Nearest Neighbor*hood* Search in Spatial Databases

Dong-Wan Choi #1, Chin-Wan Chung #2

#CS Department, KAIST, Daejeon, Korea

1dongwan@islab.kaist.ac.kr 2chungcw@kaist.edu

***Abstract*—This paper proposes a *group* version of the nearest neighbor (NN) query, called the *nearest neighborhood* (*NNH*) query, which aims to find the nearest group of points, instead of one nearest point. Given a set O of points, a query point q, and a ρ-radius circle C, the NNH query returns the nearest placement of C to q such that there are at least k points enclosed by C. We present a fast algorithm for processing the NNH query based on the incremental retrieval of nearest neighbors using the R-tree structure on O. Our solution includes several techniques, to efficiently maintain sets of retrieved nearest points and identify their validities in terms of the closeness constraint of their points. These techniques are devised from the unique characteristics of the NNH search problem. As a side product, we solve a new geometric problem, called the *nearest enclosing circle* (*NEC*) problem, which is of independent interest. We present a linear expected-time algorithm solving the NEC problem using the properties of the NEC similar to those of the *smallest enclosing circle*. We provide extensive experimental results, which show that our techniques can significantly improve the query performance.**

I. Introduction

Nearest neighbor (NN) search is one of the most fundamental problems, which has been extensively studied in various fields of computer science such as data mining, image processing, information retrieval, and spatial databases, to name a few. Given a set O of points and a query point q, the NN query finds the closest point in O to q. In spatial databases, the NN query can be used in finding the nearest *point of interest* (*POI*) such as a restaurant to a user's current location.

Nearest Neighbor*hood* Query. In this paper, by extending the NN query, we consider a *group* version of the NN query, and thereby propose the *nearest neighborhood* (*NNH*) query in spatial databases. Intuitively, the goal of the NNH query is to find the location of the nearest group of points, namely a *neighborhood*, which is different from returning the single nearest point, namely a *neighbor*, in the usual NN query. Thus, the primitive data type is extended from a neighbor to a neighborhood.

The most crucial matter in the NNH query is how to define the neighborhood. Our intuition is that a neighborhood can be represented as a particular-sized *region of interest* (*ROI*) containing multiple POIs. Of course, the number of points constituting a neighborhood cannot be too small. Thus, in the NNH query, we seek the nearest ROI (instead of the nearest POI) which contains at least a particular number of POIs.

It is reasonable for each ROI to be a particular-radius *circle* because it guarantees that distances between any pair of points in the ROI are not larger than a certain threshold, and each point is at most a particular distance away from the center of the ROI. Therefore, we formally define the NNH query as follows:

"*Given a set O of points, a query point q, a positive number k, and a circle C of a given radius ρ, the NNH query returns the nearest location (center) of C such that the number of points covered (enclosed) by C is at least k.*"

Let us consider Figure 1 to show the difference between our NNH query and the NN query. Suppose $k = 3$ and p_i is closer to q than p_j, where $i < j$ and $i, j \in [1, 9]$. Then the answer of the normal NN query is p_1. However, the NNH query will return c, which is the center of the ρ-radius circle enclosing three points, p_2, p_4, and p_5, shown with a dotted line.

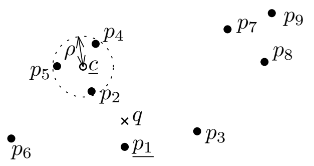


Fig. 1. An example of the nearest neighborhood query ($k = 3$)

Applications. This group version of the NN query, i.e., the NNH query, can play an important role in many applications employing spatial databases as follows:

- **Mobile Social Network.** In mobile social networks, the *location-based mobile community* can be typically formed by a group of mobile subscribers whose distances from the others are less than a particular threshold [1]. By setting the radius parameter ρ to be such a threshold, the NNH query can be utilized to extract mobile communities nearby a user's current location.
- **Clustered kNN Search.** The NNH query can also be interpreted as a variant of the kNN query. In spatial databases, the conventional kNN search may not give the best answer especially when the resulting set of k neighbors need to be clustered. Consider a tourist who is looking for a location for a dinner. The usual kNN search will return the group of k nearest restaurants to his/her current location. However, they may not be clustered, and if the tourist is not satisfied with the nearest restaurant, s/he may have to travel a long distance to go to another restaurant among other candidates in the group. In this case, the best answer for the tourist will be the nearest *spot* surrounded by clustered k restaurants, instead of k nearest restaurants themselves. It is easy to see that this can be answered by our NNH query using a small ρ value.
- **Spatial Data Mining.** It is also useful in spatial data

mining to find the nearest neighborhood. Many works in spatial data mining involve a large amount of historical location data, and aim to identify regions of interest that are usually represented as spatial clusters [2], [3], [4]. In such works, it is not always necessary to partition the all the location data into meaningful clusters, but to identify one or a few clusters in the proximity of a location of interest such as school, city hall, and museum. For instance, in order to determine *danger zones* nearby a school, it suffices to extract the clusters of crime locations that are nearest to the school rather than extracting all the clusters in the dataset.

Surprisingly, this fundamental spatial query has been rarely studied in the literature. There are several attempts to apply the concept of a *group* to the NN query, which are started from the *aggregate nearest neighbor* (*ANN*) query [5] and further extended to many variants based on it. However, the basic goal of the ANN query is to find the closest single point to a group of query points, which is far different from our objective in the NNH query. To the best of our knowledge, the NNH query is the first attempt to consider finding the nearest group of k points that are clustered in a certain region.

Solution Overview. At first glance, it may appear that the NNH query can be processed by clustering the entire dataset using a sophisticated clustering algorithm, and thereby finding the cluster nearest to the query point. Unfortunately, this solution is not only inefficient due to an expensive clustering task, but also *incomplete* for our NNH query. This is because clustering algorithms basically aim to partition the given dataset into a specific number of clusters, rather than find all clusters containing at least k points. Moreover, our NNH formulation limits the size of clusters, which makes the query even more difficult to answer using an ordinary clustering algorithm.

For a feasible solution that can answer the NNH query, we should check out all combinations of k or more points, or preprocess all such combinations in an index structure, and then identify the closest one. This, however, suffers from a prohibitively long running/preprocessing time and an infeasibly massive volume of the index structure.

To address the challenges of processing the NNH query, we propose an R-tree based algorithm together with several techniques to improve the query performance. Our basic query processing scheme is based on the incremental retrieval of nearest neighbors by the best-first search using the R-tree. More specifically, while incrementally visiting the next nearest neighbor, we maintain a structure to keep track of clusters of points retrieved so far and thereby checking whether those clusters can be covered by a circle of a given size. The most challenging task for this process is to update all the clusters accordingly for every retrieval of the next nearest neighbor. By means of the properties of the *smallest enclosing circle*s, we devise efficient update methods for the maintenance of all the clusters under consideration. Furthermore, we propose an effective pruning technique that substantially reduces the search region for answering the NNH query and yet requires only a constant times the space of the linear R-tree on O.

Nearest Enclosing Circle Problem. In addition, this paper studies a novel geometric problem, called the *nearest enclosing circle* (*NEC*) problem, which is an unavoidable sub problem to obtain the exact answer of the NNH query. Assuming that we eventually find a set of k points that can be covered by a circle with a given radius, the NEC problem addresses the following question: "how to determine the nearest center of the circle to the query point while the circle encloses all such k points?" This simple question, however, is not trivially solvable, and has not been studied so far in the literature. To solve this problem, we observe that the properties of the NEC problem are similar to those of the smallest enclosing circle (SEC) problem that has been well studied in computational geometry, and thereby proposing a linear expected-time algorithm settling the NEC problem.

Contributions. We summarize our main contributions as follows:

- We propose a group version of the NN query in spatial databases, called the nearest neighborhood (NNH) query, which is useful in many applications of spatial databases.
- We devise a fast R-tree based algorithm to process the NNH query based on the incremental retrieval of nearest neighbors. We also devise efficient methods to keep track of clusters of the retrieved nearest neighbors and a pruning technique to reduce the search region of the NNH query.
- As a sub problem, we discover a new geometric problem, called the nearest enclosing circle (NEC) problem, and propose a linear expected-time algorithm for the NEC problem.
- A thorough experimental study is performed to evaluate our query processing algorithm. Experimental results show that our proposed techniques can significantly improve the query performance.

