International Journal of
Geo-Information

Article

Efficient Processing of Continuous Reverse *k* Nearest Neighbor on Moving Objects in Road Networks

Muhammad Attique [1], Hyung-Ju Cho [2], Rize Jin [1] and Tae-Sun Chung [1,*]

[1] Department of Computer Engineering, Ajou University, Suwon 16499, Korea; attique@ajou.ac.kr (M.A.); rizejin@ajou.ac.kr (R.J.)

[2] Department of Software, Kyungpook National University, Sangju-si 37224, Korea; hyungju@knu.ac.kr

* Correspondence: tschung@ajou.ac.kr; Tel.: +82-31-219-1828

Academic Editors: Norbert Bartelme and Wolfgang Kainz
Received: 8 August 2016; Accepted: 23 November 2016; Published: 20 December 2016

Abstract: A reverse k nearest neighbor (RkNN) query retrieves all the data points that have q as one of their k closest points. In recent years, considerable research has been conducted into monitoring reverse k nearest neighbor queries. In this paper, we study the problem of continuous reverse nearest neighbor queries where both the query object q and data objects are moving. Existing state-of-the-art techniques are sensitive towards the movement of data objects, e.g., a candidate object must be verified whenever it changes its location. Further, insufficient attention has been given to the monitoring of RNN queries in dynamic road networks where the network weight changes depending on the traffic conditions. In this paper, we address these problems by proposing a new safe exit-based algorithm called CORE-X for efficiently computing the safe exit points of both query and data objects. The safe exit point of an object indicates the point at which its safe region and non-safe region meet, thus a set of safe exit points represents the border of the safe region. Within the safe region, the query result remains unchanged provided the query and data objects remain inside their respective safe regions. The results of extensive experiments conducted using real road maps indicate that the proposed algorithm significantly reduces communication and computation costs compared to the state-of-the-art algorithm.

Keywords: continuous monitoring; location-based applications; reverse nearest neighbor query; safe exit algorithm; mobile computing; road network

1. Introduction

The rapid technological advances in wireless networks and development of handheld devices equipped with location sensing technology (e.g., smart phones and tablets) have popularized location-based services in the past decades. These systems enable real-world applications such as mixed reality games, army strategy planning, object surveillance in wireless sensor networks, and enhanced emergency services [1,2]. The continuous movement of data objects demands new query processing techniques permitting frequent location updates. Although considerable effort has been devoted to moving-query processing [3–7], these studies have focused on range queries and nearest neighbor (NN) queries; there continues to be a deficiency of research addressing continuous reverse nearest neighbor (RNN) queries. Further, the majority of the research has been conducted in the Euclidean space, not on a road network.

Consider a query object q and a set of data objects O (e.g., places of accommodation, restaurants, and gas stations). We use $dist(o, q)$ and $dist(o, o')$ to represent the shortest distance from object o to query q and another data object o', respectively. A reverse k nearest neighbor (RkNN) query retrieves all of the data objects $o \in O$ for which q is one of their k closest objects.

R*k*NN queries are generally categorized into two types: monochromatic reverse *k*NN (MR*k*NN) queries and bichromatic reverse *k*NN (BR*k*NN) queries. In MR*k*NN, all moving data and query objects are of the same type. Applications of the continuous MR*k*NN include mixed reality games where the goal of each player is to shoot the nearest player. Each player must continuously monitor their own RNN to avoid being shot by other players. Given an object set O and a query q, the MR*k*NN query is formally defined as $\text{MR}k\text{NN}(q) = \{o \in O | q \in k\text{NN}(o)\}$, where kNN(o) is the *k*NN set of o.

Unlike, MR*k*NN queries, in BR*k*NN, query objects and data objects belong to two different types of objects. Applications of the continuous BR*k*NN query include decision support system. For example, consider an application where the goal is to determine the best location for opening a Chinese restaurant. The main deciding factor can be "How many users consider this potential location to be the nearest Chinese food restaurant?" Each candidate location for the Chinese food restaurant generates an RNN query; the results can then be compared to choose the best location. The above example employs BR*k*NN queries because restaurants and users belong to different types of objects. Given two different object sets O and S and a query $q \in S$, a BR*k*NN query is formally defined as $\text{BR}k\text{NN}(q) = \{o \in O | q \in k\text{NN}(o, S)\}$, where $k\text{NN}(o, S)$ denotes the *k*NN of o among all objects in dataset S.

Many real-life applications exist to illustrate the usefulness of moving R*k*NN queries, particularly in applications which require continuous monitoring of reverse nearest moving objects. For example, for enhanced 911 service, whenever a service center receives any emergency call, the team that is closest to the emergency call location is notified. Other examples could be taxi company dispatching taxis to the passengers, or army backup units interested in monitoring their closest units which may require any assistance.

The main challenge for continuous monitoring algorithms is maintaining the freshness of the query results when the query and data objects move freely and arbitrarily. A simple approach is to increase the frequency of updates; query q periodically sends requests to reevaluate the query results. However, this approach does not guarantee that results are fresh because the query results may become stale between each call to the server. Moreover, an excessive computational burden may be imposed on the server side as a high communication frequency increases the communication cost.

The existing state-of-the-art algorithm known as SAC [8] attempts to address the abovementioned problem and monitors the R*k*NN queries in two major steps, i.e., filtering and verification. In the filtering phase, using pruning rules, the part of the network that cannot contain any R*k*NN of q is pruned. The objects that are in the unpruned network are called the candidate objects *cand*. In the verification phase, the server monitors whether the object $cand \in candidate\ object\ list$ is the R*k*NN of q. More specifically, a *cand* is reported as an R*k*NN if and only if q is the closest object of *cand*. However, at each timestamp, the algorithm must verify the candidate object whenever it updates its location. The verification is considerably expensive because (a) verifying a *cand* requires checking whether q is one of the k closest objects of *cand* and (b) verification is required whenever a *cand* changes its location.

To address the aforementioned limitation, we present a safe exit-based approach for continuous monitoring of reverse k nearest queries in a road network where both query objects and data objects move arbitrarily. The proposed algorithm computes safe exit points for both query and data objects. The query result remains unchanged providing query and data objects reside within their respective safe regions. The safe exit technique reduces the two-way communication between client and server resulting in a reduction of the communication and computation cost. We first consider the continuous monitoring of reverse k nearest queries in static road networks where the network distance does not change over time. Then, we extend the proposed approach to dynamic road networks where the network weight such as the travel time changes depending on traffic conditions including traffic congestion or reversible lanes. Our key contributions are summarized as follows:

- We present a framework for continuous monitoring of R*k*NN queries where both query and data objects are moving on a road network.
- Our algorithm processes both MR*k*NN and BR*k*NN queries for arbitrary values of k.

- We present novel pruning rules that optimize the computation of safe exit points by minimizing the size of the unpruned network and number of objects.
- We demonstrate the extension of the proposed algorithm to a directed road network where each road segment has a particular orientation and to a dynamic road network where the network weight changes depending on the traffic conditions.
- We confirm through an extensive experimental evaluation that the proposed algorithm outperforms the state-of-the-art algorithm in terms of both communication and computation costs for the majority of the settings.

The remainder of this paper is structured as follows. Section 2 reviews the existing work on the continuous monitoring of R*k*NN queries in Euclidean space and road networks. Section 3 provides terminology definitions and describes the problem. Section 4 elaborates on the proposed safe exit algorithm (CORE-X) for computing safe exit points of moving R*k*NN queries in road networks. Section 5 presents a performance analysis of the proposed technique. Section 6 concludes this paper.

2. Related Work

An RNN query for moving objects searches for those objects that select object q as their nearest neighbor. The processing of RNN queries has become a recent emerging area of research. Many algorithms have been proposed for the monitoring of reverse nearest neighbors, especially in Euclidean space. However, there remains a lack of efficient algorithms for road networks. Our related work is divided into two sections: Section 2.1 addresses continuous RNN query processing in Euclidian space and Section 2.2 reviews continuous RNN query processing in road networks.

