

Efficient k Nearest Neighbor Queries on Remote Spatial Databases Using Range Estimation *

Danzhou Liu Ee-Peng Lim Wee-Keong Ng
Centre for Advanced Information Systems
School of Computer Engineering
Nanyang Technological University, Singapore 639798
{P149571472, aseplim, awkng}@ntu.edu.sg

Abstract

K-Nearest Neighbor (k -NN) queries are used in GIS and CAD/CAM applications to find the k spatial objects closest to some given query points. Most previous k -NN research has assumed that the spatial databases to be queried are local, and that the query processing algorithms have direct access to their spatial indices; e.g., R -trees. Clearly, this assumption does not hold when k -NN queries are directed at remote spatial databases that operate autonomously. While it is possible to replicate some or all the spatial objects from the remote databases in a local database and build a separate index structure for them, such an alternative is infeasible when the database is huge, or there are large number of spatial databases to be queried. In this paper, we propose a k -NN query processing algorithm that uses one or more window queries to retrieve the nearest neighbors of a given query point. We also propose two different methods to estimate the ranges to be used by the window queries. Each range estimation method requires different statistical knowledge about the spatial databases. Our experiments on the TIGER data allow us to study the behavior of the proposed algorithm using different range estimation methods. Apart from not requiring direct access to the spatial indices, the window queries used in the proposed algorithm can be easily supported by non-spatial database systems containing spatial objects.

1 Introduction

1.1 Motivation

The Nearest Neighbor (NN) queries in spatial databases refer to finding the spatial objects nearest to some given query points. NN queries are used in a wide range of applications, such as Geographic Information Systems (GIS), Computer Aided Design (CAD), computational biology, decision support, and pattern recognition [27]. NN queries in spatial databases can be classified into five major categories: simple k -NN queries [2, 6, 8, 9, 17, 21, 22, 26], approximate k -NN queries [3, 10, 14], reverse NN queries [20, 28], constrained k -NN queries [13], and k -NN join queries [16]. In this paper, we focus on simple k -NN queries. Given a set of spatial objects denoted by \mathbb{S} , and a distance function d , a simple k -NN query for a query point q is to find the k objects in \mathbb{S} with smallest $d(q, o)$, where $o \in \mathbb{S}$. The query result can be represented as

$$NN(q, k, \mathbb{S}) = \{o_1, \dots, o_k\},$$

where $d(q, o_i) \leq d(q, o') \forall i, 1 \leq i \leq k, o_i \in \mathbb{S}$ and $\forall o' \in \mathbb{S} - \{o_1, \dots, o_k\}$.

With the rapid growth of the World Wide Web (WWW), large volume of spatial data are now available for access on the Web. For example, the Clearinghouse sponsored by the Federal Geographic Data Committee (FGDC) is a collection of over 250 servers that provides geospatial data on the Web [12]. While some spatial data on the Web are stored in databases managed by spatial database systems, a large proportion of these data may still be stored in data files, or SQL databases. The storage methods used affect the way the spatial data can be queried. For example, for spatial data stored in SQL databases, it is clearly not

*This work is funded by the SingAREN Project M48020004

possible to adopt a k -NN query algorithm that requires the use of a spatial index such as R-tree. Moreover, to query spatial data on the Web, it is often necessary to use some Web-based query interfaces such as HTML forms. The Web-based query interfaces, unlike spatial query languages, can only support very simple spatial queries but not the complex ones including the k -NN queries. Hence, in our research on evaluating k -NN queries against remote spatial databases, we only assume that the Web-based query interfaces support *window queries*. A window query retrieves spatial objects within a given bounding rectangle also known as the *window*. Window queries are relatively simple, and can be easily supported by most Web-based query interfaces.

In a literature survey, we found that almost all existing k -NN query algorithms require direct access to the spatial indices. Therefore, they cannot be directly applied to remote spatial databases that do not support remote index accesses. One may consider creating a new local spatial index for all the data downloaded from a remote spatial database and directly applying the existing algorithms. Such a strategy, however, does not scale well for large numbers of remote spatial databases and for remote spatial databases containing large amount of data. It also violates the local autonomy of these databases. In other words, new strategies for k -NN query evaluation are required.

1.2 Objectives and Contributions

The main objective of this research is to develop an algorithm to evaluate k -NN queries on remote spatial databases efficiently using window queries. When the window used is just right, one expects exactly k nearest spatial objects to be returned by the window query. When the window used is *loose*, more than the required k nearest neighbors will be returned. On the other hand, when the window is *tight*, fewer than k nearest neighbors will be returned.

We would like to propose the use of statistical knowledge about the remote databases to derive the window queries. The windows used in the window queries can be obtained by different range estimation methods. We would also like to evaluate the performance of our proposed k -NN algorithm using different range estimation methods.

