Continuous Nearest Neighbor Search

Lien-Fa Lin*
Department of Computer Science and Information Engineering National Cheng-Kung University
E-mail: lienfa@cc.kyu.edu.tw

Chao-Chun Chen
Department of Information and Communication Southern Taiwan University
E-mail: chencc@mail.stut.edu.tw

Abstract

The query service for the location of an object is called Location Based Services (LBSs), and Reverse Nearest Neighbor (RNN) queries are one of them. RNN queries have diversified applications, such as decision support system, market decision, query of database document, and biological information. In the environment of wireless network, users often remain in moving conditions, and sending a query command while moving is a natural behavior. Availability of such service therefore becomes very important; we refer to this type of issue as Continuous Reverse Nearest Neighbor (CRNN) queries. Because an inquirer’s location changes according to time, RNN queries will return different results according to different locations. For a CRNN query, executing RNN search for every point of time during a continuous query period will require a tremendously large price to pay. In this study, an efficient algorithm is designed to provide precise results of a CRNN query in just one execution. In addition, a large amount of experiments were conducted to verify the above-mentioned method, of which results of the experiments showed significant enhancement in efficiency.

Keyword: Location Based Services, Location-Dependent Query, Continuous Query, Reverse Nearest Neighbor Query, Continuous Reverse Nearest Neighbor Query

1. Introduction

Owing to the popularity of personal digital devices and advances in wireless communication technologies, location-based services (LBSs) have received a lot of attention from both of the industrial and academic communities [1].

LBSs shall become an indispensable application in mobile network as its required technology has matured and 3G wireless communication infrastructure is expected to be deployed everywhere. The query that answers to LBSs is referred to as Location-Dependent Query (LDQ), of which applications include Range Query, Nearest Neighbor (NN) query, K-Nearest Neighbor (KNN) query, and Reverse Nearest Neighbor (RNN) query.

There are plenty of studies about NN [8,12], KNN [3,9,11], CNN [2,6,7], and CKNN [6,7] queries, and issues pertaining to Reverse Nearest Neighbor (RNN) Query [5,8,10] have been receiving attention in recent years. RNN query means finding a collection of nearest neighbor objects for S, a given collection of objects, with q, a given query object. Practical examples of RNN query are provided in [5]. If a bank is planning to open a new branch, and its clients prefer a branch on a nearest possible location, then such new branch should be established on a location where the distance to the majority of its clients is shorter than that of other banks. Taxi cabs selecting passengers is another good example. If a taxi cab uses wireless devices to find out the location of its customer, then RNN queries will be far more advantageous than NN queries from the aspect of competition. Figure 1 illustrates the nearest neighbor to Taxi A is Customer C, but that does not necessarily mean Taxi A is the most likely to get to Customer C because Taxi B is even closer to Customer C. On the contrary, Taxi A should head for Customer D because Taxi A is the nearest neighbor in relation to Customer D. That is, the

---

* Lien-Fa Lin is also an assistant professor at the Department of Information Communication Kao Yuan University, No.1821, JhongShan Rd, Lujhu Township, Kaohsiung Country 821, Taiwan, R.O.C
RNN for Taxi A is Customer D, and Taxi A may get to Customer D faster than all other taxis. This is an example of CRNN query for that the query object, the taxi, changes location according to time. Mobile users will be mobile in a wireless environment, and that is why the continuous query is an important issue in the wireless environment.

Because an inquirer changes location constantly according to time, changes of location will cause RNN queries to return different results. For a CRNN query, executing RNN search for every point of time during a continuous query period will require a tremendously large price to pay. The larger the number of query objects and the shorter the time segment are, the longer the calculation time will be.

In addition, due to the continuance nature of time, defining the appropriate time segment for RNN search will be a concern; if the interval between RNN searches is too short, then more CRNN queries need to be executed to complete the query, and vice versa. If a RNN search is repeated over a longer period of time to reduce the number of execution, the RNN query result for the whole time segment will lose accuracy due to insufficient frequency of sampling.

