 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.

The rest of the paper is organized as follows. We first discuss the previous work on RNN based queries in Section 2. Section 3 contains theoretical foundation for the RNN search in high dimensions. Section 4 presents the proposed algorithm. In Section 5, the results of the experiments are given with the implementation of our algorithm. In Section 6, we formalize other variants of RNN such as bichromatic RNN, RNN of order k, and matching query by giving motivating examples and showing how our framework can be useful in those kind of queries. Section 7 offers a summary and directions of future work.

2. DISCUSSION OF RELATED WORK

A brute force method for finding exact RNN is to first find the nearest neighbor of each point and then determine the points which have query point as the nearest neighbor. However, this approach is not feasible when the number of points in the database is large because its time complexity is of the order n^2.

The earlier approaches to find the RNN mostly deal with two dimensions. The problem of RNN was first introduced in [12] which used an R-tree based index structure to solve the problem. In case of a static database the authors 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. Briefly, 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. [23] improved the solution of [12] 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 standard R-tree by storing extra information about NNs of the points for each node, therefore requires prior computation of NN for each point. Recently [2] has proposed static and dynamic data structures for answering RNN queries in two dimensions.

[20] used a geometric approach to answer RNN queries in two dimensions. 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 RNN 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 subsequent step each point is checked for RNN.

The above mentioned approaches are for monochromatic case, that is when all the points are of the same category. In the bichromatic case there are two different kinds of points in the database such as clients and servers. Given a server q, RNN(q) retrieves all clients that are closer to q than to any other server. Korn et.al. used the same approach but the approach by [20] can not be applied here, because in this case a point can have more than six RNN points. Employing voronoi cells to answer the bichromatic case is proposed in [21], and it is again restricted to only two dimensions.

3. CHALLENGES AND OBSERVATIONS FOR HIGH DIMENSIONS

We first summarize some challenges in the high dimensional RNN problem, and then state an important result and some interesting observations which will allow us to develop an efficient solution for the RNN problem even 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 n dimensions, all of same size, that can be arranged around another hypersphere without any intersections [7]. In two dimensions the answer is six, i.e., six circles around another circle. This property was used in [20] 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 kissing number are known for n=1 to 9 and for n=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 [7]. Also note that, in other distance metrics such as L_∞, the cardinality of RNN(q) is at most $3^n - 1$ in n dimensions.

The exponential increase in the number of RNN with the increase in dimensions makes the geometric algorithm in [20] practically impossible for higher dimensions because number of regions would be very high (equal to number of possible RNN) and we have to search each region to find whether a RNN exists or not. So, any similar approach would be infeasible for high dimensions. The other two approaches in the literature [12, 23] used prior NN distance information

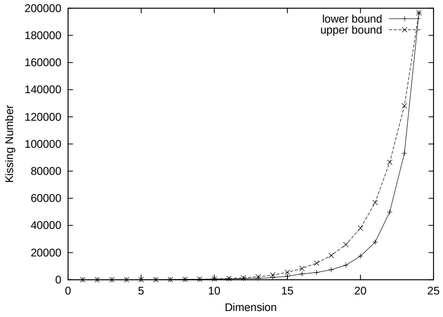


Figure 1: Kissing Number in High Dimensions

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 [12] will highly overlap for high dimensions, and the NN distance used for pruning in [23] will intersect most of the other MBRs and cause them not to be pruned during the RNN query.

Although the number of RNNs of a point in an n-dimensional vector space exponentially depends on n, 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 n-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 NNs of a point will be quite low compared to the theoretical maximum. 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 number of RNN. It is evident from the figure that in most cases the number of RNN is usually small (0-4) although the number of RNN theoretically increases exponentially with increase in dimension.

Another 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

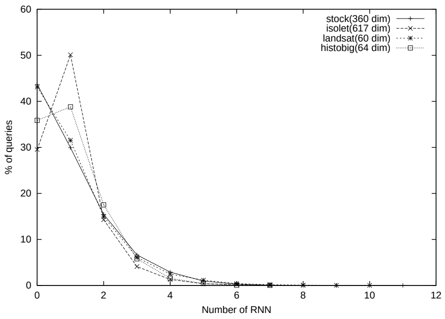


Figure 2: Percentage of Queries versus Number of RNN

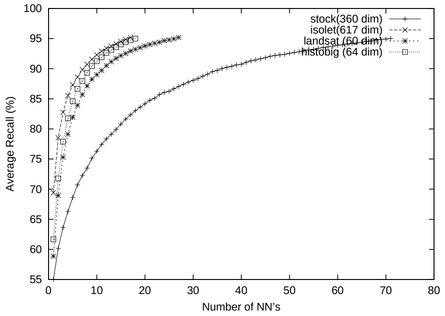


Figure 3: Plot showing average recall as a function of Number of NNs

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 two main observations mentioned in this section will serve as motivation and foundation for the proposed scheme which will be described next.

