A Safe Exit Approach for Continuous Monitoring of Reverse K-Nearest Neighbors in Road Networks

Muhammad Attique, Yared Hailu, Sololia GudetaAyele, Hyung-Ju Cho and Tae-Sun Chung
Department of Computer Engineering, Ajou University, South Korea

Abstract: *Reverse K-Nearest Neighbor (RKNN) queries in road networks have been studied extensively in recent years. However, at present, there is still a lack of algorithms for moving queries in a road network. In this paper, we study how to efficiently process moving queries. Existing algorithms do not efficiently handle query movement. For instance, whenever a query changes its location, the result of the query has to be recomputed. To avoid this recomputation, we introduce a new technique that can efficiently compute the safe exit points for continuous RKNNs. Within these safe exit points, the query result remains unchanged and a request for recomputation of the query does not have to be made to the server. This significantly reduces server processing costs and the communication costs between the server and moving clients. The results of extensive experiments conducted using real road network data indicate that our proposed algorithm significantly reduces communication and computation costs.*

Keywords: *Continuous monitoring, reverse nearest neighbor query, safe exit algorithm, road network.*

Received April 29, 2013; accepted July 11, 2013; published online December 3, 2014

1. Introduction

In recent years, Reverse Nearest Neighbor (RNN) query processing has garnered a considerable amount of attention based on applications such as location based services, decision support and resource allocation. Much research has been conducted and several efficient algorithms provided in both Euclidean and spatial networks. However, there is still a lack of research on the processing of moving queries in road networks. In addition, while a plethora of work have been devoted to moving query processing [1, 4, 8, 9, 10, 14, 21, 23, 25, 26] they all focus on the Euclidean space, not on road networks.

Given a query point q, a Reverse K-Nearest Neighbor (RKNN) query retrieves all the data points that have q as one of their K-Nearest Neighbors (KNN) (K closest points). Consider an example of location-based games, as shown in Figure 1, in which the goal of each player is to shoot the player nearest to him. Each player needs to continuously monitor his own RNN to avoid being shot by other players. The closest fighter to q is p_1. However, p_1 is not the RNN of q because the closest point to p_1 is p_2, not q. The RNN of q is p_3 because q is the nearest neighbor to this fighter. Thus, p_3 is the fighter that fighter q should monitor.

Figure 1. Example of RKNN query.

RNN queries are generally categorized into two types: Monochromatic RKNN (MRKNN) queries and Bichromatic RKNN (BRKNN) queries. The above example belongs to the monochromatic category as both query objects and data objects share the same type of objects, i.e., players.

Other efficient implementations of RNN query that are of particular interest in information systems are those that support user's queries, such as online search engines, multimedia search engines and GIS. For instance, a business owner when deciding to open a new restaurant may ask an RNN query such as "Where is the best location for the Italian food restaurant?", this question can be rephrased as "how many users consider this possible location to be the nearest Italian food restaurant?". Each candidate location for the Italian food restaurant should start an RNN query and the results then compared in order to choose the best location. The above example deals with bichromatic queries (which are defined as queries in which the query objects and the data objects belong to two different types of objects) because the data object and query object are different types of objects. Another interesting example of BRKNN queries is emergency services such as enhanced 911 services. Whenever a 911 services center receives any emergency call, the desire is to send this job to the team that is closest to the emergency call location in order to get a good response time.

The existing technique results in expensive communication and computation costs if a query changes its location. Therefore, the main challenge for continuous monitoring algorithms is the maintenance of the freshness of the query answer when the query point moves freely and arbitrarily. A simple approach is to have the client q periodically send requests to re-evaluate the query results. However, this approach still

does not guarantee that results are fresh because the query answer may still become stale in between each call to the server. In addition, this approach is very expensive in terms of computation and communication costs.

Safe region-based algorithms have been introduced as a means of overcoming the excessive computation and communication costs associated with periodic monitoring [3, 6, 29, 30]. The safe region of a query is the region where a query answer remains unchanged, provided that the query point is within the safe region. The safe region technique avoids the back and forth communication between client and server, although it also allows a client to get fresh query results without excessive overhead on the server side. However, to provide the safe region (which may consist of complex road segments) more network bandwidth is consumed compared to simply providing a set of safe exit points [3, 10] representing the boundary of the safe region.

In this paper, to overcome the problem outlined above, we propose a new safe exit technique that efficiently computes the safe exit points for moving RNN queries in road networks. At each safe exit point, the safe region of a query and its non-safe region meet so that a set of safe exit points represents the border of the safe region. In road networks, the safe exit approach is more efficient than the safe region approach because the communication cost between query (client) and server is comparatively low. Until a client q reaches a safe exit point, s/he is guaranteed to remain in the safe region and thus the query answer is valid. The query results and safe exit points are recalculated only when q travels beyond the safe exit points.

The following assumptions are made in our proposed technique:

- We consider only BRKNN queries.
- Query objects move and data objects are static.

In this paper, we present techniques that can compute safe exits efficiently. The contributions made are as follows:

- We present a framework for continuous monitoring of RKNN queries in road networks in our safe exit technique proposal.
- We present novel pruning rules that optimize the computation of safe exit points by minimizing the size of the unpruned network and the number of objects.
- A thorough experimental study confirms that our approach outperforms a traditional approach that does not use the safe exit approach, in terms of both communication and computation costs.