2.1. Algorithms for Continuous Reverse Nearest Neighbor Query Processing in Euclidian Space

Korn et al. [9] were the first to introduce the concept of RNN queries. They used a precomputing technique to search for RNNs. The main drawback of the preprocessing approach is that it is limited to supporting R*k*NN queries for a fixed number of k; moreover, it is inefficient when processing object movements. To address the shortcomings of the preprocessing technique, a new category of algorithms has been introduced, known as snapshot RNN algorithms. These snapshot algorithms have two main phases. The first phase is a filtering phase that focuses on the pruning of unnecessary objects; the second phase is refining, which verifies whether the remaining objects are valid query results. The two main filtering methods that have been developed for R*k*NN queries are 60-degree-pruning [10] and TPL-pruning [11] by Stanoi et al. and Tao et al., respectively.

Another category is continuous RNN query algorithms, which can produce incremental RNN results. A number of algorithms have also been proposed for efficient monitoring of NN queries, continuous range queries, and RNNs [3,8,11–17]. The existing continuous query processing methods focus on defining the monitoring region and updating the query results based on the moving object's location. Benetis et al. [18] were the first to study continuous RNN monitoring; however, their proposed scheme assumes that the velocities of the objects are known. Xia et al. [19] introduced an incremental and scalable algorithm for monochromatic RNN queries based on the 60-degree-pruning technique. In their approach, the monitoring region of a continuous RNN query is defined by six pie regions (determined by the query point and the six candidates) and six arc regions (determined by the six candidates and their nearest neighbors). The efficiency of this algorithm is superior to that of conventional methods because it identifies and processes the updates that fall into the monitoring region. Kang et al. [20] proposed a novel algorithm for monitoring continuous RNNs called IGERN. It is more efficient than 60-degree-pruning-based solutions because it monitors a small number of candidates rather than the entire space. Further, this method is applicable to both MR*k*NN and BR*k*NN queries. The tradeoff of this scheme is that it cannot be easily extended to continuous R*k*NN queries for $k > 1$. Therefore, the defined monitoring region applies only to continuous RNNs for $k = 1$.

Wu et al. [21] proposed a technique to monitor R*k*NNs that involves continuous filtering and continuous refining. They proposed a new refining framework called CRange-k that verifies the candidate objects by issuing kNN queries in each region rather than single nearest neighbor queries. Users that are closer than the *k*th NN in each region are the candidate objects; these candidate objects are verified if q is one of the k closest facilities. To monitor the results, for each candidate object, the method continuously monitors the circular region around it that contains the k nearest facilities. Cheema et al. [8,22] proposed several schemes for the monitoring of continuous RNNs. In [22], they developed an algorithm for continuous BR*k*NN queries. They used the concept of the influence zone where the query object is static and the data objects are moving. In [8], they proposed a new framework based on safe regions for both Euclidean and road networks where both query and data objects are 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 provided the query and objects remain in their respective safe regions.

2.2. *Algorithms for Continuous Reverse Nearest Neighbor Query Processing in Road Networks*

In recent years, reverse neighbor query processing in road networks has received attention by the spatial database systems research community. Yiu et al. [17] first addressed the issue of RNN in road networks (they represented road networks as graphs) and proposed an algorithm for both MR*k*NN and BR*k*NN queries. Safar et al. [23] presented a framework for RNN queries based on network Voronoi diagrams (NVDs) to efficiently process RNN queries in road networks. However, their scheme is not suitable for continuous RNN queries because NVDs change whenever a dataset changes its location, resulting in high computation costs.

Sun et al. [16] studied the continuous monitoring of bichromatic RNN queries. They associated a multiway tree with each query to define the monitoring region and only updates in the monitoring region affect the results. However, this method is limited to bichromatic queries and does not apply when $k > 1$. Moreover, their proposed scheme assumes that the query objects are static. Li et al. [24] proposed a novel algorithm for continuous monitoring of R*k*NNs based on a dual layer multiway tree (DLM tree) where they introduced several lemmas to reduce the monitoring region and filter the candidate objects. Their continuous monitoring of R*k*NN method comprises two phases: the initial result generation phase and the incremental maintenance phase. Cheema et al. [8] proposed a safe region approach for the monitoring of continuous MR*k*NN and BR*k*NN queries in Euclidean and road networks as mentioned in Section 2.1 and devised pruning rules that reduce the monitoring region.

Gotoh et al. [25] presented a simple routing method to process BR*k*NN queries. Wang et al. [26] proposed an influence zone-based algorithm for monitoring continuous R*k*NN in a road network. Attique et al. [27] proposed a safe exit-based scheme for continuous monitoring of RNN queries. However, this approach is applied to BR*k*NN queries; moreover, it does not address the movement of data objects. In this paper, we develop a scheme that is applicable to both MR*k*NN and BR*k*NN queries; it can also address the movement of both query objects and data objects.

This paper is an extended version of our previous work on the monitoring of R*k*NN queries for moving data objects and queries [28]. However, we present five relevant extensions that were not investigated in our preliminary work [28]. The first extension monitors the pruned objects (Section 4.2.2). The second extension studies adaptations of the proposed algorithm to address a directed road network (Section 4.3.1). In the third extension, we extend CORE-X to a dynamic road network where the network distance changes depending on the traffic conditions (Section 4.3.2). The fourth extension studies bichromatic R*k*NN queries (Section 4.3.3). In the fifth section, we conduct an extensive experimental study for both static and dynamic road networks (Section 5). In addition to these major extensions, we also present a time complexity analysis of the proposed algorithm that was not presented in our previous work (Section 4.4).

3. Preliminaries

Section 3.1 defines the terms and notation used in this paper. Section 3.2 provides the problem description based on an example.

3.1. Definition of Terms and Notations

Road Network: A road network is represented by a weighted undirected graph $G = (N, E, W)$, where N includes the set of nodes $N = \{n_1, n_2, \cdots, n_{|\mathrm{N}|}\}$, E is a set of edges that connects two distinct nodes $E = \{e_1, e_2, \cdots, e_{|\mathrm{E}|}\}$, and $W(e)$ denotes the weight of an edge e. An edge between two nodes is denoted by $e(n_s, n_\mathrm{e})$, where n_s and n_e are referred to as boundary nodes. A boundary node with a smaller node id is referred to as the base node of the sequence. This study assumes that $n_\mathrm{s} \leq n_\mathrm{e}$. Therefore, n_s is the base node. Each edge connecting two distinct nodes has a positive weight that represents the cost of traveling the length of the edge, e.g., the time required to travel along the edge.

Segment: Segment $s_{(p_1, p_2)}$ is a 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 where the nodes are the end points of the edge. The weight of a segment $s_{(p_1, p_2)}$ is denoted by W(s).

Figure 1 presents an example of an undirected road network with seven nodes, n_1 to n_7. Several edges and segments are illustrated with their respective weights. For example, the edge $e(n_1, n_2)$ consists of segments $s_{(n_1, o_1)}$ and $s_{(o_1, n_2)}$, which have weights "1" and "2," respectively. There are seven data objects in this example $\{o_1, o_2, \ldots, o_7\} \in O$ and a single query object q. Query q and the data objects are indicated by triangles and rectangles, respectively. Given two points, p_1 and p_2, the shortest path distance $dist(p_1, p_2)$ is the minimum distance between p_1 and p_2. In Figure 1, the shortest path from q to o_3 is $q \rightarrow n_3 \rightarrow n_6 \rightarrow o_3$; $dist(q, o_3) = 11$. Table 1 presents the notations used in this paper.

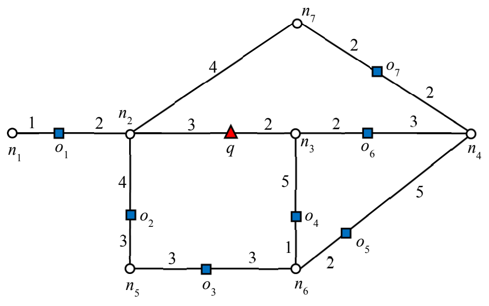


Figure 1. Example of a road network.

Table 1. Summary of notations used in this paper.