4. RNN ALGORITHM

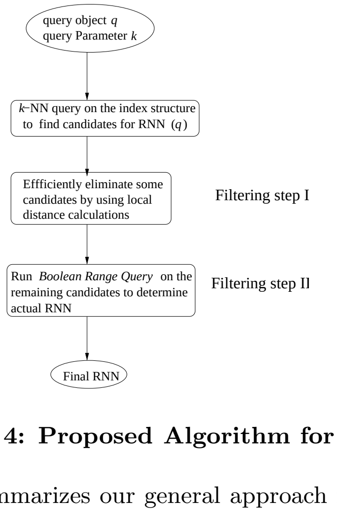

query object q
query Parameter k
k-NN query on the index structure
to find candidates for RNN (q)
Effficiently eliminate some
candidates by using local
distance calculations
Filtering step I
Run Boolean Range Query on the
remaining candidates to determine
actual RNN
Filtering step II
Final RNN

Figure 4: Proposed Algorithm for RNN

Figure 4 summarizes our general approach for answering RNN queries. The algorithm proceeds in two steps: In the first step, a k-NN search is performed returning the k closest data objects to the query and in the next step these k objects are efficiently analyzed to answer the query. The value of k that should be used in the first step depends on the data set and can be estimated by the experiments mentioned in figure 3. Based on the values of average recall we can choose the value of k we want to start with.

The following subsections explain the second step in detail with necessary optimizations that can be used.

4.1 Filtering step I

After the first step of the RNN algorithm we have k-NN of a query point. However, some of them can be ruled out from further elaboration by local distance calculations performed within the k objects, i.e., the candidate set. The principle of the elimination 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 kNN points is eliminated from the search (note that only k objects are involved in this test). We refer to this step as *filtering step I.* 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 using only this filtering. In this step, we can avoid some of the distance calculations (out of a possible $\binom{k}{2}$) by using an optimization based on comparison of the already computed distances. This elimination makes use of the fact that a multi-dimensional distance calculation is typically more expensive than a single floating point distance value comparison. This is formalized by the following lemma.

LEMMA 1. *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.*

After filtering step I, we have a set of candidate data points, each of which has a query point as its local NN (considering only the data points in the candidate set). For a point to be a RNN, we need to verify 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 a 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*, which is covered in the following subsection.