In this paper, a more efficient algorithm is designed to replace processing of each and every point of time for RNN search; just one execution of CRNN query is all it takes to properly define the segment for the query time that a user is interested in, and find out the segments that share the same answer and the RNN for each of the intervals.

Other than that, an index is also used to filter out unnecessary objects to reduce search space and improve CRNN search efficiency. The experiment results suggest that using index provides efficiency 20 times better than not using index when the number of objects is 1000. Our contributions are three-fold.
(1) This paper pioneers into continuous query processing methods opposite to static query regarding RNN issues.
(2) A CRNN search algorithm is proposed; just one execution will return all CRNN results.
(3) The proposed method allows the index which was only applicable to finding RNN for a single query point to support CRNN query to improve CRNN search efficiency.

The rest of paper is organized as follows. Section 2 is an overview of related work. Concerned issues are defined and assumptions are described in Section 3. The proposed CRNN search algorithm is introduced in Section 4. The experiment environment and evaluation parameters for experimental efficacy are described in Section 5. Finally, we summarize the paper and describe our future work in Section 6.

2. Related work

One such query is the reverse nearest neighbor (RNN) query that returns the objects that have a query object as their closest object. Related works of study about RNN search are introduced and summarized in this section:

2-1 Index Methods for RNN query

RNN search concerns about finding the objects that have a query object q as their closest object, and it is necessary to find out the distance between query q and each object, or the distance from the coordinate of query q to the coordinate of an object. For a given q, not every object is its RNN, and these objects which can not be RNN may be practically left out of consideration to reduce the number of objects to be taken into consideration and accelerate processing speed for RNN search. Many studies were dedicated to the designing of an effective indexing structure for coordinates of an object. The most famous ones are R-Tree proposed by [4] and Rdnn-Tree proposed by [5]. R-Tree is an index structure developed in early years for spatial database and was used by [5] to accelerate RNN search processing.

Rdnn-tree (R-tree containing Distance of Nearest Neighbors) [8] improves the method of [5]. The author proposes a single index structure (Rdnn-tree) to provide solutions for NN queries and RNN queries at the same time. Rdnn-tree differs from standard R-tree
structure by storing extra information about nearest neighbor of the points in each node. Information of \((ptid, dnn)\) is stored on the leaf node of Rdnn-tree, as shown in Figure 2. \(ptid\) means an object of which the data concentrate on the dimension, denoted as \(d\), and \(dnn\) means the distance from such object to its NN. Information of \((ptr, Rect, MaxDnn)\) is stored on a non-leaf node, where \(ptr\) points to the address of a child node, \(Rect\) contains the MBR of all child nodes subordinate to this node, and \(MaxDnn\) means the maximum value of \(dnn\) of all objects in the child trees subordinate to this node. The maximum distance from any object contained in these child trees to its NN will not exceed \(MaxDnn\).

![Figure 2. Data structure of Rdnn-tree](image)

2-2 Categories of RNN queries

Depending on the static or moving status of query \(q\) and the query objects, related studies can be summarized into 4 categories.

1. If query \(q\) and the query objects are both static, then this category is called static query vs. static objects.
2. If query \(q\) is moving and the query objects are static, then this category is called moving query vs. static objects.
3. If query \(q\) is static and the query objects are moving, then this category is called static query vs. moving objects.
4. If both query \(q\) and the query objects are moving, then this category is called moving query vs. moving objects.