The remainder of this paper is structured as follows: Section 2 reviews existing work on continuous monitoring of RKNN queries in Euclidean space and road networks. Section 3 presents terminology definitions and describes the problem. Section 4 elaborates on our proposed Safe Exit Algorithm (SEA) for computing the safe exit points of moving nearest neighbor queries in road networks. Section 5 presents a performance analysis conducted of the proposed technique. Section 6 concludes this paper.

2. Related Work

An RNN query for moving objects searches for those objects that take object q as their nearest neighbor. In recent years, reverse neighbor query processing has received significant attention by the spatial database systems research community. Many algorithms have been proposed for the monitoring of RNNs, especially in Euclidean space. However, there is still a lack of efficient algorithms for road networks. Our related work is divided into two sections: Section 2.1 covers continuous RNN query processing in Euclidian space while section 2.2 reviews continuous RNN query processing in road networks.

2.1. Algorithms for Continuous RKNN Query Processing in Euclidian Space

Korn and Muthukrishnan [14] were the first to introduce the concept of RNN. They used the pre-computing technique to search for RNNs. The main drawback of this approach is that they were limited to supporting RKNN queries for a fixed number of K and they were also inefficient in processing object movements. This preprocessing issue was first addressed by Song and Roussopoulos [20] who proposed the 60 degree pruning method which partitions the entire space centered at a query q into equal regions. It can be verified that the possible RNN of q can only be the nearest point to q found in each region. This also depicts that in 2D space, there are at most six possible RNNs of q, which is the main advantage of this algorithm. Another efficient algorithm is TPL pruning, introduced by Tao *et al.* [23], which uses the properties of half space to locate candidates. The algorithms proposed in [21, 23] are categorized as snapshot RNN algorithms.

A number of algorithms have also been proposed for efficient monitoring of nearest neighbor queries, continuous range queries and RNNs [6, 7, 11, 13, 14, 16, 18, 20, 21, 22, 24, 27]. The existing continuous query processing algorithms place emphasis on defining the monitoring region of a query and updating the query result based on moving object's location updates. Benetis *et al*. [1] were the first to study continuous RNN monitoring, but their proposed scheme assumes that the velocity of objects are known. Xia and Zhang [26] introduced an incremental, scalable approach to the monitoring of continuous RNNs. Their method is based on the 60 degree pruning technique. In their approach, the monitoring region of a continuous RNN query is defined as six pie regions (determined by the query point and the six candidates)

and six arc-regions (determined by the six candidates and their nearest neighbor). Theirs is an efficient algorithm because it identifies and processes the updates that fall into the monitoring region, unlike other conventional methods. However, this scheme has two major limitations; firstly, it only processes monochromatic RNN queries and, secondly, it assumes that at every time interval there are six RNNs, which is the worst-case scenario. Kang *et al.* [12] proposed a novel algorithm for monitoring continuous RNNs called IGERN. It is based on the TPL-pruning method and caters to both monochromatic and BRKNN queries. It is more efficient than the 60 degree pruning based solutions because it monitors fewer candidates as opposed to the entire space. The trade off of this scheme is that it cannot be easily extended to handle continuous RKNN queries where K>1. Therefore, the monitoring region defined in it only applies to continuous RNNs where K=1.

Wu *et al.* [25] proposed a technique to monitor RKNNs that involves continuous filtering and continuous refining. They determined that the refining step is critical especially when K>1, as at that point the refining cost becomes the system overhead. They proposed a new refining framework called CRange-k, which verifies candidate objects by issuing KNN queries in each region rather than single nearest neighbor queries. The users that are closer than the K^{th} nearest neighbor in each region are the candidate objects and they are verified if q is one of their K closest facilities. To monitor the results, for each candidate object, they continuously monitor the circle around it that contains K nearest facilities. Cheema *et al.* [3, 4, 5] proposed several schemes for the monitoring of continuous RNNs. In [4], they used the concept of influence zone and focused on continuous BRKNN queries in which the query object is static and the data objects are moving. In [6], they proposed a new framework based on safe regions for both Euclidean and road networks in which query and data objects are both moving. This scheme significantly improves the computation cost as it assigns each object and query a safe region such that expensive recomputation is not required as long as the query and objects remain in their respective safe regions.

2.2. Algorithms for Continuous RKNN Query Processing in Road Networks

The processing of RNN queries in road networks is one of the recent emerging areas of research. Yiu *et al.* [28] first addressed the issue of RNN in road networks (they represented road networks as graphs) and proposed an algorithm for both monochromatic and BRKNN queries. Safar *et al.* [19] studied snapshot RNN queries in spatial networks. They presented a framework based on Network Voronoi Diagrams (NVDs) to efficiently process RNN queries in road networks. However, their scheme is not suitable for continuous RNN queries due to the fact that an NVD changes whenever a dataset changes its location, resulting in high computation costs.

Sun *et al.* [22] presented a method for continuous monitoring of BRKNN queries. They associated a multiway tree with each query to define the monitoring region, and only the updates in the monitoring region affect the results. However, this method is limited to bichromatic queries and also does not cater for K>1. Moreover, their proposed scheme assumes that the query objects are static.