The main contributions of this paper can be summarized as follows. First, it presents a generic k -NN query processing algorithm that can accommodate different range estimation methods. Second, we have developed two range estimation methods; namely, the density-based method and the bucket-based method. Each

method requires a different type of summary knowledge about the remote spatial database and it allows us to derive the window queries. Lastly, the paper describes a series of experiments conducted to evaluate the performance of our proposed k -NN algorithm using the two range estimation methods. To compare them, we have adopted several performance metrics, such as the *iteration*, *efficiency* and *accuracy*.

Apart from assuming that the remote spatial databases support window queries, we have also assumed that the statistical knowledge about each spatial database can be made available. This assumption requires cooperation from the spatial database owners. While the assumption may not always hold, we still adopt it for this preliminary work on k -NN queries on remote spatial databases. We also believe that our proposed methods can be further extended to handle cases where such statistical knowledge cannot be provided by the database owners. For example, we can adopt sampling to collect remote database information. Nevertheless, these extensions of our range estimation methods are beyond the scope of this paper. For simplicity, we have also assumed that the spatial objects are points in 2-D space. With some modifications, our methods should be able to handle more complex spatial objects and spatial objects in a higher dimensional space.

1.3 Outline of the Paper

The remaining sections of this paper are organized as follows. In Section 2, related work is presented. In Section 3, we present a generic k -NN query processing algorithm that can accommodate different range estimation methods. In Section 4, two methods for range estimation are proposed. Section 5 describes our performance experiments and presents the results. Finally, we conclude the paper and highlight our future research directions in Section 6.

2 Related Work

In this section, we survey existing research on k -NN query algorithms. Since our work involves simple k -NN queries, we have only examined work related to simple k -NN queries [2, 6, 8, 9, 17, 21, 22, 26]. Algorithms for simple k -NN queries may be divided into three major groups: *partition-based* algorithms, *graph-based* algorithms, and *range-based* algorithms.

Partition-based algorithms partition the space containing spatial objects recursively to create spatial indices such as quadtrees, K-D trees, and R-trees. The algorithms retrieve the k nearest neighbors from the

spatial indices by pruning away nodes that cannot lead to the k nearest neighbors. In addition, cost models for these algorithms are presented in [5, 25]. For example, Roussopoulos *et al.* [26] proposed an algorithm using the R*-tree [4] for simple 1-NN queries and the algorithm can be generalized to handle k nearest neighbors. The algorithm was further extended to reduce more unnecessary disk accesses by Hjaltason and Samet [17]. On the whole, the above partition-based algorithms can be very efficient but they cannot be adopted for querying remote spatial databases on the Web. As integral components of spatial database systems, spatial indices are usually not available to non-local applications. The partition-based algorithms are also not applicable to spatial data managed by non-spatial database systems.

Graph-based algorithms pre-calculate the nearest neighbors of spatial objects and create new index structures for the pre-calculated nearest neighbor information for efficient search [5]. Examples of such algorithms include the RNG* algorithm [2] and the algorithms using Voronoi diagrams [7, 11]. For example, one can first derive the Voronoi diagram for a given set of spatial points followed by indexing the cells in the Voronoi diagram. Given a query point q , finding the nearest neighbor can be simplified to finding the Voronoi cell that contains q . Although graph-based algorithms are very efficient and can be extended to support k -NN queries, they again requires some spatial index to be maintained for the collection of cells. The index may also occupy much storage space. Hence, these algorithms are not feasible in the Web environment.

Range-based algorithms refer to using range queries to retrieve k nearest neighbors. Lang *et al.* [21] proposed an algorithm for transforming a k -NN query into at most two range queries. The first range query is obtained by estimating the required range using sampled spatial data and fractal dimensionality [23] of spatial objects. If the k nearest neighbors are found, the goal is achieved. Otherwise, the second range estimation is needed. The second range is defined as the distance between the query point and the k th nearest neighbor within the index leaf nodes accessed during the first range query. This method, however, still need to access the underlying spatial index. Papadopoulos *et al.* [24] presented alternatives in answering k -NN queries by using range queries in a parallel environment (i.e., spatial objects are stored in a spatial index and the whole index is distributed over a number of servers). Their work, motivated by [26] and to exploit parallelization as much as possible, also assumes that the spatial index is accessible. Ciaccia *et al.* [9] employed the relative distance distributions of several “witnesses” se-

lected in some way among spatial objects to estimate the range. Yu *et al.* [22] transformed k -NN queries to one-dimensional range queries by partitioning spatial objects, selecting a reference point for each partition, and using B⁺-tree to index distance between spatial objects and their corresponding reference point. Both Ciaccia’s and Yu’s methods approximate the range by employing some sampled spatial objects. However, determining the sample size and selecting samples of spatial objects properly are still a challenge, and improper sampling will result in too much inaccuracy in estimating the data distribution, and/or storage overhead.

Compared with the above three types of k -NN algorithms, our proposed solution does not rely on access to the spatial index of the databases. Our density-based method derives some statistical knowledge about the distribution density of a set of spatial objects. The storage requirement of the statistical knowledge required is much smaller than that of the spatial indices used in the above k -NN algorithms. In [1, 18, 19, 29], various two-dimensional histograms have been proposed to estimate the selectivity of range queries. Our bucket-based range estimation method is very much inspired by their work.