Notation	Definition
G = (N, E,W)	Graph model of road network
$dist(p_s, p_e)$	Length of the shortest path from p_s to p_e, where p_s and p_e represent start and end points, respectively
n_i	Node in road network
$e(n_s, n_e)$	Edge in edge set E, where n_s and n_e are start and end points of the edge
$W(e)$	Weight of edge $e(n_s, n_e)$
q	Query point in road network
k	A number that represents q can be among k number of closest facilities to a data object o
O	Set of objects $O = \{o_1, o_2, \ldots, o_n\}$
O^+	Set of answer objects $O^+ = \{o_1^+, o_2^+, \ldots, o_k^+\}$
O^-	Set of non-answer object $O^- = \{o_{k+1}^-, o_{k+2}^-, \ldots, o_{\|O\|}^-\}$
IR^+	Influence region of answer objects
IR^-	Influence region of non-answer objects
o_f^+	Farthest answer object to point $p \in G$, such that $d(p, o_f^+) = MIN(d(p, o_1^+), d(p, o_2^+), \ldots, d(p, o_{\|o^+\|}^+)$
o_n^-	Nearest non-answer object to point $p \in G$, such that $d(p, o_n^-) = MIN(d(p, o_1^-), d(p, o_2^-), \ldots, d(p, o_{\|o^-\|}^-)$
p_a	Anchor point that corresponds to start point of expansion
UO	Set of useful objects required to compute safe region of q
Ω	Safe exit point where the safe and non-safe region of q or o intersects
n_β	Boundary node corresponding to start (n_s) or end (n_e) point of edge
A_p	Set of kNNs at point p
R_p	Set of RkNNs at point p

3.2. Problem Description

In this paper, we primarily address the problem of continuous monitoring of R*k*NN queries on moving queries and data objects in road networks. To provide a clear explanation, we use the road network example illustrated in Figure 1, where there are seven objects, o_1 to o_7, and a query q in a road network. For explanation, we consider MR*k*NN queries where both data objects and query objects belong to the same data type. However, our method can also be extended to monitor continuous BR*k*NNs queries. In the remainder of the paper, we use R*k*NN query to refer to an MR*k*NN query unless mentioned otherwise. Let us assume that a moving query requests one RNN ($k = 1$) at a certain point p_1. To obtain one RNN, we traverse the road network from the active edge that contains point q. For each data object $o \in O$. encountered, we issue a verification query *verify*(o, k, q) that checks if it is an RNN. If there exists another object o' such that $dist(o, o') < dist(o, q)$, then o is not an RNN. Otherwise, o is inserted into the RNN result set denoted as R_p. The expansion in each path terminates once k objects have been found in that direction. For obtaining an RNN at point p_2, the simple method is to repeat the procedure executed at p_1. However, the use of recomputation whenever query q or any data object changes its location significantly degrades the performance of the algorithm. To address this issue, we introduce the safe exit approach.

In the proposed framework, the server computes the safe exit points for each moving object and query object. Because we are addressing a complex problem where both query objects and data objects are moving, it is necessary to compute the safe exit points for each moving data object. The server maintains a set of moving queries and a set of moving objects. The query results will be the same until all the objects lie inside their respective safe exit points. Whenever a query or data object leaves its safe exit points, the server recomputes the R*k*NN and safe exit points for the query and data objects. However, the computation of safe exit points for all of the data objects degrades the performance of the algorithm. To address this issue, we devised pruning rules that allow us to compute the safe exit points for only the unpruned objects.

As mentioned previously, we are addressing the problem of continuously monitoring RNN queries where both query and data objects are moving freely in a road network. Therefore, the following events can change the query result.

Event 1: The query object q moves outside of its respective safe exit points.

Event 2: An object $o \in UO$ moves outside of its respective safe exit points.

Event 3: The movement of pruned objects changes the R*k*NN set.

Figure 2 illustrates the request flow for processing R*k*NN queries. Initially, the query object q (client) issues an RkNN query to server along with the current location (Step 1). Upon receiving a request, the server computes RkNN result set $\mathrm{R_k}$ and safe exit points for query and unpruned data objects (Step 2 and 3). The query results remain valid, until any of the abovementioned three events occurs. To monitor Event 1, the query object checks its location against the received safe exit points (Step 5). If query object travels beyond any safe exit, client re-issues an RkNN query to the server for an updated result and its safe exit points. A server monitors the movement of data objects to verify Events 2 and 3 (Step 6). If either Event 2 or 3 occurs, the server recomputes the query results and notifies the client.

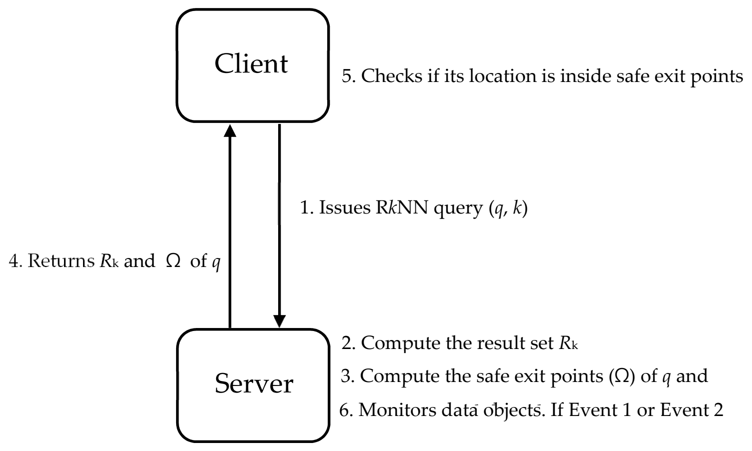

Client
5. Checks if its location is inside safe exit points
1. Issues RkNN query (q, k)
4. Returns Rk and Ω of q
Server
2. Compute the result set Rk
3. Compute the safe exit points (Ω) of q and
6. Monitors data objects. If Event 1 or Event 2

Figure 2. Request flow for processing R*k*NN queries.

4. Safe Exit Algorithm for Moving R*k*NN Queries and Moving Objects

In this section, we develop techniques to monitor moving R*k*NN queries and moving objects in a road network. In Section 4.1, we present an algorithm for computing the safe region for moving object q. The monitoring of data objects is described in Section 4.2. We present the extensions of the proposed approach to other variants of R*k*NN queries in Section 4.3. Finally, Section 4.4 analyzes the time and space complexities of CORE-X.

4.1. Safe Region for Moving q

In this section, we present a new safe exit algorithm that addresses the issue for moving R*k*NN queries and moving objects in a road network. Algorithm 1 depicts the skeleton of the proposed safe exit algorithm for computing safe regions. It consists of three phases: (1) determining the useful objects that can contribute to the safe region; (2) computing the influence region of the useful objects; (3) computing the safe exit points for the query objects.

Algorithm 1: Computation of Safe Regions (skeleton)

```
Input: $O$: data objects, $q$: query object, $k$: integer number
Output: $SR$: Safe Region
1:    Object set $O' \leftarrow roadnetwork(O, q, k)$
2:    Object set $O^+ \leftarrow \{o^+ \in O' | dist(o,q) < dist(o, o_{k+1})\}$
3:    Object set $O^- \leftarrow \{o^- \in O' | dist(o,q) > dist(o, o_k)\}$
4:    While $O^+$ or $O^-$ is non-empty do
5:      Object $o = pickobject(O^+, O^-)$
6:      If $o \in O^+$ then
7:        $IR^+ \leftarrow computeIR(O^+, O^-, k)$
8:        $SR \leftarrow SR \cap IR^+$
9:      Else
10:       $IR^- \leftarrow computeIR(O^+, O^-, k)$
11:       $SR \leftarrow SR - \cup IR^-$
12:   End while
13:   Return $SR$
```

Algorithm 1 presents the skeleton of the proposed idea. The algorithm begins by determining the answer and non-answer objects. The details of the methodology are explained in Section 4.1.1. Then, in Phase 2, the algorithm computes the influence region of the answer objects and non-answer objects (Section 4.1.2). Finally, it computes the safe region by performing intersection and set difference operations on the road segments (Section 4.1.3).