Li *et al.* [15] proposed a novel algorithm for continuous monitoring of RKNNs based on a Dual Layer Multiway (DLM) tree in which they introduced several lemmas to reduce the monitoring region and filter the candidate objects. Their continuous monitoring of RKNN method comprises two phases: The initial result generating phase and the incremental maintenance phase. Cheema *et al.* [6] proposed a safe region approach for the monitoring of continuous MRKNN and BRKNN queries in Euclidean and road networks, as mentioned in section 2.1 and devised pruning rules that reduce the monitoring region.

In the traditional techniques, every object reports its location to the server at every timestamp regardless of whether query results will be affected or not. Consequently, such a computation model increases the communication cost as it requires transmission of a large number of location updates. In our framework, each moving object reports its location update only when it leaves the safe exits. This significantly saves on communication costs. Existing approaches [6, 15] have also addressed the abovementioned issue, but their approach is safe region-based. In contrast, our approach is safe exit-based and, as discussed earlier, the safe exit approach is more efficient than the safe region approach because the communication cost between query (client) and server is comparatively low.

3. Terminology and Problem Descriptions

A road network G is a weighted undirected graph consisting of nodes and edges. An edge between two nodes is denoted by $e(n_1, n_2)$. $W(e)$ is a function that returns the weight of an edge e and represents the length of its corresponding road segment.

Segment $s(p_1, p_2)$ is the part of an edge between two points, p_1 and p_2, on the edge. An edge consists of one or more segments. An edge is also considered to be a segment in which the nodes are the end points of the edge.

Figure 2 shows an example of an undirected road network with five nodes, a to e. Several edges and segments are shown with their respective weights. For example, the edge $e(a, e)$ consists of segments $s(a, o_1)$, $s(o_1, q)$ and $s(q, e)$ having weights 3, 7 and 4, respectively. There are eight objects in this example (o_1 to o_7 and q). Query q and the data objects are shown as star and rectangles, respectively.

To simplify the presentation, Table 1 summarizes the notations used in this paper.

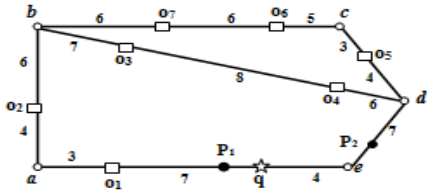

Figure 2. Example of a road network.

Table 1. Notations used in this paper.

Notation	Definition
G = (N, E, W)	Graph Model of a Road Network
dist(a, b)	Length of the Shortest Path between Objects a and b
n_i	A Node in the Road Network
e_i= (n_j, n_k)	An Edge in the Edge set E
W(e_i)= d(n_j, n_k)	Weight of the Edge e_i=(n_j, n_k)
q	A Query Point in the Road Network
k	The Number of Requested RNNs
O	The Set of Objects O={o_1, o_2, …, o_n}
p_{se}	A Safe Exit Point at which the Safe Region of q and its Non-Safe Region Meet
O^+	Set of Answer Objects O^+= {o_1^+,o_2^+, …, o_n^+}
O^-	Set of Non-Answer Object O^-= {o_1^-,o_2^-, …, o_n^-}
IO^+	Influence Region of Answer Objects
IO^-	Influence Region of non-Answer Objects

3.1. Problem Description

In this paper, we primarily address the problem of continuous monitoring of RKNN queries on moving queries in road networks. To provide a clear explanation, we use the example road network shown in Figure 2, in which there are seven objects, o_1 to o_7 and a query q in a road network. Let us assume that a moving query requests three RNNs at point P_1. In order to get the three RNNs, we expand the road network from point q until we find them. As mentioned earlier, query q is moving in a road network and it moves from P_1 to P_2. As for getting the three nearest neighbors at point P_2, the simple approach is to repeat the procedure executed at P_1. However, this recomputation whenever query q changes its location significantly degrades the performance of the algorithm. To address this issue, we introduce the safe exit approach.

Figure 3 shows the safe region of q and its safe exit points for the example in Figure 2. As shown in Figure 3, before q reaches either f or g, the RNNs of q are {o_1, o_4}. When q passes through either f or g, the query is re-evaluated based on the updated location of q in order to refresh the query answer and the safe exit points.

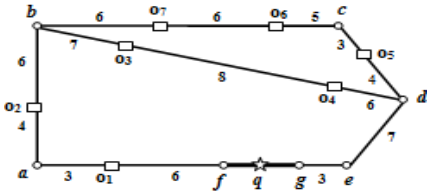

Figure 3. Safe exit points for q.

4. SEA for Moving RKNN Queries

In this section, we present a new SEA for moving RKNN queries in a road network. Algorithm 1 depicts the skeleton of our proposed SEA for computing safe regions. It consists of three phases: Finding of Useful Objects (UOs) that could contribute to the safe region, computation of the influence region of the UOs and computation of safe exit points.

Algorithm 1: Computation of safe regions (o, q).