Organization. In Section 2, we first review existing works related to the NNH query, and formalize the problem setting and the NNH query in Section 3. In Section 4, the NEC problem and its solution algorithm are presented. In Section 5, our proposed solution for processing the NNH query is derived. In Section 6, we show experimental results, and finally conclusions are discussed in Section 7.

II. RELATED WORKS

In the database community, the NN or kNN query processing is mainly focused on devising the manner of exploiting the underlying index structures such as the R-tree. An R-tree based NN query processing algorithm is firstly proposed by Nick et al. [6] with requirements from various geographic information systems (GISs). Later, several variants of the NN algorithm using the R-tree are presented in the literature [7], [8], and the most representative one is the best-first NN search invented by Hjaltason et al. [8]. The best-first algorithm maintains a priority queue whose entries are either nodes or points while traversing the R-tree from the root node in a best first manner.

The order of the queue entries is based on the minimum distance from the query, and thereby we can incrementally retrieve nearest neighbors in order. Our NNH query processing algorithm also utilizes the best-first search for the incremental retrieval of NNs.

Among numerous variants of the NN query, one interesting branch is the aggregate nearest neighbor (ANN) query (a.k.a. group nearest neighbor query) which is firstly proposed by Papadias et al. [5] and further studied by others [9], [10], [11]. The goal of ANN is to find the optimal point closest to the set Q of query points instead of one single query point. Even though ANN also tackles the concept of the group in the NN query, the problem setting of the ANN query is exactly the opposite to NNH where a single query point is given and the answer is involved with a group of target points.

Recently, the *group nearest group* (*GNG*) query, which is a spatial query returning a group of points as our NNH query, is proposed by Deng et al. [12]. However, the GNG query is basically a general extension from the ANN query. Instead of returning a single point that is closest to the set of query points, GNN returns a set O' of points such that the sum of all the shortest distances of points in O' from the set Q of query points is minimized. Thus, points in the resulting set O' of GNN do not have to be clustered, which is a major difference from NNH.

In the literature of spatio-textual queries (a.k.a. spatial keyword queries), a novel type of queries, called *collective spatial keyword queries*, are recently proposed by Cao et al. [13] and Long et al. [14]. Also, a similar query, called the *k-nearest group query*, without the textual constraint was studied by Zhang et al. [15]. This type of queries share the motivation similar to the NNH query in that its goal is to find a group of spatio-textual objects that collectively satisfy user requirements. However, in terms of the problem formulation, the NNH query is quite different from the collective spatial keyword query that does not consider any circular region. Therefore, the NNH query cannot be solved by the algorithm for processing the collective spatial keyword query.

The NNH query is also related to the type of spatial queries on finding the optimal location in the sense that the ultimate goal of NNH is to find the nearest location rather than the nearest point. The most related work to NNH is the MaxCRS query which is studied by Choi et al. [16]. The MaxCRS query returns the center location of a circular area covering the maximum number of points, which is similar to our NNH query. The difference of NNH from MaxCRS is twofold: (1) NNH has a query point while MaxCRS does not, and (2) MaxCRS returns the densest location of a circle but NNH finds the nearest location of a circle covering k points.

III. Problem Statement

In this section, we formally describe our problem environment and define the nearest neighborhood (NNH) query. Our problem setting includes:

- a set O of 2D points
- a query point q, which is also a 2D point
- a distance parameter ρ, which is a positive real value
- a cardinality parameter k, which is a positive integer
- a Euclidean distance function between the points p and q, denoted by $dist(p, q)$

Then we define the most important notion *neighborhood*, which represents a region of interest (ROI) containing a particular number of points as follows:

Definition 1 (Neighborhood): Let ρ be the distance constraint and k be the cardinality constraint. Then a neighborhood with respect to ρ and k is a ρ-radius circle enclosing at least k points in O.

We denote the neighborhood with ρ and k as $NH(\rho, k)$. Then the NNH query is defined as:

Definition 2 (Nearest Neighborhood Query):
Given O, q, ρ, and k, the *nearest neighborhood* (*NNH*) query finds the center of the nearest $NH(\rho, k)$ to q.

IV. Nearest Enclosing Circle Problem

Prior to getting down to our query processing algorithm, in this section we first study a sub problem, called the nearest enclosing circle (NEC) problem, which must be tackled for the exact NNH query processing. Let us start with a formal definition of the NEC problem as follows:

Definition 3 (Nearest Enclosing Circle Problem): Given a set P of K points, a ρ-radius circle C_ρ, and a query point q, find the nearest center c of C_ρ to q such that all points in P are enclosed by C_ρ.

It is not difficult to notice that the only difference between NEC and the NNH query is that all points in P should be enclosed by a ρ-radius circle in NEC while k points should be enclosed in NNH. This sub problem is essential not only to find the final answer of the NNH query (i.e., the location of the nearest neighborhood) but also to evaluate different neighborhoods for choosing the nearest one among candidate neighborhoods. Figure 2 shows an instance of the NEC problem where the answer is c; that is, the center of the ρ-radius circle shown with a solid line. Other circles pictured with dotted lines are also enclosing all points, but they are not the answer, as they are not the closest one to q.

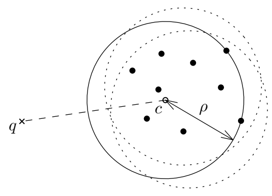


Fig. 2. An example of the NEC problem

Unlike its simple definition, this geometric problem is not trivially solvable. The most straightforward algorithm for the NEC problem is to consider all the feasible combinations of points that can be laid on the boundary of a ρ-radius circle and select the one with the nearest center while enclosing all points in P. The running time of this algorithm is $O(2^K + K \cdot B(K))$, where $K = |P|$ and $B(K)$ is the total number of all possible sets of boundary points of ρ-radius circles. This is because we have to check all the combinations of points to determine whether they can be together in the boundary of a ρ-radius circle, and also check the points in each feasible set to see whether the corresponding ρ-radius circle having the points on its boundary covers all points in P.

It is worth noting that the NEC problem has not been studied even in computational geometry. The problem most related to NEC is the *smallest enclosing circle* (*SEC*) problem, which is a well-known mathematical problem of computing the smallest circle covering all of a given set of points, formally defined as follows:

Definition 4 (Smallest Enclosing Circle Problem): Given a set P of points, find the center of the smallest circle C_{min} such that all points in P are enclosed by C_{min}.

Due to the properties of the SEC, it is known that the SEC problem can be solved in expected-linear time by applying a simple randomized technique [17].

Fortunately, the NEC of P has similar properties to the SEC of P, even though the proofs behind those properties are quite different. This enables us to adapt a randomized incremental algorithm for solving the NEC problem as the SEC problem.

A. Properties of the Nearest Enclosing Circle

Now we investigate the important properties of the NEC, which are key to solving the NEC problem efficiently. We first show the uniqueness of the NEC of a given set P of points in the following lemma, which is important because it implies that the answer of the NNH query is also unique.

Lemma 1: If there exist ρ-radius circles that enclose a set P of K points, then the closest such circle to the query point q is unique:

Proof: The possible area of the center of the ρ-radius enclosing circle of P is the intersection region, denoted by I, of K ρ-radius circles, each of which is centered at a point in P. Such intersection region is convex by the fact that: "*Intersection of convex sets is also convex.* [18]" Also, by the *closest point theorem* [19], which says "*there is a unique closest point in any convex set to another point outside of the convex set*", there is a unique closest point in I to q. This implies that there is a unique center for the closest ρ-radius circle to q. ■

Lemma 1 employs the ρ-radius circle centered at each point in P. Let us use the term *centered-circle* to refer to each of these circles. Thus, unless otherwise noted, it is supposed that each centered-circle has ρ as its radius.

Next we consider the identification of such a unique NEC of P. Thus, the following question needs to be addressed: "how is the NEC of P defined?" This is also related to the number of all possible sets of boundary points (i.e., $|B(K)|$) as mentioned in the straightforward algorithm. About this question, the NEC has one nice property as the following lemma:

Lemma 2: Let $nc(P)$ be the NEC of P. Then $nc(P)$ can be defined by at most two points in P, which are laid on the boundary of $nc(P)$.