4.1.1. Determining Useful Objects

This phase aims at determining potential objects that could contribute to the computation of the safe region. The goal is to retrieve a small set of data objects to reduce the computation overhead. In our study, the data objects are divided into two categories: answer objects (denoted by O^+) and non-answer objects (denoted by O^-).

Definition 1. *An object o is called an answer object if $dist(o,q) \leq dist(o,o')$ where o' is any other object in the road network. Similarly, we can generalize this definition for RkNN: an object* o *is called an answer object if $dist(o,q) \leq dist(o,o_{k+1})$, where o_{k+1} is the (k+1)th NN object of o. That is, we can say that all answer objects are RkNNs of query q, therefore, $o^+ \in R_k$.*

Definition 2. *An object o is called a non-answer object if $dist(o,q) > dist(o,o')$, where o' is any other object in the road network.*

Similarly, we can generalize this definition for R*k*NN: an object o is called a non-answer object if $dist(o,q) > dist(o,o_k)$, where o_k is the *k*th NN object of o. That is, we can say that all non-answer objects are not R*k*NN of query q, therefore, $o^- \notin R_k$.

A simple method for retrieving the R_k set is to traverse the network from q, and for each data object $o \in O$ encountered, issue a nearest-neighbor query. If $q \in NN(o)$, q is the closest object to o. Consequently, $o^+ \in R_k$. However, this approach must evaluate all the data objects because the size of R_k is not fixed and the road network may contain points that are far from q. To avoid unnecessary road network exploration, we present the pruning lemma.

Before presenting the lemma, it is necessary to define closed nodes. A node n is called a closed node if there exists an object o such that $dist(n,o) < dist(n,q)$. The object o is called a blocking object because it causes node n to be a closed node. In Figure 1, node n_2 is a closed node because $dist(n_2,o_1) < dist(n_2,q)$, which makes o_1 a blocking object.

Lemma 1. *An object o cannot be the RNN of q if the shortest path between q and o contains a closed node with a blocking object o', where $o' \neq o$.*

Proof. Let us assume that there exists a closed node n on the shortest path between o and q. The shortest distance between o and q is $dist(o,q) = dist(n,o) + dist(n,q)$. Let o' be the blocking object and $dist(o,o') = dist(n,o) + dist(n,o')$. As we know $dist(n,o') < dist(n,q)$, therefore, $dist(o,o') < dist(o,q)$. Therefore, o cannot be an RNN of q.

In Figure 1, the data object o_2 cannot be an RNN of q because the shortest distance between o_2 and q passes through n_2. Because $dist(n_2,o_1) = 2$ and $dist(n_2,q) = 3$, data object o_1 is closer to o_2 than q. The above lemma can be easily extended to R*k*NN. An object o cannot be the R*k*NN of q if the shortest path between q and o contains a node n and there exist k data objects o' such that for every object o', $dist(n,o') < dist(n,q)$, where $o' \neq o$.

Algorithm 2 presents the pseudo code for determining the answer objects. CORE-X traverses the network around q in a similar fashion to Dijkstra's algorithm; using Lemma 1, it eliminates the nodes that may not lead to RNNs. The algorithm begins by exploring the active edge where query object q is located and expands the network in an increasing order of distance from the query object q. Each entry in the queue has the form $\langle p_a, edge\rangle$, where p_{a} indicates the anchor point on the edge. For an active edge, q becomes the anchor point. Otherwise, neither of the boundary nodes of the edge, i.e., n_s or n_e, becomes the anchor point. If the desired number of answer objects is not found on an active edge, the edges adjacent to the boundary nodes are enqueued. The edges are popped in an increasing order of distance from q. The traversal of the edges is terminated when the queue is exhausted. Line 4 initializes a *queue* by inserting the active edge. If an edge contains a data object

o, we must verify whether $o \in \mathrm{R}k\mathrm{NN}(q)$. Thus, the algorithm issues a *verify*(o, k, q) query (Line 10). The verification query checks if q is among the kNNs of data object o by applying a range-NN query around object o with the range set to $dist(o, q)$. If $q = k\mathrm{NN}(o)$, o is the RkNN of q. Therefore, o is added to the result set R_k (Line 12). If the edge does not contain any data object o, the algorithm continues its expansion and enqueues the adjacent edges of the boundary node.

Algorithm 2: Answer object(q, k)

```
Input: q: query location, k: integer number
Output: R_k: query result (answer objects)
1:   $queue \leftarrow \varnothing$                 /* queue is a priority queue with edges ordered by distance to q*/
2:   $A_k \leftarrow \varnothing$                    /* set of answer objects*/
3:   $visited \leftarrow \varnothing$               /*stores information of visited edges */
4:   $queue$.push($q$,edge$_{active}$)   /* edge$_{active}$ indicates active edge */
5:   While $queue$ is not empty do
6:       $\langle p_a, edge\rangle \leftarrow queue.pop()$
7:       If $\langle p_a, edge\rangle \notin visited$ then
8:           $visited \leftarrow visited \cup \{edge\}$
9:               If edge contains a data object $o$
10:                  $k$NN($o$): verify($o$, $k$, $q$)
11:                  If $q$ discovered by verification
12:                      $R_k \leftarrow R_k \cup o$
13:              Else
14:                  $queue.push\langle n_\beta, edge\rangle$
15:  End while
16:  Return $R_k$
```

Let us consider the example presented in Figure 1, where query q requested one RNN (k = 1). We will consider this example throughout this section. The algorithm starts expansion from the active edge, which is $e(n_2, n_3)$. Because no data object is discovered on that edge, edges adjacent to the boundary nodes will be added to the queue. The edges adjacent to the closest boundary node are enqueued first. Therefore, edges $e\{(n_3, n_4), (n_3, n_6)\}$ are enqueued. Data object o_6 is first discovered on (n_3, n_4), which triggers the verification query verify(o_6, 1, q). Because q = NN(o_6), o_6 is added to the result set R_k. Recall that by Lemma 1, node n_4 is a closed node with object o_6 as the blocking object. Therefore, all the other objects for which the shortest distance passes through n_4 except o_6 are automatically pruned. Then, data object o_4 is discovered on $e(n_3, n_6)$, by verification of o_4, $q \neq \mathrm{NN}(o_4)$; therefore, the search also terminates in this direction. Next, the edges adjacent to n_2 are added to the queue, $e\{(n_1, n_2), (n_2, n_5), (n_2, n_7)\}$ and data object o_1 is retrieved on $e(n_1, n_2)$. The data object o_1 is verified and added to the result set because q = NN(o_1). The search terminates as no further expansion is required in this direction because node n_2 is a closed node.

Next, we determine the non-answer objects that can contribute to the safe region. Useful non-answer objects $UO^- \in O^-$ are objects for which any $o^+ = NN(o^-)$. In other words, UO^- are RNNs of answer objects. RNNs of Useful Objects (UO) can be determined by the same algorithm making these modifications: (1) the anchor point is o^+ instead of q and (2) verification query *verify*(o, k, q) is modified to *verify*(o, k, o^+). The algorithm reuses the results of the range-NN queries issued at the data objects to avoid multiple verification of the same data objects. When the computation is completed, the cached query results are removed to reduce the memory consumptions.

Finally, we can conclude that useful objects are: (1) all answer objects and (2) RNNs of answer objects. Now, let us determine the UO objects for k = 2 in the given example in Figure 1. For explanation, Table 2 displays the kNNs of each data object.

From Table 2 it is clear that o_1 and o_6 are both answer objects because q is one of the 2NNs. Now, from the non-answer objects, we can see that o_2 is the RNN of o_1 because o_1 = 2NN(o_2). Similarly, o_7 is the RNN of o_6. Hence, the set of useful objects is $UO = \{o_1, o_2, o_6, o_7\}$. Data objects o_3, o_4, and o_5 have been pruned.

Table 2. Summary of 2NNs for $o \in O$.