Input o: *Data objects*, *q*: *Query object*
Output: *Safe Region* (*SR*)

```
/*Phase 1: Retrieve UOs */
  Point set p': road network (o, q)
  Point set A: {o+ ∈p'|dist (o, q)<dist(o, ok+1)} /* find answer
  objects */
  Point set NA: {o- ∈p'|dist (o, q)>dist (o, ok)}/* find non-
  answer objects */
/*Phase 2: Computation of Influence Region */
  while A is non-empty do
        Point p ← pick object (A);
        Calculate influence region;  /*details in Algorithm 2*/
        Store in influence region set IR+;
  end
  while NA is non-empty do
        Point p ← pick object (NA);
        if object= UO then
          Calculate influence region;/* details in Algorithm 2
          */
          Store in influence region set IR-
        else
          prune the object
  end
/*Phase 3: Computation of Safe Region */
  Computesafe region from sets IR+ and IR-
  /*details in phase 3 section */
  Storesafe region SR;
  return SR;
```

4.1. Phase 1: Finding UOs

This phase aims at finding potential points that could contribute to the computation of safe regions. We first define some pruning rules to retrieve a small number of data objects to avoid the computation cost. We then divide the UOs into two types; namely, answer objects (denoted by O^+) and non-answer objects (denoted by O^-).

Before we present the pruning rules, let us first define answer objects. An object o is called an answer object if $dist(o, q) < dist(o, o')$ where o' is any other object in the road network. Similarly, we can generalize it for RkNN: an object o is called an answer object if $dist(o, q) < dist(o, o_{k+1})$ where o_{k+1} is the $(k+1)^{th}$ object. In Figure 3, object o_1 is the RNN of q because the distance from o_1 to q is less than that from o_1 to any other object in the road network. Likewise, if we want to find the 3RNNs, objects o_1 and o_4 both are the 3RNNs of q because the distance from o_4 to q is less than that of the fourth nearest object.

Let us now look at the pruning rules that identify the UOs:

- **Pruning Rule 1**: All answer objects are UOs.
  Explanation: We can generalize the above definition of answer objects to state that answer objects are RNNs. Therefore, all RNNs should be considered UOs.
- **Pruning Rule 2**: An object O cannot be a UO if its kNN does not contain any O^+ (answer object).
  Explanation: From the definition of safe exit points (which we explain in detail in phase 3), the safe region includes the intersection of all answer objects and excludes the union of all non-answer objects. In other words, if the influence region of a non-answer object does not contain any portion of the answer objects, then it cannot be a UO. In phase 2, we will explain how to calculate the influence region of answer and non-answer objects.

  Before we present the next pruning rule, let us define the term blocking object. An object o is called a blocking object if the shortest path between q and o only contains answer objects. In other words, a blocking object is considered the first non-answer object in any particular path towards q. In Figure 3, objects o_2 and o_5 are blocking objects. Object o_3 is not a blocking object because the shortest path from o_3 to q is $\{o_3, b, o_2, a, o_1, q\}$, which contains another blocking object, o_2.
- **Pruning Rule 3**: An object o cannot be a UO if the shortest path between q and o contains any blocking object.

  On applying these pruning rules to the road network example, segments $s(o_2, a)$, $s(a, o_1)$, $s(o_1, q)$, $s(q, e)$, $s(e, d)$, $s(d, o_4)$ and $s(d, o_5)$ are used to construct an unpruned network and the rest of the network is considered to be a pruned network.

4.2. Phase 2: Computation of Influence Region for UOs

After we retrieve the set of UOs, the next step is to compute the influence region of every UO.

- Influence Region of Answer Objects: Is defined as follows:

$$I(o^+) = \{p \mid dist(o^+, p) \leq dist(o^+, o_{k+1})\} \quad (1)$$

Where o_{k+1} denotes the $(k+1)^{th}$ nearest object to o.

The influence region of an answer object can be computed from the distance of the answer object to the $(k+1)^{th}$ answer object. Consider object o_1 in Figure 2: 3NNs of $o_1=(q, o_2, o_7)$ with weights (7, 7, 19).

Going by the definition of the influence region of answer objects, we need to compute the distance to the $(k+1)^{th}$ object (in this example, it refers to 4NN), which is object o_3 and $dist(o_1, o_3)=20$. This means that an area of up to 20 units, starting from o_1 towards q is the influence region of o_1. The influence region comprises segments (o_1, q), (q, e), (e, d), (d, m) and (o_1, q), (q, e), (e, d), (d, n) with weights (7, 4, 7, 2), respectively. The sum of these weights is 20, which is $dist(o_1, o_3)$. Similarly, the influence region of $o_4$can be computed as follows: 3NNs of $o_4=(o_3, o_5, q)$ with weights (8, 10, 17). Object o_6 is the 4NN of q and $dist(o_4, o_6)=18$. Therefore, the influence region of o_4 is 18 units from o_4 in the direction of q.

The bold lines in Figures 4 and 5 show the influence region of o_1 and o_4, respectively.

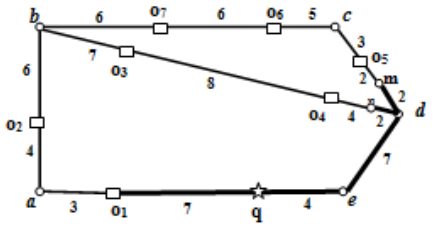

Figure 4. Influence region of o_1.

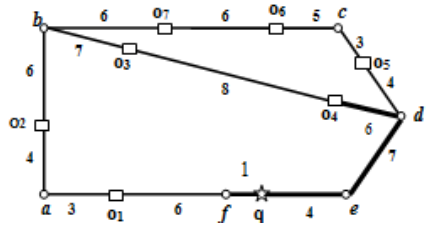

Figure 5. Influence region of o_4.