2-2-1 Static query vs. static objects

The scenario that both query \(q\) and query objects are static is first discussed because the query and query objects are immobile and are therefore easier for processing than other scenarios. The method proposed in [5] is now introduced. For static database, the author adopts a special R-tree, called RNN-tree, for answering RNN queries. For static database that requires being frequently updated, the author proposes a combined use of NN-tree and RNN-tree. NN of every object is stored in the RNN-tree, and what are stored in the NN-tree are the objects themselves and their respective collections of NN. The author uses every object as the center of a circle, of which the radius is the distance from the object to its NN, to make a circle, and then examines every circle that contains query \(q\) to find out the answers of RNN queries. Such method, however, is very inefficient for dynamic database because the structures of NN-tree and RNN-tree must be changed whenever the database is updated. In [8], the method proposed by [5] is therefore improved. The author proposes a single index structure, Rdnn-tree, for answering NN queries and RNN queries at the same time. It differs from normal R-tree; it separately stores the information of NN of every object (i.e. Distance of Nearest Neighbor), and NN of every object must be calculated in advance.

2-2-2 Static query vs. moving objects

Studies mentioned above primarily assume a monochromatic situation that all objects, including query \(q\) and query objects, are of the same type. In [18], the researcher addresses this type of issues in a bichromatic situation that objects are divided into two different types; one is inquirer, and the other is query object. NN and range query techniques are used in this Paper to handle RNN issues.

2-2-3 Moving query

This subsection discusses the situation when an inquirer is no longer static but changes his or her location according to time, and the query object can be either static or moving. That is, two categories of query: moving query with static objects and moving query with moving objects, are involved. Because the inquirer is moving, these two categories of query will return different results for the identical RNN search at different points of time. This type of issue is obviously more complicate than the issues previously discussed. As far as the knowledge available to the researcher is concerned, no related study has ever discussed about the issues of these two categories. In this Study, solutions for a moving query with static objects are pursued.

3. Problem Formulation
CRNN query concerns about a period of continuing time where adjacent points in such period of time may have the identical RNN. That is, a period of time may have the same RNN unless the query q has moved beyond this period of time. Please refer to Figure 3. When a user executes CRNN query q, the time segment of the continuous query is [Ps,Pe], and the query objects are \{a,b,c,d\}. If time point P1 can be identified, and any given point of time in the time segment from Ps to P1, or [Ps,P1], has the same RNN result, then one-time execution of RNN search is all it needs for the time segment of [Ps,P1]. If points of time, P2, P3, and P4, are also identified, and any given point of time in the time segment of [P1,P2] has the identical RNN result, while any given point of time in the time segment of [P3,P4] has the identical result, then the entire CRNN query needs only one-time execution of RNN search at each time segment.

For processing CRNN query, it is not necessary to execute RNN search for every point of time and return the RNN of every point of time back to query q; instead, i segments of time, such as segment1, segment2,...,segmenti, that have the same result, are first identified among the entire CRNN query time period. RNN result of each segment of time, such as RNN1, RNN2,...,is calculated separately, and the result is returned in the format of (q, [segmenti])={RNNi result} back to the inquirer.

Under the assumptions:
The moving direction of query q is fixed. All query objects are static.

As described above, two adjacent points of time may share the same RNN, or a segment of time has only one RNN unless query q moves to another segment of time that has a different RNN. The CRNN search algorithm proposed in this Study uses exactly this concept. First, the points of time that produce different RNN results within a query time are identified. These points of time divide the query time into several segments that have different RNN results, and then the RNN results are identified for each of the segments. The detailed algorithm of CRNN Search is explained in the next section.

4. CRNN Search Algorithm

The detailed procedure of CRNN Search algorithm is introduced in this section. CRNN Search algorithm is divided into two steps.

Step 1: Finding segment points of CRNNq.
Points of time that produce different RNN results are identified. Based on these points of time, CRNN query is divided into several time segments that require execution of RNN search. The RNN result for any given point of time within one segment will remain constant, and different segments have different RNN results.

Step 2: Calculating RNN result of each segment.
Separately calculate the RNN results for each of the segments that have been divided in the previous step.