Data Object	2NNs	Distance
o_1	(q, o_2)	$(5, 6)$
o_2	(o_3, o_1)	$(6, 6)$
o_3	(o_4, o_5)	$(4, 5)$
o_4	(o_5, o_3)	$(3, 4)$
o_5	(o_4, o_3)	$(3, 5)$
o_6	(q, o_7)	$(4, 5)$
o_7	(o_6, o_5)	$(5, 7)$

.

4.1.2. Computing Influence Region for Useful Objects

After we retrieve the set of useful objects, the next step is to compute the influence region of the answer and non-answer objects.

Influence Region of Answer Objects

The influence region of the answer objects is defined as:

$$IR^+ = \{p | dist(o^+, p) \leq dist(o^+, o_{k+1})\} \tag{1}$$

Here, $o_{\mathrm{k+1}}$ denotes the (k+1)th nearest neighbor of o. By definition, the influence region of the answer objects contains all the points for which $q = NN(o^+)$. That is, it contains all of the points where object o remains o^+.

The influence region of answer objects can be computed by exploring the network around the answer object in a manner similar to that explained in Section 4.1.1. The exploration terminates with the discovery of the (k+1)th nearest neighbor of the answer object. The influence region will be indicated by *range*(o, d), where d is the $dist(o, o_{k+1})$.

Figure 3 indicates the influence region of object o_1 for the example scenario discussed above. The expansion of the road network starts from o_1 until it finds o_{k+1}, which is 3NN in this example. Object o_7 is 3NN of o_1 and $dist(o_1, o_7) = 8$. The algorithm will issue a $range(o_1, 8)$ query that marks all of the points as the influence region of o_1 as shown in Figure 3. Similarly, the influence region of o_6 is indicated in Figure 3. The data object o_4 is the 3NN of o_6 and $dist(o_6, o_4) = 7$. Therefore, the $range(o_6, 7)$ query marks all the points within the distance of "7" from o_6. The bold lines in Figures 3 and 4 indicate the influence region of o_1 and o_6, respectively.

Influence Region of Non-Answer Objects

The influence region of non-answer objects is defined as:

$$IR^- = \{p | dist(o^-, p) \leq dist(o^-, o_k)\} \tag{2}$$

The influence region of non-answer objects can be computed from the distance between the non-answer object and kth object. Here o_{k} denotes the kth nearest neighbor of o. In other words, the influence region of non-answer objects contains all the points where object o remains o^-. The influence region of non-answer objects is computed in the same manner as the influence region of answer objects, the only difference is that the algorithm explores and computes the distance to the kNN object instead of (k + 1)NN.

Consider object o_7 in Figure 1: 2NNs of $o_7 = (o_6, o_5)$ with weights (5, 7). In this example, o_5 is the second NN of o_7 and $dist(o_7, o_5) = 7$. Therefore, the influence region of object o_7 is "7" units from o_7 in every connected direction. Similarly, we can compute the influence region of o_2. The bold lines in

Figures 5 and 6 indicate the influence region of o_2 and o_7, respectively. The NNs for answer objects change when the query object moves outside the influence region, whereas in the case of non-answer objects, the result changes when the query object moves inside an influence region. That is, the NNs of the answer object remain the same until the query object lies inside the influence region and for non-answer objects, the NNs remain the same until query object lies outside the influence region. Note that once an influence region is computed, it remains valid as long as q and UO lie within their respective safe regions. Hence, recomputation of the influence region is only required when either q or UO leave their respective safe regions.

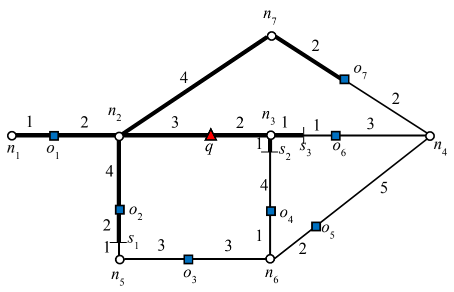


Figure 3. Influence region of o_1.

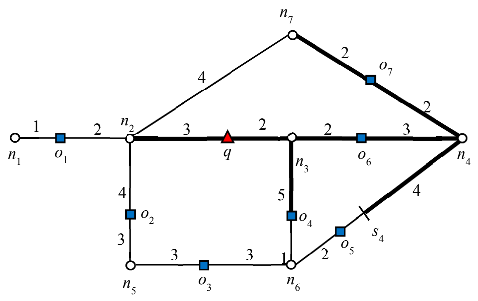


Figure 4. Influence region of o_6.

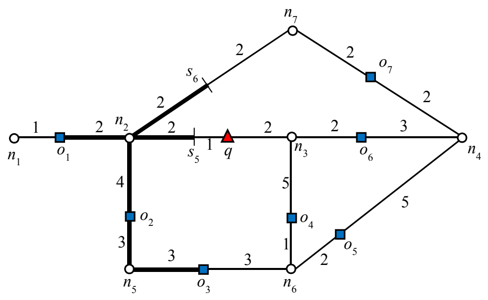


Figure 5. Influence region of o_2.

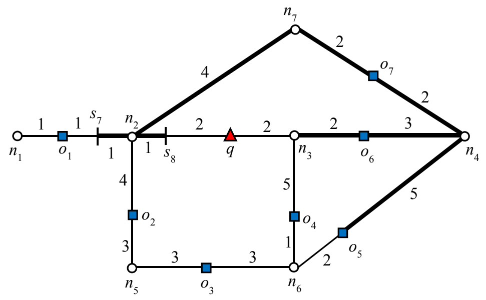


Figure 6. Influence region of o_7.

Algorithm 3 computes the influence region of answer objects and non-answer objects.

Algorithm 3: Computation of Influence Region (O^+, O^-, k)

```
Input: $O^+$: answer objects set, $O^-$: non-answer objects set, $k$: integer number
Output: $IR^+$ influence region of answer objects, $IR^-$ influence region of non-answer objects
1:   $IR^+ \leftarrow \varnothing$, $IR^- \leftarrow \varnothing$
     /* Influence Region of answer objects */
2:   For each $O^+$ do
3:      Expand the road network until $o_{k+1}$;
4:        $d \leftarrow dist(o, o_{k+1})$
5:      Mark range(o, d);
6:      $IR^+ \leftarrow IR^+ \cup IR^+$;
7:   End for
     /* Influence Region of non-answer objects */
8:   For each $O^-$ do
9:      Expand the road network until $o_k$;
10:     $d \leftarrow dist(o, o_k)$;
11:     Mark range($o$, $d$);
12:     $IR^- \leftarrow IR^- \cup IR^-$;
13:  End for
14:  Return $IR^+$ and $IR^-$
```

4.1.3. Computation of Safe Exit Points

The safe region SR of a query q is defined as follows:

$$SR = \{\cap IR^+ - \cup IR^-\} \tag{3}$$

where IR^+ denotes the influence region of answer objects and IR^- denotes the influence region of non-answer objects. From the definition, we can see that any point that lies in the intersection of the influence region of answer object O^+ is regarded as a safe region and that any point p that lies in the influence region of non-answer object O^- should be excluded.

In a road network, the safe region is expressed by a set of segments. In Figure 1, recall that o_1 and o_6 are answer objects and o_2 and o_7 are useful non-answer objects. By applying the above formula, the safe region can be expressed as:

$$SR = \{(IR(o_1) \cap IR(o_6)) - (IR(o_2) \cup IR(o_7))\} \tag{4}$$

Table 3 summarizes the computation of the safe exit points for the query object q in Figure 1. In the previous section, we computed the influence region *IR* of all useful objects, which was described by set of segments. To obtain the safe region, we must perform intersection and set difference operations on the road segments. We obtain segments $e(n_2, n_3)$, $s\{(n_3, s_2), (n_3, s_3)\}$ from the intersection of $IR(o_1)$ and $IR(o_6)$. As o_2 and o_7 are non-answer objects, the safe region must exclude the $IR(o_2)$ and $IR(o_7)$. Thus, the safe region would be the segments $s\{(s_5, n_3), (n_3, s_2)\}$. The set of boundary points s_2 and s_5 becomes the set of safe exit points. Figure 7 presents the safe exit points of q.