Proof: As stated in the proof of Lemma 1, the center of $nc(P)$ is the closest point to q in the intersection region of all the centered-circles of P, which should be either (1) an intersection of two centered-circles, or (2) the intersection of the centered-circle of a point and the line passing through the point and the query point. Therefore, at most two points can define the NEC of P. ■

By Lemma 2, the time complexity of the aforementioned straightforward algorithm turns out to be $O(K^3)$, since $B(K)$ is $O(K^2)$ and we can check all pairs of points in P instead of all subsets of P. This is not still practically efficient.

To apply the randomized technique, the most important fact behind the NEC is as follows: "*If a point $p \in P$ is outside of the NEC of $P \setminus \{p\}$ (i.e., $nc(P \setminus \{p\})$), then p must be laid on the boundary of $nc(P)$.*" This is formally proved as the following lemma:

Lemma 3: Let q be the query point, P be a set of points, R be a possibly empty set of points with $P \cap R = \emptyset$, $nc(P, R)$ be the nearest circle to q enclosing all points in P and having all points in R on its boundary, and $p \in P$. Then the following holds:

1) If $p \in nc(P \setminus \{p\}, R)$, then $nc(P, R) = nc(P \setminus \{p\}, R)$.
2) If $p \notin nc(P \setminus \{p\}, R)$, then $nc(P, R) = nc(P \setminus \{p\}, R \cup \{p\})$.

Proof: Due to the space limitation, we omit the details of this proof. Please refer to our technical report [20][1]. ■

Intuitively, Lemma 3 tells us how to determine the boundary points of the NEC of a given set P of points, which completes the answer of our initial question, i.e., how to define the NEC of P, as stated in Lemma 2. Statement 1 of Lemma 3 says that if a point p is already inside the current NEC, we do not have to change the boundary of the NEC. However, according to Statement 2 of Lemma 3, if a point p is outside of the current NEC, such a point p must be a boundary point of the newly updated NEC including p as well as all the points that have been added so far. Based on these important properties, we can efficiently find the NEC of a given set of points, as explained in the following section.

B. Linear Expected-Time Algorithm for the NEC Problem

Now we are ready to apply the incremental randomized algorithm to the NEC problem. The basic process is to add the points in P one by one in a random order while maintaining the NEC of the set of the points that have been added so far. The details of the process, namely NearestCircle, are presented in Algorithm 1.

Whenever a point p_i is considered, the NEC under consideration is not updated if p_i is already inside the current NEC by Statement 1 of Lemma 3 (Lines 5 - 6). Otherwise, p_i must be in the boundary of the new NEC enclosing p_i together with all points that have been already considered (i.e., $p_1, p_2, ..., p_{i-1}$) by Statement 2 of Lemma 3 (Lines 7 - 8). This invokes another subroutine, namely NearestCircleWithPoint, that computes the NEC of $\{p_1, p_2, ..., p_i\}$ with the additional information that p_i lies on the boundary of the new NEC, which is presented in Algorithm 2. Algorithm 2 is basically the same as Algorithm 1 except for the way of

[1]https://cs.kaist.ac.kr/research/techReport

tackling the case when a point p_j is not inside the current NEC (Lines 6 - 7). Now we can determine, without invoking any subroutines, the NEC of the set of points that have been checked, since at most two points can uniquely define an NEC by Lemmas 1 and 2.

ALGORITHM 1: NEARESTCIRCLE(P, q, ρ)

```
Input: $P$ := a set of points, $q$ := the query point, $\rho$ := a given radius
Output: $NEC$ := the nearest $\rho$-radius circle enclosing all points in $P$
1 $p \leftarrow$ an arbitrary point in $P$
2 $NEC \leftarrow$ the nearest $\rho$-radius circle having $p$ on its boundary
3 Compute a random permutation $p_1, ..., p_{|P|-1}$ of $P \setminus \{p\}$
4 for $i \leftarrow 1$ to $|P| - 1$ do
5     if $p_i$ in $NEC$ then
6         do nothing
7     else
8         $NEC \leftarrow$ NEARESTCIRCLEWITHPOINT($\{p, p_1, ..., p_{i-1}\}$, $p_i$, $q$, $\rho$)
9 return $NEC$
```

ALGORITHM 2: NEARESTCIRCLEWITHPOINT(P', p', q, ρ)

```
Input: $P'$ := a set of points, $p'$ := a point $\notin P'$, $q$ := the query point,
       $\rho$ := a given radius
Output: $NEC$ := the nearest $\rho$-radius circle enclosing all points in $P'$ and
        having $p'$ on its boundary
1 $NEC \leftarrow$ the nearest $\rho$-radius circle having $p'$ on its boundary
2 Compute a random permutation $p_1, ..., p_{|P'|}$ of $P'$
3 for $j \leftarrow 1$ to $|P'|$ do
4     if $p_j$ in $NEC$ then
5         do nothing
6     else
7         $NEC \leftarrow$ the nearest $\rho$-radius circle having $p'$ and $p_j$ on its boundary
8 return $NEC$
```

Theorem 1: Given a set P of points, a query point q, and a given radius ρ, Algorithm 1 correctly returns the ρ-radius circle enclosing P nearest to q.

Proof: This follows from Lemmas 1, 2, and 3. ■

Theorem 2: The expected running time of Algorithm 1 is $O(K)$, where $K = |P|$.

Proof: This running time analysis is almost the same as that presented in [17]. The basic idea is that Algorithm 2 is invoked with a probability that p_i is not inside the current NEC, which is $2/i$ since at most two boundary points can update the current NEC. Therefore, the expected cost of Algorithm 1 is linear to K since the cost of Algorithm 2 is also linear to i with a probability of $2/i$. ■

V. NEAREST NEIGHBOR*hood* QUERY PROCESSING

In this section, we explain our solution for processing the NNH query. First, we examine the hardness of NNH by introducing an exhaustive solution. Next, we propose an R-tree based query processing algorithm using the incremental retrieval of nearest neighbors. Finally, we present a pruning technique that utilizes the augmented R-tree with additional information about the validity of each point in terms of the closeness constraint.

A. Hardness of the NNH Problem

Obviously, the simplest approach for NNH is to consider all the combinations of at least k points and then find the closest one. Alternatively, we can preprocess all these possible subsets of O in an extremely huge index structure and find the nearest one to the query point from the index structure.

A natural question to be addressed is: "how many combinations of points should we preprocess?" The answer seems to be $O(2^N)$ at first glance, but actually it is $O(N^3)$. This is because it suffices to consider only the *circular convex* sets. Thus, if points p_1 and p_2 are very far away from each other, then they do not have to be considered without the points in the middle of them. More specifically, all we have to do is to preprocess all the smallest enclosing circles (SECs) that can be generated from O. It is proved that each SEC is unique and can be defined by at most three boundary points [17]. Therefore, the number of all the possible SECs of O is at most N^3.

Once the nearest *qualified* SEC (i.e., it covers at least k points and its radius is not larger than ρ) is found, the corresponding NEC can be computed in linear time using our algorithm presented in Section IV-B. Note that, after the first SEC is found, we may have to find the next nearest qualified SEC as long as its corresponding NEC can be closer to q than the NEC previously found.

Unfortunately, in a practical situation, this $O(N^3)$ number of cases is too large to be preprocessed in an index structure as well as processed on the fly.

B. R-tree based NNH Query Processing Algorithm

In practice, only a constant times the number of points is reasonable to be stored in a preprocessed index structure. This leads us to use a basic index structure on O such as the R-tree (or any kinds of spatial index structures having the linear space complexity and supporting the incremental kNN search).

Similar to the exhaustive space approach, we only consider the qualified SECs satisfying the constraints with ρ and k. However, we now maintain SECs on the fly by incrementally retrieving nearest points to q, instead of choosing the nearest one among all the possible SECs stored in the preprocessed structure. The overall steps of the query processing algorithm are as presented in Algorithm 3.

This algorithm addresses the following challenging questions:

- How to efficiently retrieve all SECs for each p_{next} (Line 5)
- How to efficiently update SECs (Line 7)

1) SEC Retrieval: Let us first deal with the first issue, i.e, how we can efficiently find all SECs for each nearest point being retrieved (i.e., p_{next}). The most straightforward approach is scanning all the SECs constructed so far and checking whether there exists a point in the SECs intersecting the 2ρ-radius circle centered at p_{next}. This method requires $O(T)$ time for each retrieval of p_{next}, where T is the number of SECs being maintained.