3 k -NN Query Algorithm based on Range Estimation

In this section, we will outline our proposed algorithm for k -NN queries. Unlike the other algorithms, our algorithm transforms a k -NN query into one or more **window queries** without using any spatial index. The **window** used in a window query can be represented by $[(x_l, y_l), (x_u, y_u)]$, where (x_l, y_l) and (x_u, y_u) refer to its lower-left and upper-right corners respectively. Given a set of spatial objects denoted by \mathbb{S} and a window w , a window query about w refers to finding all spatial objects in \mathbb{S} located in w [15]. The window query result can be represented as

$$\{o \in \mathbb{S} \mid (x_l \leq o.x \leq x_u) \wedge (y_l \leq o.y \leq y_u)\}.$$

Ideally, we would like to retrieve exactly k nearest neighbors for a given query point by retrieving all spatial objects within a circle with the query point as the center and the distance between the query point and the k th nearest neighbors as the radius. Since our spatial databases are assumed to support window queries but not circle queries, we approximate a circle query by defining a window query with a window that inscribes the circle as illustrated in Figure 1.

Our proposed k -NN query algorithm is shown in Figure 2. This algorithm is designed to be generic enough

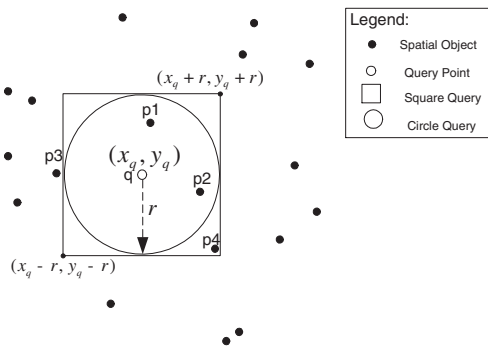


Figure 1. Example of Window Query

to accommodate different range estimation methods. At line 1, the `ESTIRANGE1` function first estimates the range (or radius) of the circle query to retrieve the k nearest neighbors of a given query point. The algorithm then derives the window query from the estimated range and evaluates the window query. The window query result is inserted into `rlist` at line 3. At line 4, we create an empty priority queue `nnqueue` of size k to maintain the k nearest neighbors. Only those spatial objects with a distance from q that is not larger than `range` are inserted into `nnqueue` from lines 6 to 12. For example, in Figure 1, p_4 will not be inserted into `nnqueue` because it is not within the circle with the query point as the center and r as the range. At line 13, `count` represents the number of nearest neighbors retrieved so far. If `count` is larger than or equal to k , all the k nearest neighbors would have been found and stored in `nnqueue`. Otherwise, more nearest neighbors have to be obtained by expanding the range. The expanded range can be computed from the current window and `count` using the `ESTIRANGE2` function at line 14. With a revised range, a new window query is evaluated against the spatial database. The whole process ends when all the k nearest neighbors are found.

Depending on the number of window queries required to retrieve all k nearest neighbors, this generic k -NN algorithm may involve one or more iteration. We say that a range estimation method is **loose** when the range given by the `ESTIRANGE1` function is large enough to cover all the k nearest neighbors, and `ESTIRANGE2` is not required at all. On the other hand, we say that a range estimation method is **tight** when the `ESTIRANGE1` function may return less than k nearest neighbors. Hence, the `ESTIRANGE2` function may be invoked to derive the revised range(s) to be used in the second or further window queries.

`KNNQUERY`(k, q)

Input:

required number of nearest neighbors k
query point q

Output:

k nearest neighbors `nnqueue`

```

01  range ← ESTIRANGE1( $k, q$ )
02  window ← [( $x_q - range, y_q - range$ ), ( $x_q +$ 
03   $range, y_q + range$ )]
04  rlist ← WINDOWQUERY(window)
05  nnqueue ← NEWPRIORITYQUEUE( $k$ )
06  count ← 0
07  for each object in rlist do
08    distance ← DIST(object,  $q$ )
09    if distance ≤ range then
10      ENQUEUE(nnqueue, (object, distance))
11      count ← count + 1
12    endif
13  enddo
14  while count <  $k$  do
15    range ← ESTIRANGE2( $k, q, window, count$ )
16    window ← [( $x_q - range, y_q - range$ ), ( $x_q +$ 
17     $range, y_q + range$ )]
18    rlist ← WINDOWQUERY(window)
19    count ← 0
20    for each object in rlist do
21      distance ← DIST(object,  $q$ )
22      if distance ≤ range then
23        ENQUEUE(nnqueue, (object, distance))
24        count ← count + 1
25      endif
26    enddo
27  endwhile
28  return nnqueue

```

Figure 2. k -NN Query Algorithm

4 Range Estimation Methods

In this section, we will present two different range estimation methods; namely, *density-based* and *bucket-based* methods. The first method yields tight ranges, and the second one yields loose ranges.