4-1 Finding segment points of CRNN

What CRNN query pursues is a period of continuous time; the moving distance of query objects is very short among some adjacent points of time for the query, thus possibly resulting in the same RNN result. That is, the entire period of continuous query is divided into several segments, and the RNN results in each segment are the same. If these points of time share the same RNN result, then it is not necessary to execute RNN search for each of the points of time; one-time calculation is enough. Therefore, CRNN query does not require executing RNN search for all points of time. Instead, points of time that share the same RNN result are grouped into time segments, and
one-time RNN search is executed for each of the segments. RNN of query \( q \) is a collection of the objects of which the NN is query \( q \). If the distance, or \( N \), is realized in advance, then these objects are the RNN for query \( q \) when the distances from query \( q \) to the objects are shorter than the distances from the objects to their respective NN.

As illustrated in Figure 4, if the NN of object \( a \) is \( b \), and a circle is made using \( ab \) as the radius with \( a \) as the center point, then the distance from query \( q \) to \( a \) must be shorter than the distance from \( a \) to its NN, or object \( b \), as long as query \( q \) falls within this circle. Therefore, during the period of time when query \( q \) remains within this circle, RNNs of object \( a \) must include \( a \), unless query \( q \) moves out of this circle. Because the moving direction of query \( q \) is assumed to be fixed, CRNN query will form a query line (\( q \)-line) from its beginning to its end. The point to which this CRNN query begins to leave this circle is the intersection \( S \) of this circle and the query line formed by CRNN query. Before intersection \( S \), the result of RNN query must include object \( a \); beyond intersection \( S \), the result of RNN query will not include object \( a \); the RNN results will be different. This intersection is referred to as a segment point.

This explains why the intersection of the circle with NN as its radius and the query line is the point of time where RNN query produces different results. Making a circle by using an object itself as the center and the distance to its NN as the radius will enable all of the intersections of the circle and the query line of CRNN query to cut CRNN query into several time segments that have different results of RNN query.

4-2 Calculating RNN result of each segment

In the previous section, intersections of \( q \)-lines and the circles with the distances between the objects and their respective NNs as the radiuses are defined. With these intersections, CRNN query is cut into several time segments. The next step is to find RNNs for each of the time segments. Because the distances from query objects to their respective NNs are used as the radiuses to make circles which are coded by the objects’ numbers, if a segment falls within a certain circle, then the resulting RNN of this time segment for the CRNN query is the object collection represented by such circle. This is illustrated in Figure 6. First, intersections of \( q \)-lines that represent the CRNN query and the circles of the objects are sorted by time; every two intersection points define a time segment, and there are five segments, \([P_s,P_1],[P_1,P_2],[P_2,P_3],[P_3,P_4]\), and \([P_4,P_e]\). Segment \([P_s,P_1]\) is contained only by circle \( a \), therefore: RNN(\( q,[P_s,P_1]\)) = \{a\}. Next, examine segment \([P_s,P_1]\); this segment is contained by circle \( a \) and circle \( b \). Therefore: RNN(\( q,[P_s,P_1]\)) = \{a,b\}. If this process is repeated, then the obtained results will be RNN(\( q,[P_2,P_3]\)) = \{a,b,c\}, RNN(\( q,[P_3,P_4]\)) = \{b,c\}, and RNN(\( q,[P_4,P_e]\)) = \{c\}.
4-3 CRNN algorithm with index