- Influence Region of Non-Answer Objects: Is defined as follows:

$$I(o^-) = \{p \mid dist(p,q) < dist(p, o_k)\} \quad (2)$$

The influence region of a non-answer object can be computed from the distance of an answer object to the k^{th} object. Consider object o_2 in Figure 2: 3NNs of $o_2=(o_1, o_7, o_3)$ with weights (7, 12, 13).

The influence region of object o_2 is 13 units from o_2 towards q since the distance from o_2 to 3NN with o_3 is 13. Similarly, if we consider o_5, its third nearest neighbor is o_7 and $dist(o_5, o_7)=14$. Thus, the influence region of object o_5 is 14 units from o_5 towards q.

The bold lines in Figures 6 and 7 show the influence region of o_2 and o_5, respectively. The nearest neighbor s for answer objects change when the query object moves outside of the influence region, whereas in the case of non-answer objects, the result changes when the query object moves to the inside of the influence region. In other words, the nearest neighbors of an answer object remain the same until the query object lies inside the influence region, but for non-answer objects the nearest neighbors remain the same until the query object lies outside of the influence region.

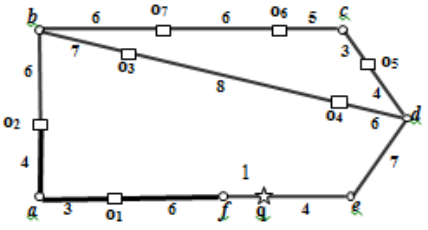

Figure 6. Influence region of o_2.

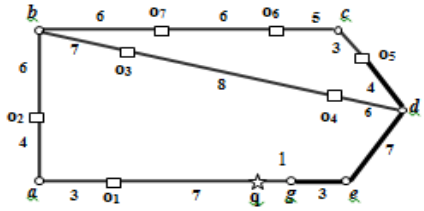

Figure 7. Influence region of o_5.

Algorithm 2: Computation of influence region (A, NA).

Input A: *Answer objects set, NA: non-answer objects set*
Output: IR^+ *Influence Region of answer objects,* IR^- *Influence Region of non-answer objects*

*/*Influence Region of answer objects */*

while A is non-empty do
Point x ← pick object (A);
Compute distance from chosen object to o_{k+1}*;*
Expand the region from the chosen object to the o_{k+1} *unit distance towards q;*
Store in the influence region set IR^+*;*
end
while NA is non-emptydo
Point y ← pick object (NA);
Compute distance from the chosen object to o_k*;*
Expand the region from the chosen object to the o_k *unit distance towards q;*
Store in the influence region set IR^-
end
return IR^+ *and* IR^-

4.3. Phase 3: Computation of Safe Exit Points

The safe region $S(q, r)$ of a query "q" is defined as follows:

$$S(q,r) = \{IIO^+ - UIO^-\} \quad (3)$$

The safe region of query q is based on the influence region of answer objects and non-answer objects. Figure 8 shows how the safe region can be computed based on data points. In the figure, the dataset contains four points, p_1, p_2, p_3 and p_4. The answer objects of query q are $\{p_2, p_4\}$. The intersection of the influence regions of all the answer objects is considered to be the safe region. Secondly, in the example, data points $\{p_1, p_3\}$ are non-answer objects. The safe region of query q should exclude the entire influence region of non-answer objects.

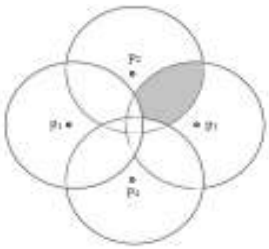

Figure 8. Conceptual example.

In the previous section, we computed the influence region of all answer and non-answer objects. Therefore, by applying the safe region definition, the area between points f and g can be considered a safe region. Points f and g are the safe exit points and form the boundary where the safe region of a query and its non-safe region meet. Figure 3 shows the safe exit points for the road network example.

5. Performance Evaluation

In this section, we evaluate the performance of our proposed Continuous Monitoring of RNN (CMRNN) algorithm by means of simulation experiments and a comparison with a conventional naïve method that recomputes the results at each timestamp. Section 5.1 presents our experimental settings while section 5.2 discusses our experimental results.

5.1. Experimental Settings

In the experiments, we used real-world road network data obtained from [17], which comprises roads in San Francisco consisting of approximately 175, 812 nodes and 223,000 edges. Table 2 lists the default parameters used in our experiments. In each experiment, we evaluated the performance by varying a single parameter with all the other parameters set to default values marked in bold. Each query object randomly selected a node and started moving towards it. Whenever a moving client reached a node, one of its adjacent nodes was selected randomly as the destination and it continued traveling towards it. Since, the query objects move, we simulated their movement using a network-based moving objects generator [2].

Table 2. Experimental parameter settings.

Parameter	Range
Number of Objects	1, 5, **50**, 70, 100 (×1000)
Number of Queries (N_{qry})	1, 3, **5**, 7, 10 (×1000)
Number of Requested RNNs (k)	8, 16, **32**, 64, 128
Query Speed (V_{qry})	20, 40, **60**, 80, 100 (km/h)

We evaluated and compared the performance of our algorithm with that of a naive algorithm in terms of total computation cost (total CPU time) and total communication cost. Communication cost refers to the number of messages sent between server and clients.