4.1 Density-Based Method

Density-based range estimation method is based on uniform distribution assumption. This method requires only a very simple piece of statistical knowledge about a spatial database; i.e., the *density* of the database. Given a database of spatial objects, we define the minimum bounding box (MBB) of the database as the minimum rectangle containing all the spatial objects. The **density** of the database, denoted by $D(MBB)$, is defined as $\frac{N}{Area(MBB)}$ where N is the total number of spatial objects. Given k and a query point q , the density-based range estimation method computes the first range estimate by

ESTIRANGE1(k, q)

Input:

required number of nearest neighbors k
 query point q

Output:

estimated range r

01 $r \leftarrow \sqrt{\frac{k}{\pi D(MBB)}}$
 02 **return** r

ESTIRANGE2($k, q, window, count$)

Input:

required number of nearest neighbors k
 query point q
 previous window $window$
 the number of nearest neighbors retrieved so far $count$

Output:

estimated range r

01 **if** $count == 0$ **then**
 02 $r \leftarrow 2 \times \frac{(window.x_u - window.x_l)}{2}$
 03 **else**
 04 $D(window) \leftarrow \frac{count}{(window.x_u - window.x_l) \times (window.y_u - window.y_l)}$
 05 $r \leftarrow \sqrt{\frac{k}{\pi D(window)}}$
 06 **endif**
 07 **return** r

Figure 3. Density-based Range Estimation Method

$r = \sqrt{\frac{k \text{Area}(MBB)}{\pi N}} = \sqrt{\frac{k}{\pi D(MBB)}}$. The corresponding ESTIRANGE1 function is shown in Figure 3. Given the range, the window used in the first window query is $[(x_q - r, y_q - r), (x_q + r, y_q + r)]$.

The above range estimate is not loose, and the corresponding window query may return less than k nearest neighbors. When that happens, further refinement on the range estimate is required. If the first window query does not return any spatial objects, we simply double the original range r . Otherwise, we compute the density of spatial objects within the window and derive the next range estimate by the new density information. Let $D(window)$ be the density of the $window$ used in the last window query, the new range estimate is derived by $\sqrt{\frac{k}{\pi D(window)}}$. The corresponding range estimation function ESTIRANGE2 is shown in Figure 3.

The above density-based range estimation method demonstrates some important features. Firstly, the space required to store the density information takes only several bytes. Thus, the space complexity is $\mathcal{O}(1)$. Secondly, every time a new range estimate is required; it is derived from the density of the window used in the previous window query. This range estimation

method further guarantees that the estimated range increases monotonically. The upper bound on the number of times the range estimation functions are called (or number of window queries) for a k -NN query can be determined using the following theorem.

Theorem 1 *Let the MBB of a spatial database that contains N spatial objects be $[(x_l, y_l), (x_u, y_u)]$. Assume that the query point q is inside the MBB. In order to retrieve k nearest neighbors, the upper bound on the number of times the range estimation functions are called (denoted by I_{max}) for the density-based method is*

$$I_{max} = \begin{cases} \left\lceil \frac{\ln r_{max} - \ln r_0}{\ln 2} \right\rceil + 1 & \text{if } k = 1; \\ \left\lceil \frac{\ln r_{max} - \ln r_0}{\ln \sqrt{\frac{4k}{\pi(k-1)}}} \right\rceil + 1 & \text{if } k > 1. \end{cases}$$

where

$$r_0 = \sqrt{\frac{k(x_u - x_l)(y_u - y_l)}{\pi N}}$$

and

$$r_{max} = \sqrt{(x_u - x_l)^2 + (y_u - y_l)^2}.$$

Proof:

According to the given conditions, the first range, denoted by r_0 , estimated by ESTIRANGE1 is

$$r_0 = \sqrt{\frac{k}{\pi D(MBB)}} = \sqrt{\frac{k(x_u - x_l)(y_u - y_l)}{\pi N}}.$$

The function ESTIRANGE2 always returns a range value (denoted by r_i) that is larger than the previous range (denoted by r_{i-1}). That is,

$$r_i = \begin{cases} 2r_{i-1} & \text{if } count = 0; \\ \sqrt{\frac{4k}{\pi \times count}} r_{i-1} & \text{if } 0 < count \leq k - 1. \end{cases}$$

The maximum value for r_i , denoted by r_{max} , is the length of the MBB's diagonal, $\sqrt{(x_u - x_l)^2 + (y_u - y_l)^2}$. For any query point inside the MBB, the window query with the range r_{max} will retrieve all the spatial objects including the k nearest neighbors.

There are two cases to be considered for calculating I_{max} . Consider the first case when $k = 1$. The maximum number of calls to the range estimation functions is achieved when the last window query retrieves the nearest neighbor with a range not less than r_{max} and the previous window queries retrieve none; i.e., $count = 0$. In other words, ESTIRANGE1 is called

once, while ESTIRANGE2 is called $I_{max} - 1$ times with $count = 0$. That is, I_{max} should be the smallest integer such that

$$r_{max} \leq 2^{I_{max}-1} r_0$$

Hence,

$$I_{max} = \left\lceil \frac{\ln r_{max} - \ln r_0}{\ln 2} \right\rceil + 1, \text{ if } k = 1.$$

Consider the second case when $k > 1$. To have maximum number of calls to ESTIRANGE2, r_i should increase with the smallest rate. This is achieved when $\sqrt{\frac{4k}{\pi \times count}}$ is as small as possible. In other words, the value of $count$ need to be $k - 1$. The range r_i will therefore increase at the smallest rate of $\sqrt{\frac{4k}{\pi \times (k-1)}}$.