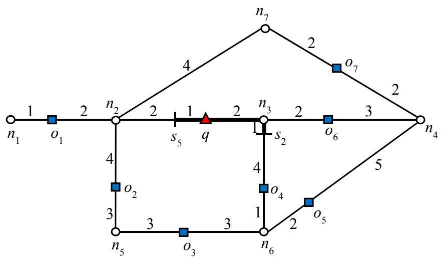


Figure 7. Safe exit points of q.

Table 3. Computation of safe region of q.

O	Influence Region	$IR(o_1) \cap IR(o_4)$	$\{IR(o_1) \cap IR(o_4)\} - \{IR(o_2) \cup IR(o_5)\}$
o_1	$e\{(n_1, n_2), (n_2, n_3), (n_2, n_7)\}s\{(n_2, s_1), (n_3, s_2), (n_3, s_3), (n_7, o_7)\}$		
o_6	$e(n_2, n_3), (n_3, s_4), (n_4, n_7)s\{(n_3, s_4), (n_3, o_4), (n_4, s_6)\}$	$e(n_2, n_3)s\{(n_3, s_2), (n_3, s_3)\}$	
o_2	$e(n_2, n_5)s\{(o_1, n_2), (n_2, s_5), (n_2, s_6), (n_5, o_3)\}$	$e(n_2, n_3)s\{(n_3, s_2), (n_3, s_3)\}$	$s\{(s_5, n_3), (n_3, s_2), (n_3, s_3)\}$
o_5	$e(n_2, n_7), (n_4, n_7), (n_3, n_4)s\{(n_4, o_5), (n_2, s_7), (n_2, s_8)\}$	$e(n_2, n_3)s\{(n_3, s_2), (n_3, s_3)\}$	$s\{(s_5, n_3), (n_3, s_2)\}$

4.2. Monitoring of Data Objects

As mentioned previously, we are studying a special case where both data objects and query objects can move randomly in a road network. The R*k*NN for a query *q* can be changed by the motion of data objects. In this section, we present techniques to continuously monitor the data objects. As discussed in Section 4.1.1, the data objects are divided into types: (1) UO objects and (2) pruned objects. Since the movement of UO is critical and a slight movement can change the results, we compute the safe exit points of all the UO, whereas for pruned objects, we use the monitored network based on the influence region computed in Section 4.1.2. The computation of safe exit points of all the pruned objects will also increase the computation cost.

4.2.1. Computation of Safe Exit Points of Useful Objects (*UO*)

Recall that answer objects are those objects whose query object *q* is one of its *k*NN objects. This means that an answer object lies within the safe exit points as long as its *k*NN objects remain the same. Similarly, non-answer useful objects are those objects for which any of the answer objects are among its *k*NN. This means that all useful objects lie inside the safe exit points as long as their respective *k*NNs are the same. Therefore, we must monitor the nearest neighbors of moving objects in a road network. We use the approach of Cho et al. [3], Safe Exit Algorithm (SEA), which can efficiently compute the safe exit points of a moving nearest neighbor query on a road network.

First, we formally define a set of safe exit points for a moving NN query in the road network. Let Ω be the set of safe exit points for a kNN query point q and $O = \left\{o_1, o_2, \ldots, o_{|O|}\right\}$ be the set of objects of interest to q. Assume that the answer set (i.e., O^+) of q and its non-answer set (i.e., O^-) are $O^+ = \{o_1^+, o_2^+, \ldots, o_k^+\}$ and $O^- = \left\{o_{k+1}^-, o_{k+2}^-, \ldots, o_{|O|}^-\right\}$, respectively. Then, it holds that $d(q, o^+) \leq d(q, o^-)$ for an answer object $o^+ \in O^+$ and a non-answer object $o^- \in O^-$. Finally, Ω is defined as follows:

$$\Omega = \left\{\omega \in G \middle| MAX(d(\omega, o_1^+), d(\omega, o_2^+), \ldots, d(\omega, o_k^+)) = MIN(d\left(\omega, o_{k+1}^-\right), d\left(\omega, o_{k+2}^-\right), \ldots d\left(\omega, o_{|o|}^-\right))\right\} \quad (5)$$

where $MIN()$ and $MAX()$ return the minimum and maximum values of the input array, respectively. That is, a safe exit point ω is the center point, i.e., $AX(d\left(\omega, o_1^+\right), \ldots, d\left(\omega, o_k^+\right)) = MIN(d\left(\omega, o_{k+1}^-\right), \ldots, d\left(\omega, o_{|o|}^-\right))$, between the farthest answer object and the nearest non-answer object.

The following two main lemmas are presented to determine whether a safe exit point exists in the segment.

Lemma 2. *If* $A_{n_\beta} \cup O_{(n_\beta, p_a)} \neq A_{p_a}$, *there is a safe exit point* ω *in the segment.*

Proof. Please refer to [3]. This lemma indicates that a safe exit point in a segment exists if the set of answer objects at n_β is not equal to the set of answer objects at p_a.

Lemma 3. *If* $A_{n_\beta} \cup O_{(n_\beta, p_a)} = A_{p_a}$, *there is no safe exit point in the segment.*

Proof: Please refer to [3]. This lemma indicates that a safe exit point in a segment does not exist if the set of answer objects at n_β is equal to the set of answer objects at p_a.

For this purpose, we introduce o_n^- and o_f^+. For simplicity, let us assume that $A_{n_\beta} \cup O_{(n_\beta, p_a)} - A_{p_a}$ corresponds to $O^- = \{o_1^-, o_2^-, \ldots, o_{|O^-|}^-\}$ and A_{p_a} corresponds to $O^+ = \{o_1^+, o_2^+, \ldots, o_{|O^+|}^+\}$. Then, at a point $p \in \left[n_\beta,\ p_a\right]$, o_n^- is referred to as the nearest non-answer object to p such that $d(p, o_n^-) = MIN(d(p, o_1^-), d(p, o_2^-), \ldots, d(p, o_{|o^-|}^-))$. Similarly, at a point $p \in \left[n_\beta,\ p_a\right]$, o_f^+ is referred to as the farthest answer object from p such that $d(p, o_f^+) = MAX(d(p, o_1^+), d(p, o_2^+), \ldots, d(p, o_{|o^+|}^+))$. The midpoint between o_n^- and o_f^+ becomes a safe exit point ω. That is, $d(\omega, o_n^-) = d(\omega, o_f^+)$.

We now discuss the computation of the safe exit points for the answer object o_1 in the example road network given in Figure 1. Table 4 summarizes the computation of the safe exit points for data object o_1 for the example scenario in Figure 1. Recall that we are considering $k = 2$. The answer objects of data object o_1 are $A_{o_1} = \{q, o_2\}$.

Table 4. Computation of safe exit points for o_1.

Segment/Edge	p_a	A_{p_a}	$A_{n_{fi}}$	$O_{(n_{fi},p_a)}$	Safe exit point
$s(n_1, o_1)$ *in* $e(n_1, n_2)$	o_1	$A_{o_1} = \{q, o_2\}$	$A_{n_1} = \{q, o_2\}$	$\{\varnothing\}$	none
$s(o_1, n_2)$ *in* $e(n_1, n_2)$	o_1	$A_{o_1} = \{q, o_2\}$	$A_{n_2} = \{q, o_2\}$	$\{\varnothing\}$	none
$e(n_2, n_3)$	n_2	$A_{n_2} = \{q, o_2\}$	$A_{n_3} = \{q, o_6\}$	$\{q\}$	ω_1
$e(n_2, n_5)$	n_2	$A_{n_2} = \{q, o_2\}$	$A_{n_5} = \{o_2, o_3\}$	$\{o_2\}$	ω_2
$e(n_2, n_7)$	n_2	$A_{n_2} = \{q, o_2\}$	$A_{n_7} = \{q, o_7\}$	$\{q\}$	ω_3

SEA starts exploration from the active edge containing object o_1. Because $e(n_1, n_2)$ is the active sequence, the location of o_1 is the anchor point. Each of the two segments $s(n_1, o_1)$ and $s(o_1, n_2)$ within $e(n_1, n_2)$ is explored individually. As shown in Table 4, for $s(n_1, o_1)$, $p_a = o_1$, $A_{o_1} = \{q, o_2\}$, $A_{n_1} = \{q, o_2\}$, and $O_{(n_1,o_1)} = \{\varnothing\}$. By Lemma 3, $A_{n_1} \cup O_{(n_1,o_1)} = A_{o_1}$; therefore, there is no safe exit point within $s(n_1, o_1)$. Similarly, for segment $s(o_1, n_2)$ there is no safe exit point based on Lemma 3.