5.2. Experimental Results

Figure 9 illustrates the effect of number of objects on computation cost. Both algorithms demonstrated poor performance when the number of objects was very high because then the algorithms needed to handle the updates of more objects. In addition, in the case of our CMRNN algorithm, the influence regions of more objects are calculated, which increases the computation time. However, when the number of objects was low or medium our algorithm performed better because of the safe region.

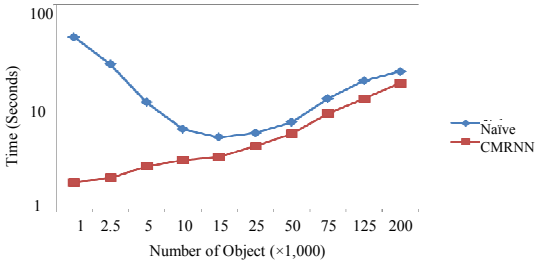

Figure 9. Effect of number of objects on computation time.

Figure 10 illustrates the effect of number of objects on communication cost. It shows that the messages sent by both algorithms tended to increase as the number of objects increased. However, our algorithm shows better performance because of the fact that when the query remains within the safe exit points, recomputation of query results is not required, which ultimately reduces the number of messages sent between query and server.

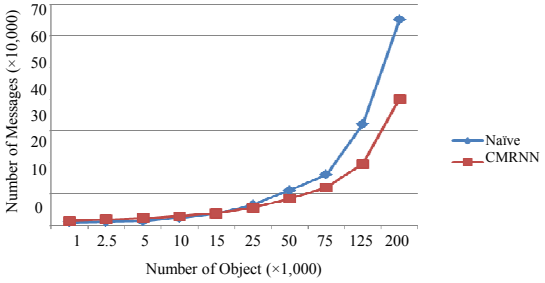

Figure 10. Effect of number of objects on communication cost.

In Figures 11 and 12, we study the effect of the speed of the query objects on the performance of the algorithms. The naïve algorithm incurred constant computation and communication costs because each client in the naïve algorithm asked for updates at every timestamp, regardless of the speed of the moving query. For our CMRNN algorithm, the experimental results indicate that the performance of the algorithm decreased gradually as the speed of the query object increased because queries left their safe regions more frequently. The figures show that the communication and computation costs of our algorithm increased when the speed was very high. This is because the algorithm had to run all three phases (find UOs, compute the influence region of UOs and compute safe exit points) frequently.

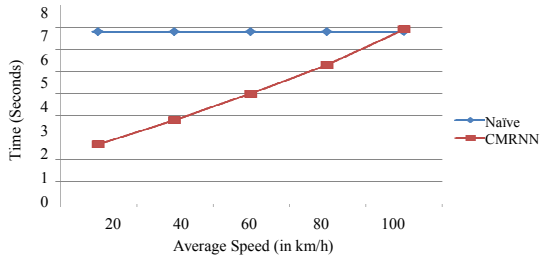

Figure 11. Effect of speed on computation cost.

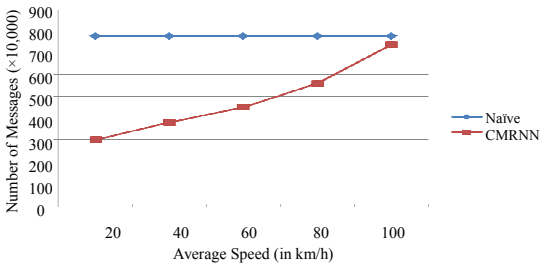

Figure 12. Effect of speed on communication cost.

Figures 13 and 14 compare the performance of CMRNN and the naïve algorithm WRT the number of queries. Figure 13 shows that for both algorithms, computation time increased as the number of queries increased, but CMRNN performed much better. The communication cost of the naïve algorithm does not depend on the number of queries because each object reports its location whenever it changes its location (refer to Figure 14). However, the computation and communication costs of CMRNN increase mainly because more UOs are required to be found if the number of queries is large. The fact is that when the number of UOs is large then the influence region of more objects need to be computed, which degrades the performance of CMRNN.

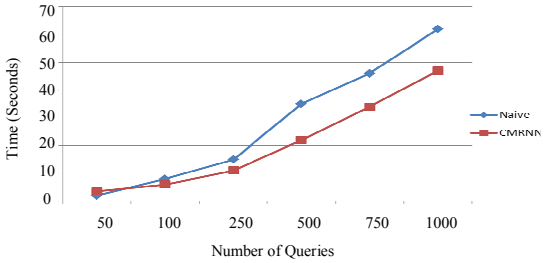

Figure 13. Effect of number of queries on computation cost.

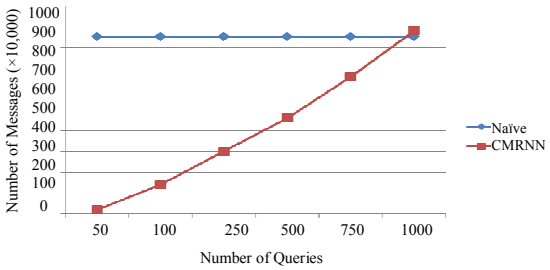

Figure 14. Effect of number of queries on communication cost.