Then we can derive I_{max} as the smallest integer such that:

$$r_{max} \leq \left(\sqrt{\frac{4k}{\pi(k-1)}} \right)^{I_{max}-1} r_0$$

Hence,

$$I_{max} = \left\lceil \frac{\ln r_{max} - \ln r_0}{\ln \sqrt{\frac{4k}{\pi(k-1)}}} \right\rceil + 1, \text{ if } k > 1.$$

■

4.2 Bucket-Based Method

In the bucket-based range estimation method, we use summary information about partitions or buckets of spatial objects to estimate ranges. Buckets are created by dividing the entire space into different groups [1]. For each bucket, we maintain its minimum bounding box (MBB) and the total number of spatial objects inside the bucket.

In [1], Acharya *et al.* proposed four strategies to create buckets. They are the *Equi-Count*, *Equi-Area*, *Min-Skew* and *Min-Overlap* partitioning strategies. The *Equi-Count* partitioning strategy creates buckets containing roughly the same number of spatial objects. The *Equi-Area* partitioning strategy creates buckets with MBBs having the same area. The *Min-Skew* partitioning strategy divides spatial objects into buckets such that each bucket contains uniformly distributed spatial objects. The *Min-Overlap* partitioning strategy is derived from R*-tree [4] and it creates buckets that have minimal overlaps among them.

Our bucket-based range estimation method may adopt any of the above partitioning strategies. In our experiments, we have implemented the *Equi-Area* partitioning strategy. As shown in Figure 5, the range estimation method first calculates the maximum distance

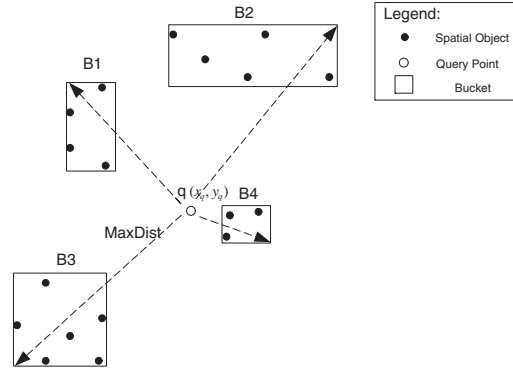


Figure 4. Example of Bucket-based Range Estimation Method

between the query point and each bucket from lines 1 to 3. The **maximum distance** here is defined as the farthest distance between the query point and the MBB of the bucket. Afterwards, we sort the buckets by their maximum distances in ascending order at line 4. From lines 7 to 13, the method finds the first few buckets in the ordered list that can return k or more spatial objects. For example, as shown in Figure 4, the buckets are sorted by their maximum distance in the following order: B4, B1, B2, B3. Suppose k is 4. B4 and B1 together return 7 spatial objects while B4 only returns 3. Hence, the estimated range is assigned the maximum distance between B1 and query point. It is quite obvious the bucket-based range estimation method is *loose* since it always provides an estimated range that covers all the k nearest neighbors. The ESTIRANGE2 function is therefore not required.

The performance of the bucket-based range estimation method is affected by the distribution of spatial objects in the buckets and the number of buckets. If the number of buckets is too small, it will likely overestimate the range leading to the retrieval of many unwanted spatial objects. While more buckets will yield better performance, it will require more storage overhead. Suppose we need 16 bytes to store both the upper-right corner and lower-left corner of the MBB of a bucket, and 4 bytes to store the number of spatial objects within the bucket (i.e., *bucket.count*). The storage space required to store all the bucket information is $20N_B$, where N_B is the total number of buckets. Hence, the space complexity is $\mathcal{O}(N_B)$.

```

ESTIRANGE1( $k, q$ )
Input:
  required number of nearest neighbors  $k$ 
  query point  $q$ 
Output:
  the estimated range  $r$ 

01  for each bucket in the BucketList do
02    bucket.maxdist  $\leftarrow$  MAXDIST(bucket,  $q$ )
03  enddo
04  BucketList.sort() // sort buckets by the
maximum distance to  $q$ 
05  count  $\leftarrow$  0
06  maxdist  $\leftarrow$  0
07  for each bucket in the BucketList do
08    count  $\leftarrow$  count + bucket.count
09    maxdist  $\leftarrow$  MAX(maxdist, bucket.maxdist)
10    if(count  $\geq$   $k$ ) then
11      break
12    endif
13  enddo
14  return maxdist

```

Figure 5. Bucket-based Range Estimation Method

5 Experiments

In our experiment, we used the *NJ Road* dataset from TIGER [30]. This dataset contains the road data for the state of New Jersey in the line segment format. In our experiments, we calculated the centroid of each line segment, and randomly selected almost 5,000 points among all centroids. The selected centroids are shown in Figure 6. The latitude and longitude of the original centroids were shifted for the computational convenience.