Therefore, the edges adjacent to n_2 are explored with $n_2 = p_a$. The edge $e(n_2, n_3)$ will be explored next. As indicated in Table 4, for $e(n_2, n_3)$, $p_a = n_2$, $A_{n_2} = \{q, o_2\}$, $A_{n_3} = \{q, o_6\}$, and $O_{(n_2,n_3)} = \{q\}$. By Lemma 2, i.e., $A_{n_3} \cup O_{(n_2,n_3)} \neq A_{n_2}$, a safe exit point exists at the edge. For each point $p \in e(n_2, n_3)$,o_f^+ will be selected from the answer objects in $A_{n_2} = \{q, o_2\}$, whereas o_n^- will be selected from the non-answer objects in $A_{n_3} \cup O_{(n_2,n_3)} - A_{n_2} = \{o_6, o_4\}$. As illustrated in Figure 8, $o_f^+ = o_2$ because for every point $\in e(n_2, n_3)$, $dist(p, o_2) > dist(p, q)$, whereas $o_n^- = o_6$ because for every point $p \in e(n_2, n_3)$, $dist(p, o_6) < dist(p, o_4)$. The safe exit point ω_1 is the midpoint between o_2 and o_6. That is, $dist(\omega_1, o_2) = dist(\omega_1, o_6)$, where $dist(\omega_1, o_2) = x + 4$ and $dist(\omega_1, o_4) = -x + 7$ for $0 < x < 5$. Consequently, $x = 1.5$, which means that the distance from n_2 to ω_1 is 1.5.

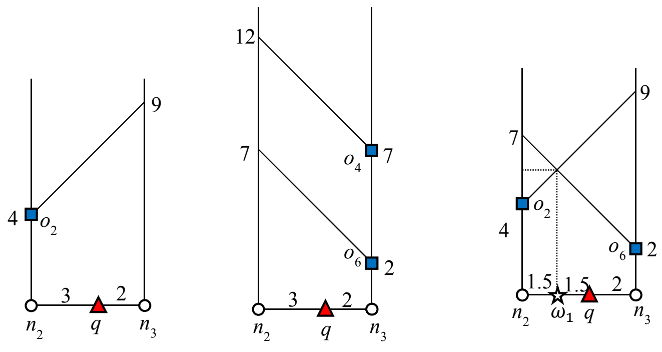


Figure 8. Determination of safe exit point ω_1.

Next, we determine a safe exit point in the edge $e(n_2, n_5)$. As shown in Table 4, for $e(n_2, n_5)$, $p_a = n_2$, $A_{n_2} = \{q, o_2\}$, $A_{n_5} = \{o_2, o_3\}$, and $O_{(n_2,n_5)} = \{o_2\}$. By Lemma 2, i.e., $A_{n_6} \cup O_{(n_2,n_6)} \neq A_{n_2}$, a safe exit point exists in the edge. For each point $p \in e(n_2, n_6)$, o_f^+ will be selected from the answer objects in $A_{n_2} = \{q, o_2\}$, whereas o_n^- will be selected from the non-answer objects in $A_{n_6} \cup O_{(n_2,n_6)} - A_{n_2} = \{o_3\}$. As indicated in Figure 9, $o_f^+ = q$ because for every point $\in e(n_2, n_6)$, $dist(p, q) > dist(p, o_2)$, because there is only one non-answer object for $e(n_2, n_6)$, therefore, $o_n^- = o_3$. The safe exit point ω_2 is the midpoint between q and o_3. That is, $dist(\omega_2, q) = dist(\omega_2, o_3)$, where $dist(\omega_2, q) = x + 3$ and $dist(\omega_2, o_3) = -x + 10$ for $0 < x < 7$. Consequently, $x = 3.5$, which means that the distance from n_2 to ω_2 is "3.5."

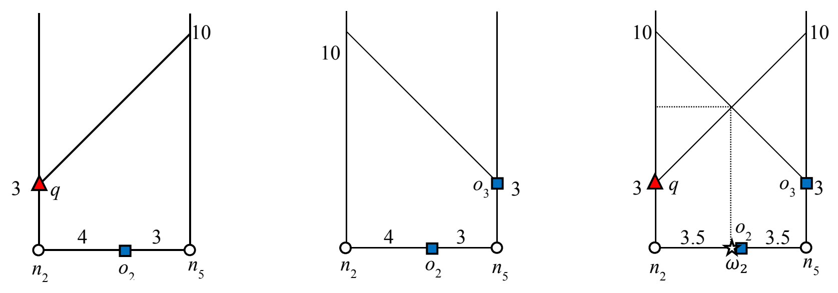


Figure 9. Determination of safe exit point ω_2.

Similarly, we compute a safe exit point in the edge $e(n_2, n_7)$. Figure 10 displays safe exit point ω_3 in the edge $e(n_2, n_7)$. Safe exit point ω_3 is the midpoint between o_2 and o_7. That is, $dist(\omega_3, o_2) = dist(\omega_3, o_7)$, where $dist(\omega_3, o_2) = x + 4$ and $dist(\omega_3, o_7) = -x + 6$ for $0 < x$. Consequently, $x = 1$, which means that the distance from n_2 to ω_3 is "1.".

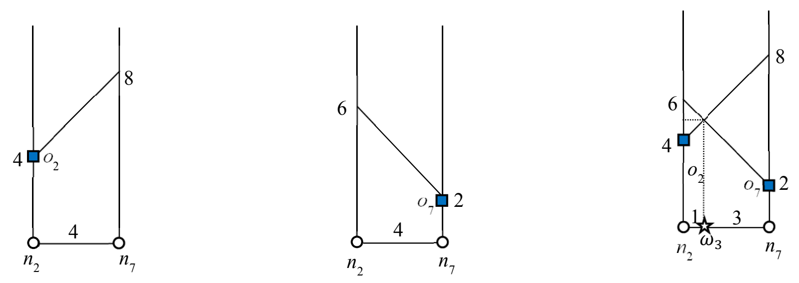


Figure 10. Determination of safe exit point ω_3.

4.2.2. Monitoring of Pruned Objects

Recall that there are three different events that can affect the R*k*NN results. Sections 4.1 and 4.2.1 present the continuous monitoring of query and UO, respectively. In this section, we discuss the monitoring of pruned objects. To avoid any additional computation, we are using the precomputed influence region of UO as the monitoring region for pruned objects. By definition, the influence region of an answer object is valid until q is a *k*NN of the answer object, whereas in the case of a non-answer object, the influence region remains valid until q is not a *k*NN of the non-answer object. Therefore, it is necessary to monitor the objects that are entering or leaving the influence region because the movement of objects in and out the influence region can change the results. Therefore, we only start monitoring the movement of a pruned object once it enters or leaves the influence region.

Let o be a pruned object located outside the influence region of any UO. The monitoring of object o will not trigger unless it enters the influence region of any UO. Once it enters the influence region, the monitoring will start and the results set will only change when the NN of any UO changes. If the movement of a pruned object does not change the NN set of any UO, then the result set remains valid and it is not necessary to call the determining UO phase again. The determining UO phase will only be called when the NN set changes.

4.3. Extensions

4.3.1. R*k*NN Queries in Directed Road Networks

Previous sections discussed the monitoring of R*k*NN queries in a road network that are represented by bidirectional graphs. In this section, we identify the modifications required to extend the proposed technique to a directed road network where each road segment has a particular orientation.