4.2 Boolean Range Queries (Filtering step II)

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 case 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 schemes. 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 a RNN. The range queries typically are more efficient than NN queries [14], 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 5.

```
Procedure $BooleanRangeQuery(Node n, point q, radius)$
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 $\Longrightarrow$ At least one point inside search region
        False $\Longrightarrow$ No point inside the search region
    If $n$ is a leaf node do:
        check if there exists a point $p$ such that
        $dist(p,q) \leq q$ 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.,
        $minmaxdist \leq radius$ 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 5: Boolean Range Query

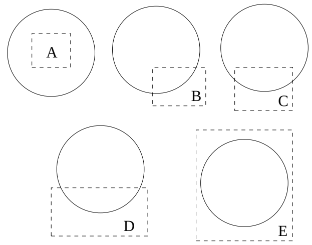


Figure 6: Cases for boolean range query intersecting with a region

Boolean Range Query versus Range Query

In a range query, typically multiple MBRs intersect the query region. Especially when the number of dimensions is high, the number of MBRs intersecting the query region is also high. [1]

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 multiple MBRs intersect a search region we do not need to access all of the MBRs.

Figure 6 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] 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.

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 don't need any page access (case A and C in Figure 6). 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 [18]. 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 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 with in 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 mimimal 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 in this framework. 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 [4]. 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 next.

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).

- 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 [1] 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 [11] is generally used to reduce the dimensionality. Depending on the chosen number of dimensions we took only those columns from the SVD matrix. 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 7 shows the comparison of the various data sets for different percentage of dimensions. These

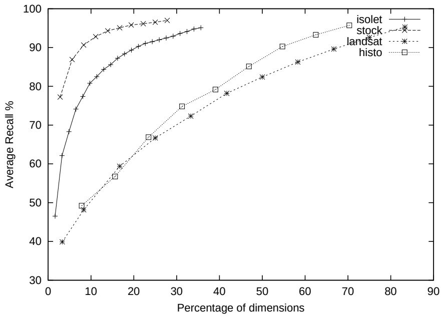


Figure 7: Plot showing average recall as a function of % of dimensions

Table 1: Comparison of Boolean Range Query with Range Query

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

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 1 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 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 2 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 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 same pages. Therefore, if we use the pages that are retrieved in the first phase for the second phase there will be a significant improvement in performance. So we ran experiments to measure the costs for NN queries, multiple boolean range queries, and the combined NN-BRQ. Table 3 shows the corresponding results. From the table and figure, it is evident that the cost of combined queries is essentially the same as that of NN query and is small compared to separate NN and BRQ. This shows that the proposed approach can effectively and efficiently answer RNN query in the running time of NN queries.

Table 2: Comparison of Single vs Multiple Boolean Range queries

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

Analysis of each step

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 three 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 filtering I)

3. k-NN + filter 1 + filter 2 (the overall algorithm)

In each of the above cases first step is same, i.e., finding a k-NN. 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 and isolet data with 10 dimensions only.

Figure 8 shows the average I/O and time for landsat data. 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. For comparison purposes on a landsat data, for a single query, the number of page accesses needed by brute force method is 3868986 and the time taken is 256 secs. 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 depend on the number of data points and both will increase as the size of the data increases.

6. VARIANTS OF RNN

In this section, we extend the proposed approach to other variants of RNN such as bichromatic RNN and RNN of order

Table 3: Performance of combined NN and Boolean range queries

Dim.	5			10			15		
	NN	Combined	Separate	NN	Combined	Separate	NN	Combined	Separate
Landsat	107.91	111.49	148.72	257.71	259.1	284.74	377.23	378.29	391.76
Histo	20.47	23.44	29.70	78.28	79.37	95.66	161.36	162.57	185.49
Stock	58.58	58.84	84.46	102.6	102.86	124.72	137.18	137.44	159.3
Isolet	102.69	102.69	146.84	171.24	171.29	177.1	228.57	228.57	229.74

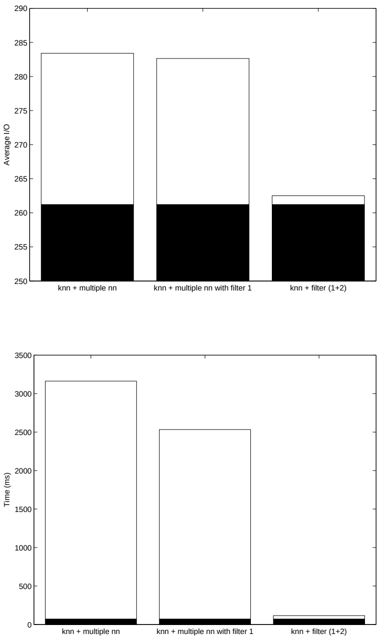


Figure 8: Average I/O and time for landsat data

k. Our approach can be used to answer these with slight modifications.

Bichromatic RNN: As mentioned in the section 2 for bichromatic case we have two sets of points, say client and server. RNN query for a server s asks to find all the client points that have s as the closest. The average number of RNN here is the ratio of number of clients/servers. Our algorithm can be used to answer the bichromatic case as follows.

1. Build two index structures (e.g. R-trees), one for client and one for server.

2. Run k-NN query on the client assuming server as the query point.

3. Run BRQ on those k points (candidate points) by taking radius as the distance between query point and candidate point.

4. If BRQ returns TRUE than the candidate point is not a RNN otherwise it is a RNN.

RNN of order k: RNN of order k asks 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, we may want to inform the customers affected by the opening of a new store location. Instead of informing the customers that are closest to this new store we can inform the 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 some applications it may be more useful to keep the query general, i.e., being second (or kth) closest to a data point is usually equally interesting to the user, especially given the approximate nature of the applications. We define *count range queries* which can be used to answer RNN of order k. Instead of running BRQ in the last step we will run count range queries. 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 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.

Matching Query: 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 of its own, i.e., 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. The proposed technique naturally answers a matching query almost in the same time as a k-NN query.

7. CONCLUSIONS AND FUTURE WORK

The problem of finding RNN in high dimensions has not been explored in the past. We discussed the challenges and elaborate some important observations related to high dimensional RNNs. Then we proposed an approximate solution to answer RNN queries in high dimensions. Our approach is based on the strong correlation in practice between k-NN and RNN. The algorithm works in two phases. In the first phase the k-NN of a query point is found and in the next phase they are further analyzed using *Boolean range queries* (*BRQ*). 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 also define other variants of RNN such as bichromatic RNN, RNN of order k and matching queries, which have many applications in the real world and showed that our algorithmic approach is also adaptable to these kind of queries.

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-tree. 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. Although we have studied the RNN problem for monochromatic case in detail, extension of our approach to bichromatic and RNN of order k is straightforward. As a future work, we plan to further explore the RNN and variants of RNN on top of clustering based scheme and perform more extensive performance evaluation of our techniques using various architectures.

8. ACKNOWLEDGEMENTS

This work was partially supported by DOE Career Award DE-FG02-03ER25573.

9. REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] I. Stanoi, M. Riedewald, D. Agrawal, and A. El Abbadi. Discovery of influence sets in frequently updated databases. In *Proceedings of the 27th International Conference on Very Large Databases, Rome, Italy*, 2001.

[22] 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.

[23] 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.