To measure the performances of our methods, we adopted three measures; namely, *iteration*, *average accuracy*, and *average efficiency* [31]. *Iteration* refers to the number of window queries needed to obtain all k nearest neighbors. Average accuracy refers to the average ratio of the number of actual nearest neighbors retrieved to k in each window query. Mathematically, $accuracy_{avg} = \frac{\sum_{i=1}^{iteration} accuracy_i}{iteration}$, where $accuracy_i = \frac{nn_i}{k}$ (nn_i denotes the number of actual nearest neighbors retrieved in the i th window query). Average efficiency is the average ratio of the number of actual nearest neighbors retrieved to the number of spatial objects retrieved in each window query. Formally, $efficiency_{avg} = \frac{\sum_{i=1}^{iteration} efficiency_i}{iteration}$, where $efficiency_i = \frac{nn_i}{o_i}$ (o_i denotes the number of spatial objects retrieved in the i th window query). For some window queries that do not return any spatial objects, their efficiency is undefined and hence excluded from the com-

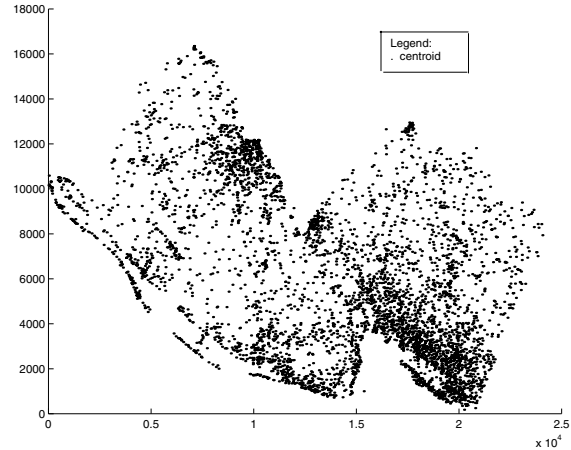


Figure 6. NJ Road Dataset

Iteration	k						
	1	5	10	15	20	25	50
minimum	1	1	1	1	1	1	1
maximum	8	7	7	6	6	6	5
upper bound	9	20	25	26	27	27	26

Table 1. Minimum, Maximum and Upper Bounds on the number of Iterations for Density-based Method

putation of the average efficiency measure. Ideally, all the measures are 1. If a method requires fewer iterations to retrieve k nearest neighbors and the average accuracy and efficiency are high, this method is considered good.

In our experiments, we used different k values, $k \in \{1, 5, 10, 15, 20, 25, 50\}$. We randomly selected 100 points in the space as query points. For each k , the average of the three measures (i.e., iteration, average accuracy and average efficiency) were taken for the 100 query points. In addition, for the bucket-based method, we only calculated average efficiency. Other measures are omitted since the method uses loose range estimation (i.e., $iteration = 1$, and $accuracy_{avg} = 1$).

Figures 7 to 9 depict the performance results of the k -NN query algorithm using density-based and bucket-based range estimation methods. For the bucket-based method, we experimented with different bucket numbers; i.e., 64, 100, and 256.

As shown in Figure 7, the k -NN query algorithm based on density-based range estimation method requires fewer number of iterations (window queries) for each k -NN query as k increases. When $k = 1$, an average of about 4.25 iterations (window queries) are re-

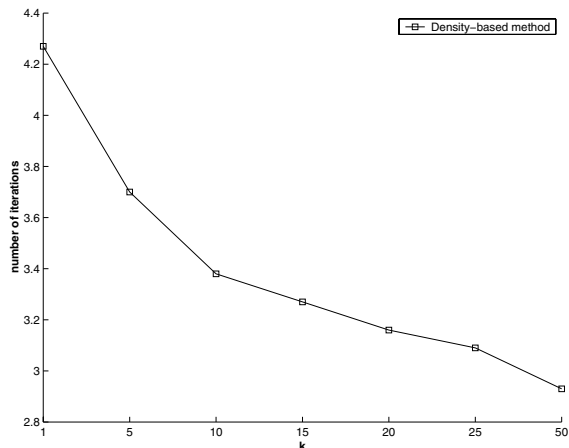


Figure 7. Number of Iterations for Density-based Range Estimation Method

quired. The number of window queries drops to about 3 when $k = 50$. This suggests that the density-based method works better for larger k 's. In Table 1, the minimum and maximum numbers of iterations for each k value are given. The maximum number of iterations among all the k -NN queries ranges from 5 to 8. Using Theorem 1, we derived the upper bounds on the number of iterations and show them in Table 1. It was encouraging to see the maximum numbers of iterations occurred during our experiments stayed well within the upper-bounds.