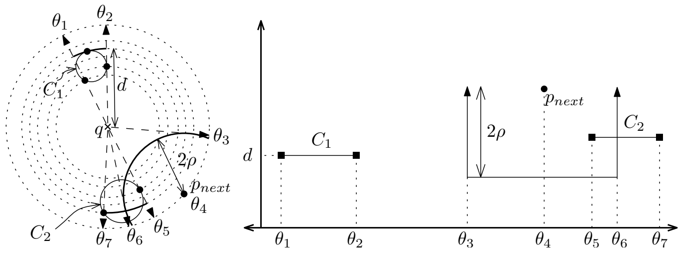
A more improved approach is maintaining the R-tree-like structure on SECs, and thereby retrieving the SECs which overlap the 2ρ-radius circle centered at p_{next} by performing a range search on the R-tree. This is basically rooted on a window range search in the Cartesian coordinate system.

ALGORITHM 3: Overall Algorithm

```
Input: $O$ := a set of points, $q$ := a query point, $\rho$ := a distance parameter,
    $\rho$ := a cardinality parameter
Output: $NNH(\rho, k, q)$ := the nearest $NH(\rho, k)$ to $q$
1  $\tau \leftarrow \infty$, $NNH(\rho, k, q) \leftarrow$ nil
2  repeat
3      $p_{next} \leftarrow$ retrieve the next nearest point to $q$
4      $S \leftarrow$ retrieve all SECs which can be expanded to a larger circle (but
       smaller than a $\rho$-radius circle) including $p_{next}$ or split into new
       maximal SECs including $p_{next}$
5      foreach $C \in S$ do
6          Update $C$ for $p_{next}$ to be included
7          if the number of points covered by $C$ is not less than $k$ then
8              $NEC \leftarrow$ the nearest circle to $q$ enclosing all points covered
               by $C$
9              $\tau' \leftarrow$ the distance between $NEC$ and $q$
10             if $\tau > \tau'$ then
11                 $\tau \leftarrow \tau'$, $NNH(\rho, k, q) \leftarrow NEC$
12 until There cannot exist a closer NEC to q than $NNH(\rho, k, q)$
13 return $NNH(\rho, k, q)$
```

Unfortunately, it is proved that the optimal index structure, the k-d tree, still requires $O(\sqrt{N} + K)$ time to process the window range search in the $2D$ space, where N is the total number of elements, and K is the number of elements to be returned [17].

Our strategy is to transform the coordinate system of objects (i.e., SECs), and thereby reduce the time complexity for the task of finding SECs. We observe that each SEC can be identified by its distance from the query point and the angle range of its corresponding set of points, which naturally results in the Polar coordinate system. Figure 3 shows an example of transforming SECs in the Cartesian coordinate system to those in the Polar coordinate system. Consider the SEC C_1 including three points that have been retrieved. It can be represented as an arc in the Polar coordinate system in the form of $C_1 =< d, [\theta_1, \theta_2] >$. Whenever p_{next} is retrieved, we find all the SECs C_i's whose arcs in the Polar coordinate system satisfy the following two conditions: (1) $dist(q, p_{next}) - 2\rho < C_i.d$ and (2) $C_i.[\theta_1, \theta_2]$ overlaps the angle range of the 2ρ-radius circle centered at p_{next}. In Figure 3, only C_2 will be retrieved, as its one point can be enclosed by a ρ-radius circle together with p_{next}.



(a) Cartesian coordinate system (b) Polar coordinate system

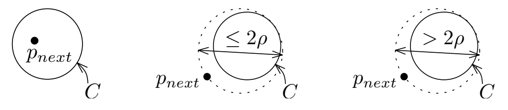
Fig. 3. Transformation of the coordinate system

The benefit of this transformation is that the window range search is now transformed into a *3-sided range search* which is easier to solve. Note that one side is open in the condition (1) while both two sides are closed in the condition (2). This 3-sided range search can be seen as a 1D range search on the dimension of angles with an additional constraint on the lower bound of distances from the query point q.

It is known that the 3-sided range search can be solved in $O(\log N + K)$ using the *priority search tree* [21]. Even though the priority-search tree works on the set of points with priority values while we are interested in the set of angle intervals with priority values (i.e., distances from the query point), we can still utilize the priority-search tree on the set of end points of intervals. This is due to the fact that at least one end point of any intervals intersecting the queried range should also be included in the range.

2) SEC Maintenance: After finding all the SECs that can be combined with the next nearest neighbor (i.e., p_{next}), we should update the SECs accordingly. Thus, for each SEC that intersects the 2ρ-radius circle centered at p_{next}, we should check whether the set of points of the SEC along with p_{next} can be enclosed by a ρ-radius circle.

For the clarity of our argument, we first consider the cases arising most frequently, where the cardinality of the set of points covered by each SEC is less than k. Let C be an SEC retrieved for p_{next} and P_C be the set of points covered by C, where $|P_C| < k$. Then the following cases arise when updating C based on p_{next}, which are also shown in Figure 4.



(a) Case 1 (b) Case 2 (c) Cases 3 and 4

Fig. 4. Cases arising when updating SECs

Case 1. p_{next} is inside C.
This is the simplest case, and it suffices to insert p_{next} into C without any modification of C.

Case 2. $\{p_{next}\} \cup P_C$ can be enclosed by a ρ-radius circle.
In this case, there are two issues to be addressed. The first one is how to check whether $\{p_{next}\} \cup P_C$ can be enclosed by a ρ-radius circle, and the second one is how we can update C for p_{next} to be also included in C. Of course, both issues can be simply addressed by computing a new SEC of $\{p_{next}\} \cup P_C$.

For the first issue, there is an easy condition to test whether p_{next} and P_C can be together in a ρ-radius circle as follows:

$$dist(p_{next}, C) + C.r \leq 2\rho,$$

where $dist(p_{next}, C)$ is the distance between p_{next} and the center of C and $C.r$ is the radius of C. If this condition is satisfied, we can guarantee that p_{next} and P_C can be enclosed by a ρ-radius circle without the computation of the new SEC.

However, due to the second issue, it is still necessary to compute the SEC of p_{next} and P_C. Even though computing the SEC of a given set of points only requires linear time [22], it is still expensive in the sense that such computation should be performed for every SEC relevant to p_{next}, whenever p_{next} is retrieved. Therefore, instead of maintaining exact SECs, we maintain approximate SECs with a reasonable bound, which can be computed and updated in a constant time. To this end, we exploit the MBR of P_C, and let the approximate SEC $\hat{C}$ be the SEC enclosing all the corner points of the MBR of P_C.

The approximation bound of this simple approach using the MBR is proved to be $\sqrt{2}$ as follows:

Lemma 4: Let P be a set of points, C be the exact SEC of P, and $\hat{C}$ be the approximate SEC of P that is the SEC of the corner points of the MBR of P. Then $\hat{C}.r \leq \sqrt{2} \cdot C.r$.

Proof: One important property of the MBR of P is that there must be at least one point lying on each edge of the MBR. Therefore, the exact SEC of P should touch every edge of the MBR to enclose such points lying on the edges. This implies that the diameter of C cannot be smaller than the length of the longest edge of the MBR. Also, it is easy to know that the diameter of $\hat{C}$ is the length of the diagonal of the MBR. Since the diagonal of a rectangle cannot be $\sqrt{2}$ times longer than its longest edge, $\hat{C}.r$ should be smaller than or equal to $\sqrt{2} \cdot C.r$. ■

Using the approximate SEC also helps to test the question in the first issue. Thus, we can quickly check whether the SEC of the corner points of the previous MBR and p_{next} is smaller than or equal to a ρ-radius circle without computing the exact SEC.

Case 3. $\{p_{next}\} \cup P'$ can be enclosed by a ρ-radius circle, where $P' \subset P_C$ is the set of the points in P_C that are located in the 2ρ-radius circle centered at p_{next}.
In this case, computing the exact SEC of P', which is separated from C, is unavoidable. Also, this newly generated SEC must be inserted to the priority search tree for further references by the next nearest neighbor.

Case 4. $\{p_{next}\} \cup P'$ cannot be enclosed by a ρ-radius circle, where P' is the same as P' in Case 3.
This is the most difficult case arising when only subsets of P' can be enclosed by a ρ-radius circle including p_{next}. In this case, we need to compute all new *maximal* SECs from C that are not contained by any other SECs. A natural question is raised as follows: "How many new maximal SECs can be generated at most for each update?" If the number of newly generated maximal SECs is not reasonably small, the entire query processing cost will become fairly high due to the excessive creations of new SECs for this case.