Figure 15 shows the performance trend when k was modified in real time while all the other parameters were set to the default values shown in Table 2. The simulation reveals an interesting trend and comparison between the performance of CMRNN and the naive algorithm. The horizontal axis in Figure 15 represents real time. At specified instances, the value of k was incremented. This resulted in temporary performance degradation for our CMRNN algorithm and caused spikes. For short durations, the naïve algorithm outperformed CMRNN for those values of k. However, after CMRNN updated its new objects database and computed safe exit points, its query processing time began to decrease again and it subsequently outperformed the naive algorithm. This gain in performance was sustained until there was another change in the value of k. These results lead to the conclusion that while the naïve algorithm is a good choice for systems with rapid variations in k, CMRNN based systems yield higher performance gains for systems with stable k.

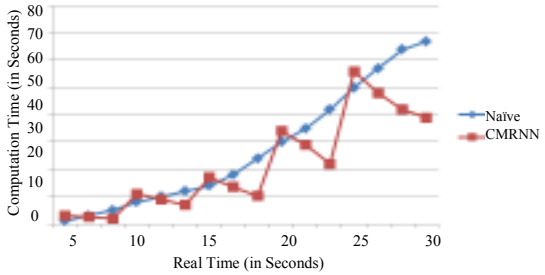

Figure 15. Effect of *k* on computation time.

6. Conclusions

In this paper, we studied the processing of continuous RKNN queries in road networks and proposed a new algorithm called CMRNN that computes the safe exit points for moving RNN queries in road networks. The results of experiments conducted using real datasets indicate that our algorithm significantly reduces computation costs as well as the communication costs between server and client. Consequently, CMRNN could be highly beneficial in real-life scenarios where mobile devices have limited network bandwidth and where the server demands a high throughput. There are several promising directions for future research. We plan to extend the algorithm to cater to scenarios in which both the queries and the data objects are moving.

Acknowledgements

We thank anonymous reviewers for their valuable comments and suggestions. This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2013R1A1A2A10012956 and NRF-2012R1A1A2043422).

References

[1] Benetis R., Jensen C., Karciauskas G., and SaltenisS., "Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects," *in Proceedings of International Symposium on Database Engineering and Applications*, pp. 44-53, 2002.

[2] Brinkhoff T., "A Framework for Generating Network-Based Moving Objects," *GeoInformatica*, vol. 6, no. 2, pp. 153-180, 2002.

[3] Cheema M., Brankovic L., Lin X., Zhang W., and Wang W., "Continuous Monitoring of Distance-Based Range Queries," *IEEE Transaction on Knowledge and Data Engineering*, vol. 23, no. 8, pp. 1182-1199, 2011.

[4] Cheema M., Lin X., Zhang W., and Zhang Y., "Influence Zone: Efficiently Processing Reverse K Nearest Neighbors Queries," *in Proceedings of the 27th International Conference on Data Engineering*, Hannover, Germany, pp. 577-588, 2011.

[5] Cheema M., Lin X., Zhang Y., Wang W., and Zhang W., "Lazy Updates: An Efficient Technique to Continuously Monitoring Reverse kNN," *the International Journal on Very Large Data Bases*, vol. 2, no. 1, pp. 1138-1149, 2009.

[6] Cheema M., Lin X., Zhang Y., Zhang W., and Li X., "Continuous Reverse K Nearest Neighbors Queries in Euclidean Space and in Spatial Networks," *the International Journal on Very Large Data Bases*, vol. 21, no. 1, pp. 69-95, 2012.

[7] Cho H. and Chung C., "An Efficient and Scalable Approach to CNN Queries in a Road Network," *in Proceedings of the 31st International Conference on Very Large Data Bases*, Trondheim, Norway, pp. 865-876, 2005.

[8] Cho H., "Continuous Range K-Nearest Neighbor Queries in Vehicular Ad Hoc Networks," *the Journal of Systems and Software*, vol. 86, no. 5, pp. 1323-1332, 2013.

[9] Cho H., Choe S., and Chung T., "A Distributed Approach to Continuous Monitoring of Constrained K-Nearest Neighbor Queries in Road Networks," *Mobile Information Systems*, vol. 8, no. 2, pp. 107-126, 2012.

[10] Cho H., Kwon S., and Chung T., "A Safe Exit Algorithm for Continuous Nearest Neighbor Monitoring in Road Networks," *Mobile Information Systems*, vol. 9, no. 1, pp. 37-53, 2013.

[11] Gao Y., Zheng B., Chen G., Lee W., Lee K., and Li Q., "Visible Reverse K-Nearest Neighbor Queries," *in Proceedings of the 25th International Conference on Data Engineering*, Shanghai, China, pp. 1203-1206, 2009.

[12] Kang J., Mokbel M., Shekhar S., Xia T., and Zhang D., "Continuous Evaluation of Monochromatic and Bichromatic Reverse Nearest Neighbors," *in Proceedings of the 23rd International Conference on Data Engineering*, Istanbul, Turkey, pp. 806-815, 2007.

[13] Kolahdouzan M. and Shahabi C., "Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases," *in Proceedings of the 30st International Conference on Very Large Data Bases*, Ontario, Canada, pp. 840-851, 2004.