Figure 8 shows the accuracy of k -NN query algorithms based on density-based method. The average accuracy ranges from 0.43 (when $k = 1$) to 0.58 (when $k = 50$). Again, the density-based method delivers better performance for larger k . To give an idea how the density-based method performs in the first window query, we also showed the average accuracy for the first window query ($accuracy_{first}$) in the figure. The figure shows that the accuracy of the first window query is about 0.2 lower than the average of all window queries. On average, about 25% of the k -NN queries got the nearest neighbors for $k = 1$. As k approaches 50, the first window query returns on average more than 35% of the nearest neighbors.

In Figure 9, we observe that both the density-based and bucket-based methods improve their efficiency as k increases. The figure also depicts that the density-based method has a better efficiency. Efficiency is a measurement of the proportion of nearest neighbors in the spatial objects returned by the window query. The figure suggests that the bucket-based method always over-estimates the ranges required to find nearest

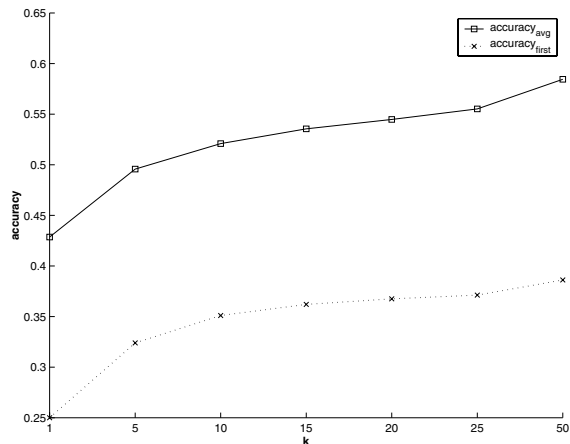


Figure 8. Accuracy for Density-based Range Estimation Method

neighbors as it uses only one window query for each k -NN query. On the other hand, the density-based method usually gives much smaller ranges for the window queries. The figure, in some way, surprised us as the efficiency of the density-based method was lower than our original expectation. Due to its conservative approach of increasing the window size, we expected perfect efficiency (i.e., 1) for the density-based method for its first few window queries. After a more detailed investigation, we realized that the efficiency of some initial window queries of the density based method could be zero as some window queries retrieved spatial objects (some may qualify as nearest neighbors) which are not within their corresponding circular ranges. Since the k -NN algorithm can only consider objects within these circular ranges as nearest neighbors, such window queries will have zero efficiencies. Since density-based range estimation method only returns all the k nearest neighbors in the last iteration, we also computed the efficiency of the last iteration (denoted by $efficiency_{last}$) for the density-based method as shown in Figure 9. The efficiency of the last iteration (window query) for the density-based method is clearly better than that of bucket-based method although the gap between them becomes closer as k increases.

6 Conclusions

In this paper, we describe a window query approach to evaluate k -NN queries on remote spatial databases. Our research has been motivated by the large amount of spatial information on the Web and their limited query interface. While most existing k -NN research as-

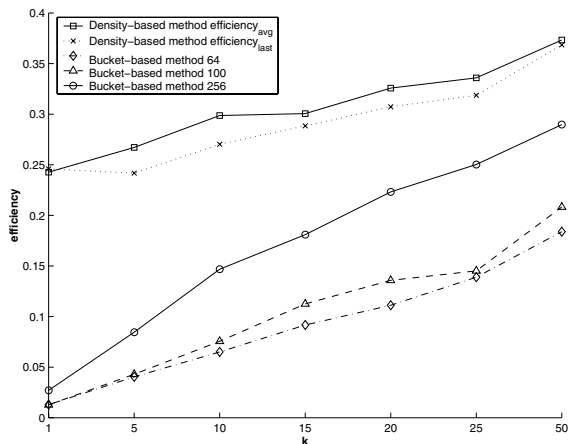


Figure 9. Efficiency for Density-based and Bucket-based Range Estimation Methods

sumes direct access to the indices of spatial databases, we have adopted a less intrusive approach to evaluate k -NN queries.

We have described a k -NN query algorithm that incorporates different range estimation methods for determining the window queries to be used for the remote databases. We have also proposed the density-based and bucket-based range estimation methods, and conducted experiments to evaluate their performance. Our experiments have shown that the k -NN query algorithm based on both range estimation methods improves as k increases. While the density-based method has better efficiency, it requires an average of 3 to 4 window queries to find all the nearest neighbors.

In the following, we outline two interesting topics for future research. These include:

- Extending our range estimation methods with sampling techniques. At present, our range estimation methods depend on statistical knowledge provided by the database owners. Although this is less intrusive compared with the k -NN algorithms using indices, it is still not good enough in the realistic environment. We therefore plan to investigate how the statistical knowledge can be automatically constructed using sampling techniques.
- Developing strategies to select the appropriate range estimation methods for evaluating k -NN queries. The density-based and bucket-based methods are only the first two range estimation methods proposed so far. We anticipate more range estimation methods could be developed in the future. It is therefore important to study how

the different methods should be chosen for a given remote spatial database.