Fortunately, we prove that the maximum number of newly separated SECs from C is at most $O(k)$ in the following lemmas:

Lemma 5: Let C'_i be a maximal SEC including p_{next} separated from C. Then the center of C'_i should be located in a minimal intersection region I'_i such that (1) I'_i is covered by the centered-circles of some points in P' together with the centered-circle of p_{next}, and (2) I'_i does not contain any sub intersection region covered by other centered-circles.

Proof: Let $P'_i \subset P'$ be the set of points whose centered-circles are covering I'_i. When the center of a ρ-radius circle is placed in I'_i, which is inside the centered-circle of p_{next}, the circle must contain all points in $P'_i \cup \{p_{next}\}$. Furthermore, since there is not any sub intersection region in I'_i, that is, I'_i is minimal, any ρ-radius circles whose centers are in I'_i cannot contain any other points besides $P'_i \cup \{p_{next}\}$. Thus, such circles are maximal in terms of the set of enclosed points. Finally, the center of the SEC of $P'_i \cup \{p_{next}\}$ must be in I'_i. Otherwise, the SEC cannot cover all points in $P'_i \cup \{p_{next}\}$ since the radius of the SEC is not larger than ρ. ■

Lemma 5 indicates that the upper bound of the number of maximal SECs that can be separated from C is identical to the number of all the minimal intersection regions inside the centered-circle of p_{next}. In the centered-circle of p_{next}, the intersecting parts of other centered-circles can be seen as *arc*s as shown in Figure 5. These arcs residing in the centered-circle of p_{next} have the following property:

Lemma 6: Let $C(p_i)$ be the centered-circle of the point p_i. Then any pair of arcs inside $C(p_{next})$, which originate from the centered-circles of P', can intersect at most once in $C(p_{next})$.

Proof: Suppose by contradiction that there exists a pair of arcs $\widehat{x}$ and $\widehat{y}$ that intersect twice in $C(p_{next})$. Since all points in P_C are covered by C, all the centered-circles of $P' \subset P_C$ should cover one intersection region, that is, $\bigcap_{p_i \in P'} C(p_i) \neq \emptyset$. Also, the region of $\bigcap_{p_i \in P'} C(p_i)$ should be outside of $C(p_{next})$ since $\{p_{next}\} \cup P'$ cannot be enclosed by a ρ-radius circle by the definition of Case 4. This is a contradiction to our first assumption implying that the entire intersection region of the original centered-circles of $\widehat{x}$ and $\widehat{y}$ is inside $C(p_{next})$. ■

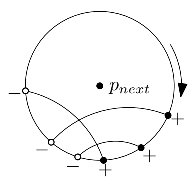


Fig. 5. Arcs in $C(p_{next})$

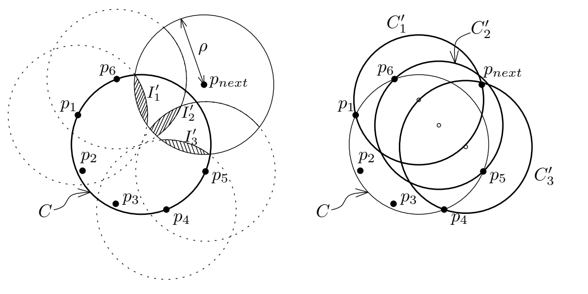
Now we are ready to answer our first question as the following theorem:

Theorem 3: For each retrieval of p_{next}, the number of newly created maximal SECs that are separated from one existing SEC is at most $O(k)$.

Proof: By Lemma 5, to prove this theorem, it suffices to check whether the number of minimal intersection regions in the centered-circle of p_{next} (i.e., $C(p_{next})$) is at most $O(k)$. Since all the arcs residing in $C(p_{next})$ can meet only once by Lemma 6, each minimal intersection region requires two end points of arcs in the boundary of $C(p_{next})$. End points of arcs in $C(p_{next})$ can be classified as a positive or negative point by the order of the clockwise direction as pictured in Figure 5. It is easy to notice that each minimal intersection region consists of a pair of one positive and one negative end points. Once an end point is consumed for defining a minimal intersection region, it cannot further contribute to other minimal intersection regions. Since there are less than $2(k-1)$ end points in $C(p_{next})$, the number of minimal intersection regions is at most $O(k)$. ■

The remaining issue is how to construct such linear number of new SECs from C efficiently. Our findings in the lemmas and theorem above derive the following algorithm for this task:

1) Find all intersection points, which are end points of arcs, in $C(p_{next})$.



(a) Centered-circles (b) Separated maximal SECs

Fig. 6. Finding maximal SECs separated from an existing SEC

2) Sort the set of end points in the clockwise direction.
3) While scanning the sorted set of end points, maintain a set P_{temp} of points and its MBR by performing the following operations:
   a) Whenever encountering a positive end point, insert the corresponding point in P' into P_{temp}
   b) Whenever encountering a negative end point, delete the corresponding point from P_{temp}, and create a new approximate SEC of P_{temp} only if the previous end point is positive.

The worst-case time complexity of this algorithm is $O(k \log k)$, which is dominated by the sorting process in Step 2.

Example 1: Figure 6 shows an example of constructing new maximal SECs from C. In this example, P_C and P' are given as $\{p_1, p_2, p_3, p_4, p_5, p_6\}$ and $\{p_1, p_4, p_5, p_6\}$, respectively. Thus, the distances of p_{next} from p_2 and p_3 are larger than 2ρ. Figure 6(a) illustrates the centered-circles of P' (dotted line), the centered-circle of p_{next} (solid line), and C (bold line). In the centered-circle of p_{next}, there are three minimal intersection regions, I'_1, I'_2, and I'_3, which correspond to the three maximal SECs in Figure 6(b), C'_1, C'_2, and C'_3, including $\{p_1, p_6, p_{next}\}$, $\{p_5, p_6, p_{next}\}$, and $\{p_4, p_5, p_{next}\}$, respectively.

Now we discuss the cases for SECs covering k or more points, i.e., $|P_C| \geq k$, even if they rarely happen. For *Case 1* and *Case 2*, we can apply the same procedures as before. However, in *Case 3* and *Case 4*, we should find all the intersection regions covered by at least k centered-circles as well as minimal intersection regions covered by less than k centered-circles. The number of such intersection regions is still linear to $|P_C|$ as before because they still require a pair of positive and negative end points that can contribute to only one intersection region. Let t be $|P_C|$. Then all our arguments remain valid by substituting t for k.

3) Termination Condition: Our query processing algorithm is based on the continuous retrieval of the next nearest neighbor, and hence we should stop the retrieval when it is guaranteed that there is no additional possible nearest neighborhood beating the current best (i.e., closest) neighborhood. Let τ be the current nearest distance so far. We can safely stop finding next nearest neighbors when the following condition holds:

$$dist(q,\ p_{next}) \geq \tau + \rho,$$

This condition is based on the following lemma:

Lemma 7: Let P and P' be sets of points, and let $nc(P)$ be the NEC of P with respect to q. If $P \subseteq P'$, then $dist(nc(P), q) \leq dist(nc(P'), q)$.

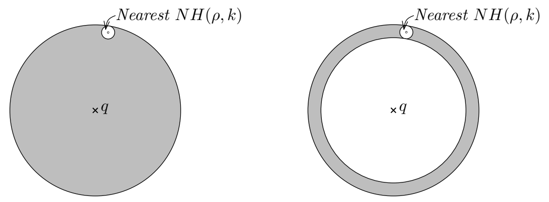
Proof: Let I be the intersection region of centered-circles of P, and I' be that of P'. Then $I \supseteq I'$. There should be a point c_0 in I which is closest to q, and c_0 is the center of $nc(P)$. Since we cannot find any other point in I' that is closer to q than c_0, the center of $nc(P')$ cannot closer to q than the center of $nc(P)$. ■

By Lemma 7, we can guarantee that once the nearest $NH(\rho, k)$ is found, it cannot get closer by adding a new point to the $NH(\rho, k)$.

C. Reducing the Search Region

The main overhead of our R-tree based NNH query processing algorithm lies in the process of maintaining SECs. This is even worse when the search region of the algorithm becomes larger. The search region of our algorithm can be represented as a circle with a radius as large as the distance from the query point to the center of the nearest $NH(\rho, k)$ plus ρ according to the termination condition of the algorithm. Until we encounter the first point belonging to the nearest $NH(\rho, k)$, any other points that have been retrieved and maintained are indeed not necessary. It would be much better to discard these unnecessary points, which leads to the substantial reduction of the search region.