[14] Korn F. and Muthukrishnan S., "Influence Sets based on Reverse Nearest Neighbor Queries," *ACM SIGMOD Record*, vol. 29, vo. 2, pp. 201-212, 2000.

[15] Li G., Li Y., Li J., Shu L., and Yang F., "Continuous Reverse K Nearest Neighbor Monitoring on Moving Objects in Road Networks," *Information Systems*, vol. 35, no. 8, pp. 860-883, 2010.

[16] Li P., Fan Y., and Du J., "An Efficient Technique for Continuous K-Nearest Neighbor Query Processing on Moving Objects in a Road Network," *in Proceedings of the 10th International Conference on Computer and Information Technology*, Bradford, UK, pp. 627-634, 2010.

[17] Real Datasets for Spatial Databases, available at: http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm, last visited 2013.

[18] Rizvi S. and Chung T., "JAM: Justifiable Allocation of Memory with Efficient Mounting and Fast Crash Recovery for NAND Flash Memory File Systems," *the International Arab Journal of Information Technolgy*, vol. 7, no. 4, pp. 395-402, 2010.

[19] Safar M., Ebrahimi D., and Taniar D., "Voronoi-Based Reverse Nearest Neighbor Query Processing on Spatial Networks," *Multimedia Systems*, vol. 15, no. 5, pp. 295-308, 2009.

[20] Song Z. and Roussopoulos N., "K-Nearest Neighbor Search for Moving Query Point," *in Proceedings of the 7th International Symposium Advances in Spatial and Temporal Databases*, California, USA, pp. 79-96, 2001.

[21] Stanoi I., Agrawal S., and Abbadi A., "Reverse Nearest Neighbor Queries for Dynamic Databases," available at: http://infolab.usc.edu/csci599/Fall2007/papers/b-2.pdf, last visited 2013.

[22] Sun H., Jiang C., Liu J., and Sun L., "Continuous Reverse Nearest Neighbor Queries on Moving Objects in Road Networks," *in Proceedings of the 9th International Conference on Web-Age Information Management*, Hunan, China, pp. 238-245, 2008.

[23] Tao Y., Papadias D., and Lian X., "Reverse Knn Search in Arbitrary Dimensionality," *in Proceedings of the 30th International Conference on Very Large Data Bases*, Ontario, Canada, pp. 744-755, 2004.

[24] Wang H. and Zimmermann R., "Snapshot Location-Based Query Processing on Moving Objects in Road Networks," *in Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, California, USA, 2008.

[25] Wu W., Yang F., Chan C., and Tan K., "Continuous Reverse K-Nearest-Neighbor Monitoring," *in Proceedings of the 9th International Conference on Mobile Data Management*, Beijing, China, pp. 132-139, 2008.

[26] Xia T. and Zhang D., "Continuous Reverse Nearest Neighbor Monitoring," *in Proceedings of the 22nd International Conference on Data Engineering*, Georgia, USA, 2006.

[27] Xuan K., Zhao G., Taniar D., and Srinivasan B., "Continuous Range Search Query Processing in Mobile Navigation," *in Proceedings of the 14th International Conference on Parallel and Distributed Systems*, Victoria, Australia, pp. 361-368, 2008.

[28] Yiu M., Mamoulis N., Papadias D., and Tao Y., "Reverse Nearest Neighbor in Large Graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 4, pp. 540-553, 2006.

[29] Yung D., Yiu M., and Lo E., "A Safe-Exit Approach for Efficient Network-Based Moving Range Queries," *Data Knowledge Engineering*, vol. 72, pp. 126-147, 2012.

[30] Zhang J., Zhu M., Papadias D., Tao Y., and Lee D., "Location-Based Spatial Queries," available at: http://infolab.usc.edu/csci599/Fall2007/papers/c-4.pdf, last visited 2013.

Muhammad Attique received the BCs degree in information and communication systems engineering from National University of Science and Technology, Pakistan, in 2008. He is currently doing MS degree in computer engineering from Ajou University, South Korea. His research interests include location based services, spatial queries in road network, moving objects and moving query processing in mobile networks.

Yared Hailu received the BCs degree in electrical engineering from Arba Minch University, Ethiopia in 2008; received Ms degree in computer engineering from Ajou University, South Korea 2013. His current research interest includes flash memory storages, embedded system, energy efficient computing and location based services.

Sololia GudetaAyele received BCs degree in electrical engineering from Haramaya University, Dire Dawa, Ethiopia in 2011. She is currently perusing MSC degree in Computer engineering in Ajou University, South Korea. Her research interest includes flash memory performance enhancement, location based services, spatial queries in road network, moving objects and moving query processing in mobile networks.

Hyung-Ju Cho received his BCs and MS degrees in computer engineering from Seoul National University in 1997 and 1999, respectively and his PhD degree in computer science from KAIST in 2005. He is currently a research assistant professor at the department of information and computer engineering, Ajou University, South Korea. His current research interests include moving object databases and query processing in mobile peer-to-peer networks.

Tae-Sun Chung received the BCs degree in computer science from KAIST, in 1995 and the MS and PhD degree in computer science from Seoul National University, in 1997 and 2002, respectively. He is currently an associate professor at school of information and computer engineering at Ajou University. His current research interests include flash memory storages, XML databases and database systems.