References

- [1] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 13–24, Philadelphia, 1999.
- [2] S. Arya. *Nearest Neighbor Searching and Applications*. PhD thesis, Department of Computer Science, University of Maryland, College Park, MD, USA, 1995.
- [3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.
- [4] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, Atlantic City, NJ, USA, 1990.
- [5] S. Berchtold, C. Bohm, D. A. Keim, and H. P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proc. of the ACM Symposium on Principles of Database Systems*, pages 78–86, Tucson, AZ, USA, 1997.
- [6] S. Berchtold, B. Ertl, D. A. Keim, H. P. Kriegel, and T. Seidl. Fast nearest neighbor search in high-dimensional spaces. In *Proc. of the 14th Int. Conf. on Data Engineering*, pages 23–27, Orlando, FL, USA, 1998.
- [7] S. Berchtold, D. A. Keim, H. P. Kriegel, and T. Seidl. Indexing the solution space: A new technique for nearest neighbor search in high-dimensional space. *IEEE Trans. on Knowledge and Data Engineering*, 12(1), 2000.
- [8] K. L. Cheung and A. W. C. Fu. Enhanced nearest neighbor search on the R-tree. *SIGMOD Record*, 27(3):16–21, 1998.
- [9] P. Ciaccia, A. Nanni, and M. Patella. A query-sensitive cost model for similarity queries with M-tree. In *Proc. of the 10th Australasian Database Conf. (ADC'99)*, pages 65–76, Auckland, New Zealand, 1999.

- [10] P. Ciaccia and M. Patella. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proc. of the 16th Int. Conf. on Data Engineering*, pages 244–255, San Diego, CA, USA, 2000.
- [11] K. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal of Computing*, 17:830–847, 1988.
- [12] The Federal Geographic Data Committee. *The Clearinghouse*. URL:<http://www.fgdc.gov/clearinghouse>.
- [13] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. E. Abbadi. Constrained nearest neighbor queries. In *Proc. of the 7th Int. Symposium on Spatial and Temporal Databases (SSTD)*, Los Angeles, CA, USA, 2001.
- [14] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proc. of the 17th Int. Conf. on Data Engineering*, Heidelberg, Germany, 2001.
- [15] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
- [16] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 237–248, Seattle, WA, USA, 1998.
- [17] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Trans. on Database Systems*, 24(2):265–318, 1999.
- [18] J. Jin, N. An, and A. Sivasubramaniam. Analyzing range queries on spatial data. In *Proc. of the 16th Int. Conf. on Data Engineering*, pages 589–598, San Diego, CA, USA, 2000.
- [19] F. Korn, T. Johnson, and H. V. Jagadish. Range selectivity estimation for continuous attributes. In *Proc. of Int. Conf. on Scientific and Statistical Database Management*, pages 244–253, Cleveland, OH, USA, 1999.
- [20] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, Dallas, TX, USA, 2000.
- [21] C. A. Lang and A. K. Singh. A framework for accelerating high-dimensional NN-queries. Technical Report TRCS01-04, University of California, Santa Barbara, 2001.
- [22] B. C. Ooi, C. Yu, K. L. Tan, and H. V. Jagadish. Indexing the distance: an efficient method to KNN processing. In *Proc. of the 27th Int. Conf. on Very Large Data Bases*, Roma, Italy, 2001.
- [23] B. U. Pagel, F. Korn, and C. Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *Proc. of the 16th Int. Conf. on Data Engineering*, pages 589–598, San Diego, CA, USA, 2000.
- [24] A. Papadopoulos and Y. Manolopoulos. Nearest neighbor queries in shared-nothing environments. *GeoInformatica*, pages 369–392, 1997.
- [25] A. Papadopoulos and Y. Manolopoulos. Performance of nearest neighbor queries in R-trees. In *Proc. of the 6th Int. Conf. on Database Theory*, pages 394–408, Delphi, Greece, 1997.
- [26] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 71–79, San Jose, CA, USA, 1995.
- [27] S. Shekhar, S. Chawla, S. Ravada, A. Fetterer, X. Liu, and C. T. Lu. Spatial databases - accomplishments and research needs. *IEEE Trans. on Knowledge and Data Engineering*, 11(1):45–55, 1999.
- [28] I. Stanoi, D. Agrawal, and A. E. Abbadi. Reverse nearest neighbor queries for dynamic databases. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 44–53, Dallas, TX, USA, 2000.
- [29] Y. Theodoridis and T. Sellis. A model for the prediction of R-tree performance. In *Proc. of the 14th ACM Symposium on Principles of Database Systems*, pages 161–171, Montreal, Canada, 1996.
- [30] U.S. Bureau of the Census. *Tiger/line files*. URL:<http://www.census.gov>.
- [31] C. Yu, P. Sharma, W. Y. Meng, and Y. Qin. Database selection for processing k nearest neighbors queries in distributed environments. In *Proc. of the ACM/IEEE Joint Conf. on Digital Libraries*, pages 215–222, Roanoke, VA, USA, 2001.