This motivates us to take into account an ideal search region, which can be represented as a ring area whose inner circle is the circle with a radius of the distance between the query and the nearest feasible point that belongs to a $NH(\rho, k)$, and whose outer circle is the circle with a radius as large as the radius of the inner circle plus 2ρ. The search region of our R-tree based NNH algorithm and the ideal search region are illustrated in Figure 7.



(a) Search region of the algorithm (b) Ideal search region

Fig. 7. Search region of the NNH query

Our purpose is to reduce the search region needed in the current algorithm to the ideal one. To this end, we observe the fact that k is much smaller than N in practice. This gives an opportunity to allocate more space in the linear R-tree on O for storing additional information to prune unnecessary points.

Theoretically, this ideal search region is possible if we associate each point p with an array of radii, denoted by $r_i(p)$'s, such that $r_i(p)$ is the radius of the smallest circle enclosing p and any other $i - 1$ points, for $i \in [1, \kappa]$, where κ is the maximum of k values. Similarly, for each node in the R-tree, we augment the minimum values of $r_i(p)$'s of all the points residing in the node. By doing this, whenever we find the next nearest neighbor p_{next}, it suffices to consider the points p's satisfying the following condition: $r_k(p) \leq \rho$. Otherwise, we can safely discard the point since it cannot belong to a $NH(\rho, k)$. This augmented R-tree only requires $O(\kappa N)$ space, which is reasonable in practice in the sense that $\kappa \ll N$.

Unfortunately, a major difficulty for the above approach lies in the fact that computing the $r_i(p)$ value is an expensive task whose running time turns out to be $O(N \log^2 N)$ [23]. Therefore, the entire preprocess can be done in $O(\kappa N^2 \log^2 N)$ time. Although this process will be performed prior to the query processing phase, its high time complexity makes it infeasible to do in most practical situations.

We remedy this issue by using approximate $\hat{r_i}(p)$ values instead of exact values. For this purpose, let us first consider the following inequalities:

Lemma 8: Let $nn_i(p) \in O$ be the i-th nearest neighbor of $p \in O$. Then the following inequalities hold:

$$\frac{1}{2}\ dist(p, nn_{i-1}(p)) \leq r_i(p) \leq dist(p, nn_{i-1}(p))$$

Proof: Let us first deal with the first inequality. Suppose that there exists a circle C_i containing p and other $i-1$ points such that the radius of C_i, denoted by $C_i.r$, is smaller than $dist(p, nn_{i-1}(p))/2$. This means there are $i-1$ points that are at most $2 \cdot C_i.r$ away from p where $2 \cdot C_i.r < dist(p, nn_{i-1}(p))$. This is a contradiction to the definition of $dist(p, nn_{i-1}(p))$.

The second inequality is due to the fact that there must be $i-1$ points within a circular region whose center is p and radius is $dist(p, nn_{i-1}(p))$. ■

Based on Lemma 8, we set $\hat{r_i}(p)$ to be $dist(p, nn_{i-1}(p))/2$, which is indeed a 2-approximation of $r_i(p)$. Computing $\hat{r_i}(p)$ values is much cheaper than $r_i(p)$ values, since it suffices to perform the κNN query for each p. This task is known as the *all nearest neighbors* problem that can be solved in $O(\kappa N \log N)$ time [24], which is much smaller than $O(\kappa N^2 \log^2 N)$.

VI. Experimental Study

In this section, our objective is to check whether our query processing algorithm can be done in a reasonably small running time. Since there is no existing algorithm processing the NNH query that can be compared with our algorithm, we focus on showing the effectiveness of our techniques, that is, whether they can improve the overall query processing performance, particularly in terms of the cost of maintaining SECs on the fly.

A. Setup

Algorithms. We use the following different versions of the proposed query processing algorithm presented in Sections V-B and V-C:

BRUTEFORCE. This algorithm uses a simple list to maintain SECs. Thus, for every retrieval of an NN point, it scans the list, and thereby finding the SECs intersecting the retrieved NN point.

CARTESIAN. This algorithm uses an in-memory R-tree to maintain SECs. Each SEC is stored in the in-memory R-tree in the form of an MBR. For every retrieval of an NN point, it performs a window range search on the R-tree to find the SECs intersecting the 2ρ-radius circle centered at the retrieved NN point.

POLAR. This algorithm uses the priority search tree to maintain SECs. As mentioned in Section V-B1, each SEC is maintained in the form of an arc, which consists of an angle interval and the distance from the query point. We implement the priority search tree on points represented by the angle and the distance from the query point, and store two end points of each arc in the tree. For every retrieval of the NN point, it performs a 3-sided range search on the priority search tree to find all the relevant SECs.

POLARREDUCED. This algorithm is the same as POLAR except that it uses the augmented R-tree presented in Section V-C thereby reducing the search region of POLAR close to the ideal search region. We set the maximum value of k, denoted by κ, to be 64.

The straightforward solution presented in Section V-A is not feasible to process with a large dataset, and hence it cannot be compared with other algorithms[2].

Datasets. Our experiments are based on three datasets in the real world, which are namely NE, RR, and CAS including 123,593, 257,942, and 196,902 points of interest. All of them are originated from the TIGER project at the US Census Bureau and can be downloaded from the Chorochronos website[3] (formerly the R-tree Portal). It is worthwhile to note that these datasets have been widely used in plenty of existing works on spatial databases. The data space has a length of 10^9 for each dimension, and the coordinates of each point are normalized accordingly.

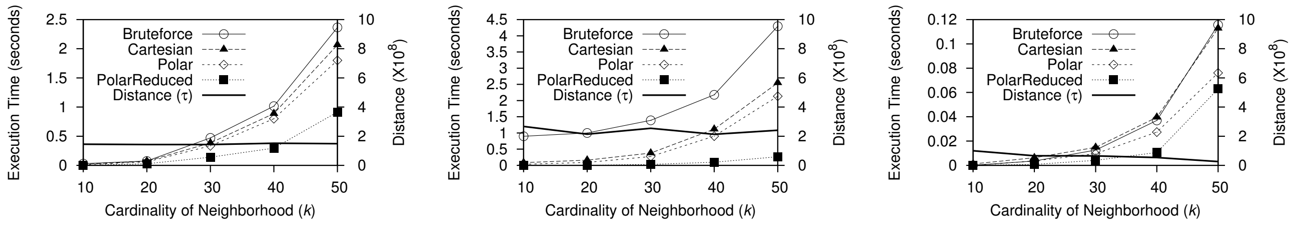
Parameters. There are several parameters that can affect the execution time of the NNH query. The first category is the basic query parameters of NNH, ρ and k. It is expected that larger ρ and k will increase the running time for processing the query. Another interesting parameter is the distance between the query point and the nearest neighborhood finally returned, denoted by τ. Similarly, a larger τ might increase the execution time.

Basically, when we vary one parameter, the other parameters should be fixed. However, this is not easy in the NNH query since there is a correlation between ρ and τ or between k and τ. Thus, we cannot fix both ρ and τ when varying k. Similarly, when varying ρ, it is not feasible to fix both k and τ.

To tell the conclusion first, for all three real datasets, the query performance was more affected by τ than ρ. In this sense, when varying k, we attempt to fix τ values by accordingly adjusting ρ values. Also, when varying ρ (τ), we fix k values and do not care τ (ρ) values. For the experiment on varying τ, with several target τ values for a fixed k, we

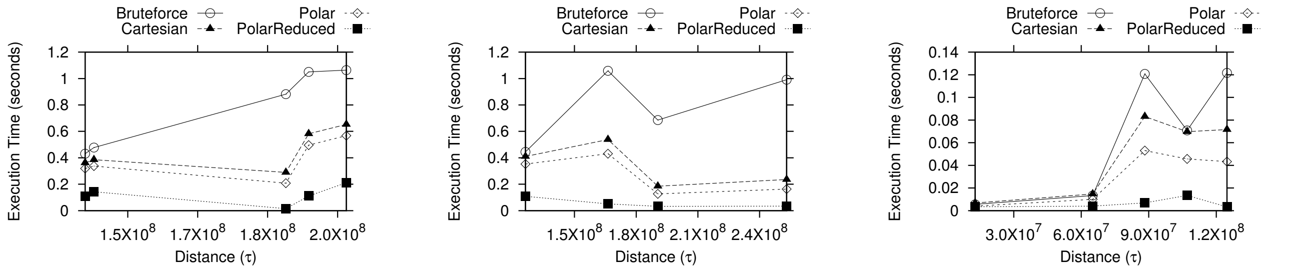
[2]With a toy dataset including only 1000 points, its preprocessed structure was built in about 6 days.