Not every object will be an answer in the processing of CRNN query. To improve RNN query efficiency, it is preferred that the objects that can not be answers are filtered out in advance to greatly reduce search space for CRNN query, size of data that requires CRNN query, and consequently, computation cost. The process that further improves CRNN query efficiency dramatically is referred to as pruning process. The pruning process, an index structure for Rdnn-tree is designed to effectively execute the pruning process. The pruning process is described below. For every internal node of Rdnn-tree, the distance from query q to its node will be computed for every separation, and the distance is denoted as D(q, Rect). If D(q, Rect) is larger than MaxDnn of the node, then all the objects beneath it will not be considered because the distance from query q to Rect Node will be equal to or larger than the distances from query q to all the objects underneath Rect node. When the distance from query q to Rect node is longer than MaxDnn, it is impossible that query q is closer to its NN than any other object underneath Rect node, and no object underneath can be the RNN result for query q. On the contrary, if D(q, Rect) equals the MaxDnn of such node, then the distances from some objects underneath Rect node to their respective NNs are shorter than the distance from query q to Rect. That is, some objects are the RNN results for query q. The examination continues along the branch all the way to the lead node. All entries underneath such leaf node are recorded as the candidate objects for RNN query result. The collection of these candidate objects is referred to as RNNCanSet, which means the possible results for RNN query must exist within this collection, and the objects outside of RNNCanSet can not possibly be RNN query results. All that are needed to be considered when finding segment point of CRNNq of CRNN search algorithm are the objects inside RNNCanSet. This will greatly reduce the quantity of objects needed to be handled and enhance CRNN search algorithm efficiency.

Figure 2 explains the pruning process. It begins with root node R. Because D(q, R) ≤ MaxDnn of R, child nodes of B1 and B2 must be examined. Because the MaxDnn of MBR B1 ≤ D(q, B1), all child nodes underneath B1 can be pruned. Next, D(q, B2) ≤ MaxDnn of B2, so child nodes b3 and b4 of B2 must be examined. D(q, b3) is equal to or smaller than the MaxDnn of b3, which is also a leaf node; therefore, h and i are placed inside RNNCanSet. Next, b4 is examined. MaxDnn of b4 is equal to or smaller than D(q, b4); therefore, b4 can be pruned. The entire pruning process then ends.

However, the CRNN query to be processed is not a RNN query of a single query point; therefore, the pruning process in [8] can not be directly used. To ensure that no possible RNN result is deleted, the criteria of pruning is changed from the condition that D(q, Rect), the distance from query point to Rect, must be longer than MaxDnn to the condition that MinD(qline, Rect) > MaxDnn, where MinD(qline, Rect) represents the minimum distance from qline to Rect node. The reason why the shortest distance is selected is that if the minimum distance from the entire qline to Rect node is larger than MaxDnn, then the distance from any given point of time on the qline to Rect node must be longer than MaxDnn. Therefore, all the objects underneath Rect node can not be RNN for qline, and pruning is out of consideration. Details of the pruning algorithm are exhibited in Algorithm 1:

```
Algorithm 1: Pruning Algorithm.

INPUT:
- RdnnTree: B, Rect, MaxDnn, qline
- root, Node B
- Function MinD: qline, Rect)

OUTPUT:
- RNN Candidate Set: RNNCanSet

1: initial N = root;
2: if N is a leaf Node then
3: Add each object of Node N into RNNCanSet;
4: end if
5: if N is an internal node then
6: for each branch B of Node N do
7: if MinD(qline, B) ≤ B.MaxDnn then
8: Call Prune(B, qline);
9: end if
10: end for
11: end if
12: Return RNNCanSet;
```

5. Conclusions and Future Works

An efficient CRNN search algorithm is proposed in this Study. Such algorithm requires only one execution to find out RNN results from all continuous RNN searches. The diversified experiments in this Study also prove the efficiency of the proposed method. As wireless communication and mobile device technology become mature, more and more users access information from wireless information systems through mobile devices. To process requests from more and more mobile users, data dissemination through broadcast is an effective solution for scalability. The future goal of this Study is extending the issues of CRNN search to the wireless broadcasting environment.

6. Acknowledgement

This work was supported by National Science Council of Taiwan (R.O.C) under Grants NSC96-2221-E-218-050-MY2, NSC96-2515-S-218-002 and NSC96-2515-S-218-003. The authors would like to thank the anonymous reviewers for their helpful comments, which improved presentation of this paper.

7. References