[3]http://www.chorochronos.org



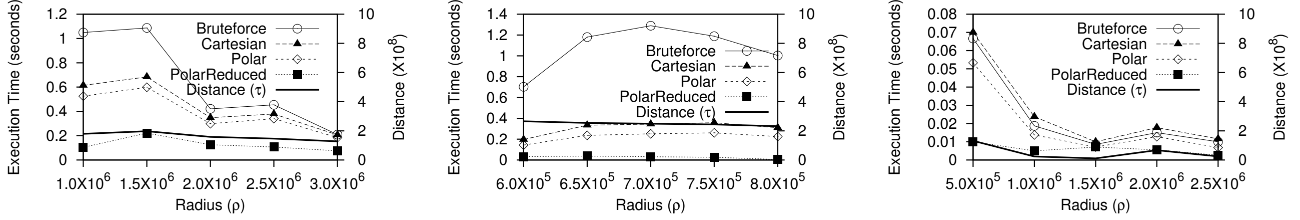
(a) NE (b) CAS (c) RR

Fig. 8. Execution time with varying cardinalities of neighborhood, k



(a) NE (b) CAS (c) RR

Fig. 9. Execution time with varying distances from the query point, τ



(a) NE (b) CAS (c) RR

Fig. 10. Execution time with varying radii of neighborhood, ρ

find appropriate ρ values by performing our algorithm tens of times (once per random query), and then reversely use the found ρ values to do the experiments on varying τ.

Environment. All the algorithms are implemented in C++, and all the experiments are conducted on a PC running Linux (Ubuntu 13.10) equipped with Intel Core i7 CPU 3.4GHz and 16GB memory.

B. Comparison of the Running Time

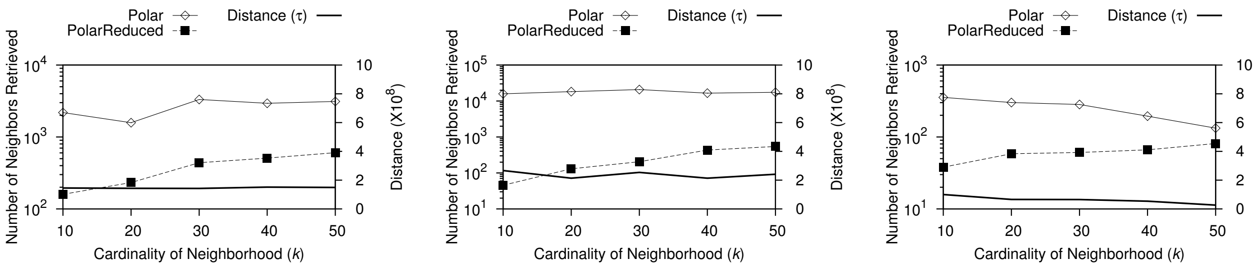
For the first experiment, we compare the execution time that effectively reveals the overall query performance in the sense that our algorithms cannot be regarded as I/O dominant tasks. Each time is average over hundreds of query results with randomly selected query points.

Effect of the Cardinality (k) of the Nearest Neighborhood. Figure 8 shows the execution time when varying k while trying to minimize the effect of τ. Even though τ cannot be exactly fixed due to randomly selected query points, we could keep τ values relatively stable by adjusting ρ values. As expected, the running time of each algorithm increases as k increases. Our first interest is to check whether our technique of transforming the coordinate system is effective to reduce the query time. Based on the results of CARTESIAN, BRUTEFORCE, and POLAR, POLAR is upto 27.4 and on the average 4.2 times faster than BRUTEFORCE, and upto 2.5 times and on the average 1.4 times faster than CARTESIAN. This result shows that the retrieval and maintenance of SECs based on the Polar coordinate system is effective in improving the query performance. Next, to investigate the effectiveness of our pruning technique presented in Section V-C, we compare the result of POLARREDUCED with those of other algorithms. In all cases, POLARREDUCED outperforms other algorithms, which is upto 587.5 times faster than BRUTEFORCE and upto 52.7 times faster even compared with POLAR. On the average, POLARREDUCED is 57.0 times faster than BRUTEFORCE, and 8.6 times faster than POLAR.

An interesting point in this result is that CARTESIAN is even slower than BRUTEFORCE in some cases such as the result shown in Figure 8(c). This is because the cost of updating the in-memory R-tree containing SECs is higher than that of updating the simple list used by BRUTEFORCE, even if the retrieval of relevant SECs in the R-tree might be faster.

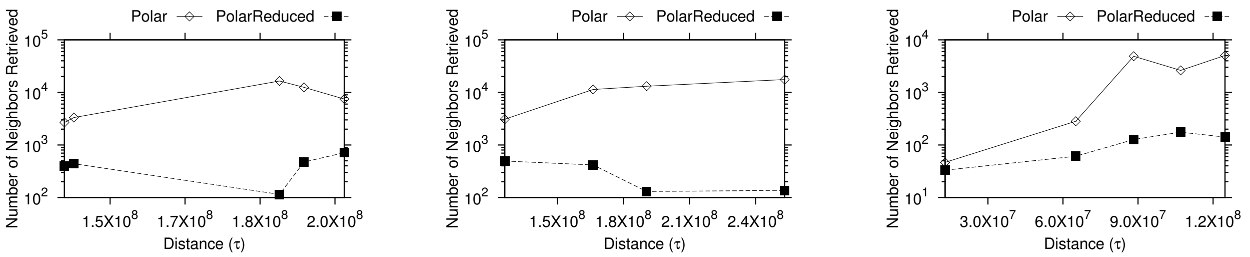
Effect of the Distance (τ) of the Nearest Neighborhood. Figure 9 shows the execution time when varying τ with a fixed $k = 30$. The result shows that the correlation between τ and the execution time is not dramatically high, even though its overall trend can be seen that the bigger τ, the longer execution time. This is because distributions of our real datasets are far from the uniform distribution. Thus, our real datasets have plenty of empty or sparse areas containing only few points, along with relatively dense areas containing many points.

Similar to the result in Figure 8, POLAR outperforms



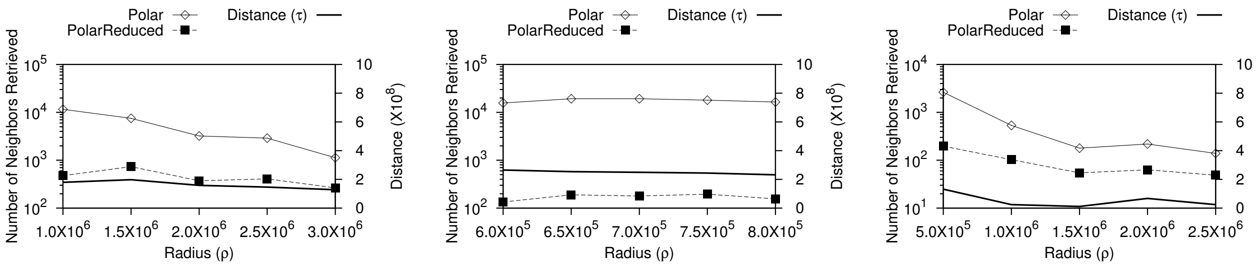
(a) NE (b) CAS (c) RR

Fig. 11. Number of neighbors retrieved with varying cardinalities of neighborhood, k



(a) NE (b) CAS (c) RR

Fig. 12. Number of neighbors retrieved with varying distances from the query point, τ



(a) NE (b) CAS (c) RR

Fig. 13. Number of neighbors retrieved with varying radii of neighborhood, ρ

CARTESIAN and BRUTEFORCE in all cases, and POLARREDUCED shows the best query performance. Furthermore, the difference of the execution time of POLARREDUCED from those of others tends to grow when τ increases, which well explains that the area differences of the search regions of POLARREDUCED and others increase as τ increases (recall Figure 7).

Effect of the Radius (ρ) of the Nearest Neighborhood. Figure 13 shows the execution time when varying ρ with a fixed $k = 30$. Differently from the expectation that a lager ρ will increase the execution time, the execution time decreases as ρ increases. This is because a bigger ρ tends to decrease τ (even if the correlation is not very stable, depending on distributions of real datasets), and therefore the execution time decreases for such smaller τ values. Thus, as mentioned earlier, τ is a more dominant factor on the query performance than ρ.

Overall ratings of the algorithms are the same as before in this experiment. In every case, POLARREDUCED is the best, and followed by POLAR. Note that the case of large ρ values can be interpreted as a trivial case, meaning that k is too small compared with ρ. This is the reason why the performance gap is tiny in some datasets when ρ values are large.

C. Comparison of the Number of Neighbors Retrieved

Another performance metric employed in our experiments is the total number of points that are retrieved until the final nearest neighborhood is found. This is the primary metric for showing the effectiveness of our pruning technique presented in Section V-C. Since the search regions of BRUTEFORCE, CARTESIAN, and POLAR are the same, we only use POLAR and POLARREDUCED for this experiment.

Effect of the Cardinality (k) of the Nearest Neighborhood. Figure 11 shows the number of retrieved neighbors with various k and stable τ. As expected, POLARREDUCED probes a much smaller number of points than POLAR does. Note that the y-axis is in the log scale. On the average, POLAR requires 912.0 times k points retrieved until the nearest neighborhood is found while POLARREDUCED needs only 23.8 times k points. This result justifies that our approximate $\hat{r_i}(p)$ values are fairly effective to reduce the search region and thereby improving the overall query performance as shown in Figure 8.

One minor drawback is that $\hat{r_i}(p)$ seems to get less accurate as k increases, which can be deduced from the result that the performance gap gets smaller for a larger k.

Effect of the Distance (τ) of the Nearest Neighborhood. Figure 12 shows the number of retrieved neighbors when varying τ with a fixed $k = 30$. As with the graphs in Figure 11, the y-axis is set in the log scale. In a macro view, when τ increases, the performance gap gets larger due to the fact that the area of search region is mathematically proportional to the square of its radius, which is τ^2. However, the result does not show a strong correlation between τ and the number of neighbors retrieved, and it shows some cases where the number of processed neighbors decreases even when τ increases. This also well explains why the fluctuation in the result in Figure 9 is high. Instead, it can be seen that the overall query performance is more related to the number of retrieved points by the observation that the graphs in Figure 9 and their corresponding graphs in Figure 12 show similar patterns to each other.

Effect of the Radius (ρ) of the Nearest Neighborhood. Figure 13 shows the number of retrieved neighbors when varying ρ with a fixed $k = 30$. Overall, graphs show the pattern similar to those of Figure 10. Also, we can again observe that the trend of τ values, which are varied by ρ values, is similar to that of the number of retrieved neighbors. Thus, when varying ρ and fixing k, the performance is influenced by τ values rather than ρ value.

VII. Conclusions

In this paper, we proposed a group version of the NN query, called the NNH query, inspired by the requirement of finding the nearest group of points that are closely located, instead of the nearest single point. For the NNH query processing, we discovered an interesting geometric problem, called NEC, that has not been studied in the literature of computer sciences. We found and proved key properties of the NEC similar to those of the SEC so that the incremental randomized algorithm can be adapted to solve the NEC problem. Moving back to the main goal of the present paper, we proposed an R-tree based NNH query processing algorithm and developed efficient methods of maintaining and updating SECs on the fly based on some theoretical findings with respect to SECs and NECs. Also, we grasped the fact that an important factor affecting the overall query performance is the size of the search region required to process the NNH query, and presented a way of augmenting the R-tree with additional information that can considerably reduce the search region. All the experimental results based on real datasets justified that our techniques are effective to improve the query performance.

In closing, we would like to briefly discuss about our future work. In this work, we have only considered circular clusters, but non-circular clusters such as rectangular clusters are also useful in various applications. Our next plan is to devise similar yet different techniques for finding the nearest non-circular neighborhood.

Acknowledgment

This work was supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract UD140022PD, Korea.

References

[1] R. Lübke, D. Schuster, and A. Schill, "Mobilisgroups: Location-based group formation in mobile social networks," in *Workshop Proceedings of IEEE International Conference on Pervasive Computing and Communications*, 2011, pp. 502–507.

[2] S. Srivastava, S. Ahuja, and A. Mittal, "Determining most visited locations based on temporal grouping of gps data," in *Proceedings of the International Conference on Soft Computing for Problem Solving (SocPros)*, 2011, pp. 63–72.

[3] S. Tiwari and S. Kaushik, "Extracting region of interest (roi) details using lbs infrastructure and web-databases," in *Proceedings of the 13th IEEE International Conference on Mobile Data Management (MDM)*, 2012, pp. 376–379.

[4] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *Proceedings of the 18th International Conference on World Wide Web (WWW)*, 2009, pp. 791–800.

[5] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, "Aggregate nearest neighbor queries in spatial databases," *ACM Trans. Database Syst. (TODS)*, vol. 30, no. 2, pp. 529–576, 2005.

[6] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *Proceedings of ACM International Conference on Management of Data (SIGMOD)*, 1995, pp. 71–79.

[7] K. L. Cheung and A. W.-C. Fu, "Enhanced nearest neighbour search on the r-tree," *SIGMOD Record*, vol. 27, no. 3, pp. 16–21, 1998.

[8] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM Trans. Database Syst. (TODS)*, vol. 24, no. 2, pp. 265–318, 1999.

[9] F. Li, B. Yao, and P. Kumar, "Group enclosing queries," *IEEE Trans. Knowl. Data Eng. (TKDE)*, vol. 23, no. 10, pp. 1526–1540, 2011.

[10] H. Li, H. Lu, B. Huang, and Z. Huang, "Two ellipse-based pruning methods for group nearest neighbor queries," in *Proceedings of ACM International Workshop on Geographic Information Systems (GIS)*, 2005, pp. 192–199.

[11] H. L. Razente, M. C. N. Barioni, A. J. M. Traina, C. Faloutsos, and C. T. Jr., "A novel optimization approach to efficiently process aggregate similarity queries in metric access methods," in *Proceedings of ACM Conference on Information and Knowledge Management (CIKM)*, 2008, pp. 193–202.

[12] K. Deng, S. W. Sadiq, X. Zhou, H. Xu, G. P. C. Fung, and Y. Lu, "On group nearest group query processing," *IEEE Trans. Knowl. Data Eng. (TKDE)*, vol. 24, no. 2, pp. 295–308, 2012.

[13] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *Proceedings of ACM International Conference on Management of Data (SIGMOD)*, 2011, pp. 373–384.

[14] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu, "Collective spatial keyword queries: a distance owner-driven approach," in *Proceedings of ACM International Conference on Management of Data (SIGMOD)*, 2013, pp. 689–700.

[15] D. Zhang, C.-Y. Chan, and K.-L. Tan, "Nearest group queries," in *Proceedings of the Conference on Scientific and Statistical Database Management (SSDBM)*, 2013, p. 7.

[16] D.-W. Choi, C.-W. Chung, and Y. Tao, "A scalable algorithm for maximizing range sum in spatial databases," *PVLDB*, vol. 5, no. 11, pp. 1088–1099, 2012.

[17] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry 3rd revised ed.* Springer-Verlag, 2008.

[18] J.-B. H. Urruty and C. Lemaréchal, *Fundamentals of convex analysis*. Springer, 2001.

[19] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear programming: theory and algorithms*. John Wiley & Sons, 2013.

[20] D.-W. Choi and C.-W. Chung, "Nearest neighborhood search in spatial databases," Department of Computer Science, KAIST, Tech. Rep. CS-TR-2014-388, 2014.

[21] E. M. McCreight, "Priority search trees," *SIAM J. Comput.*, vol. 14, no. 2, pp. 257–276, 1985.

[22] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in *Results and New Trends in Computer Science*. Springer-Verlag, 1991, pp. 359–370.

[23] A. Efrat, M. Sharir, and A. Ziv, "Computing the smallest k-enclosing circle and related problems," *Comput. Geom.*, vol. 4, pp. 119–136, 1994.

[24] P. M. Vaidya, "An $O(n \log n)$ algorithm for the all-nearest.neighbors problem," *Discrete & Computational Geometry*, vol. 4, pp. 101–115, 1989.