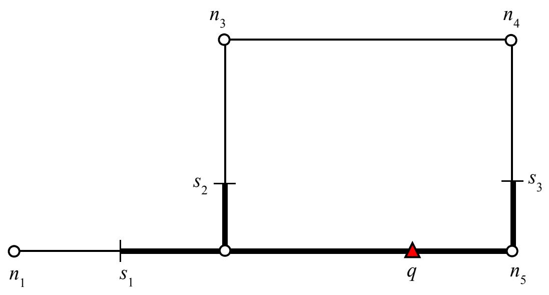
First, we discuss the modifications required for the computation of the safe exit points of q. The majority of the changes are required to be implemented in Phase 1 (i.e., determining useful objects) and are automatically addressed by redefining $dist(p_1, p_2)$. Given two points p_1 and p_2, the shortest path distance $dist(p_1, p_2)$ is the minimum distance *from* p_1 *to* p_2. By applying this modified definition, the categorizing of answer and non-answer objects remains unchanged. The definition of the closed node will be same. Consequently, Lemma 1 remains valid. Algorithm 2, can be used in a similar manner. The only difference is that to determine the UO, we explore the network in the reverse direction, i.e., an edge $e(n_1, n_2)$ is considered the adjacent edge of n_1 if and only if the edge is bidirectional or the direction of the edge is from n_2 to n_1.

Phase 2 is the same, except to determine o_{k+1} or o_k we expand the road network in the direction of the edges. Further, the computation of the influence region of o^+ is the same, except the distance $dist(o^+, o_{k+1})$ is the shortest path from o^+ to o_{k+1} instead of the shortest path from o^+ to o_{k+1}. Similarly, the computation of the influence region of o^- is the same, except the distance $dist(o^-, o_k)$ is the shortest path *from* o^- *to* o_k instead of the shortest path between o^- and o_k. Phase 3 is the same as for a directed road network. Finally, we discuss the modifications in the computation of the safe exit points of the UO. SEA functions in a similar fashion except it only explores the network in the direction of the edges. The remainder of the methodology, i.e., Lemma 2, Lemma 3, and the definition of Ω, remains unchanged.

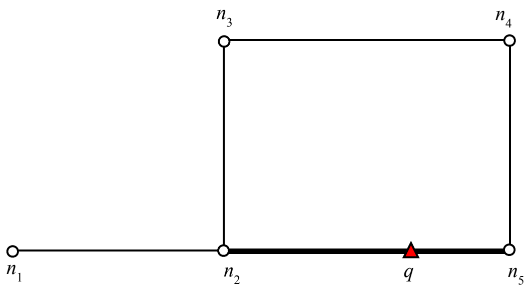
4.3.2. R*k*NN Queries in Dynamic Road Networks

In this section, we extend CORE-X to dynamic road networks where the network distance changes depending on the traffic conditions. The query result or safe region of query or data object may frequently be nullified by updating the weights of some edges, even though the query object q or data object o remains within their respective safe region. Thus, we introduce a compressed safe region to reduce the frequency of evaluations of the safe region in a dynamic road network. In this section, the safe region in a static road network is denoted as the original safe region, SR_{or}, whereas the safe region in a dynamic road network is denoted as the compressed safe region, SR_{co}, which is clearly a subset of SR_{or}. To simplify the explanation, we study the impact of a dynamic road network with respect to q only.

Unlike the original safe region, the compressed safe region is determined within only the active sequence. Figure 11 illustrates the difference between SR_{or} and SR_{co}, where n_2n_3 is the active edge. The bold line in Figure 11a indicates that for SR_{or}, s_1, s_2, and s_3 are the safe exit points. In dynamic road networks, the safe region is compressed to active edge n_2n_5 as indicated in bold lines in Figure 11b.



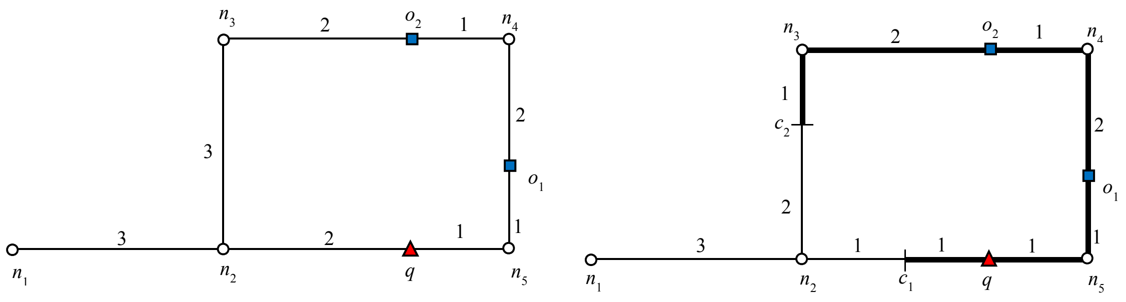
(**a**) Original safe region SR_{or}



(**b**) Compressed safe region SR_{co}

Figure 11. Difference between original safe region and compressed safe region SR_{or} and compressed safe region SR_{co}. (**a**) Original safe region SR_{or}; (**b**) Compressed safe region SR_{co}.

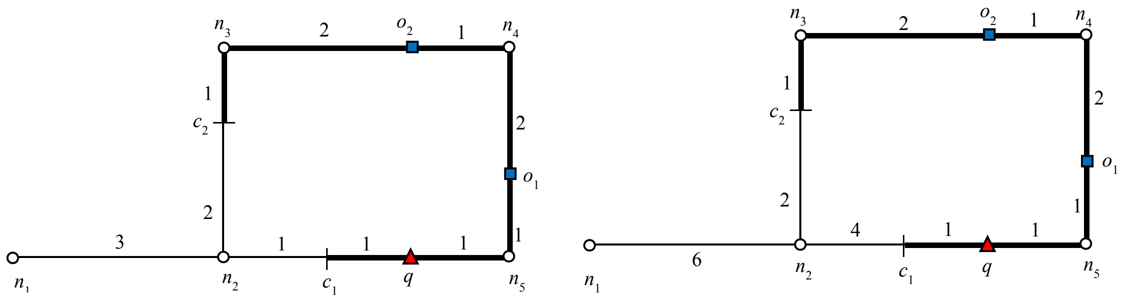
Next, we introduce a critical region to monitor the validity of the safe region effectively when the weight of an edge is changed. The critical region CR is defined as $CR = IR^+ \cup IR^-$, where IR^+ and IR^- are the influence region of answer and non-answer objects, respectively. By definition, the critical region contains all the points p_{cr} such that $p_{cr} \in \{IR^+, IR^-\}$. As illustrated in Figure 12a, there are two data objects o_1 and o_2 and a query object q. For $k = 1$, it is obvious that o_1 is an answer object because $q = NN(o_1)$ and o_2 is a non-answer object because $o_1 = NN(o_2)$. Thus, the critical region contains all the points $p_{cr} \in \{IR(o_1), IR(o_2)\}$, denoted as the bold line in Figure 12b with c_1 and c_2 as boundary points. The safe region of q may be affected when the weight of an edge associated with the critical region changes. Therefore, the critical regions of q are stored and indexed to promptly verify whether a change affects the safe region. If the influence region of any UO is updated, the critical region of q is also updated accordingly. Clearly, changes to edges that are not associated with the critical region can be safely ignored.



(**a**) Road Network Example (**b**) Critical Region of q

Figure 12. Difference between original safe region SR_{or} and compressed safe region SR_{co}. (**a**) Road Network Example; (**b**) Critical Region of q.

Figure 13a indicates the critical region by bold lines at time t_i. Suppose, because of heavy traffic conditions, the weights of two edges n_1n_2 and n_2n_5 are changed at time t_j from "3" to "6" as displayed in Figure 13b. The update to edge n_2n_5, which is associated with a critical region, may affect the safe region. However, the update to edge n_1n_2, which is not part of a critical region, does not affect the safe region and thus can be safely ignored.



(**a**) Critical Region of q at t_i (**b**) Updates to n_1n_2 and n_2n_3 at t_j

Figure 13. Updating the weights of two edges n_1n_2 and n_2n_3 where $t_i < t_j$. (*a*) Critical Region of q at t_i; (*b*) Updates to n_1n_2 and n_2n_3 at t_j.

4.3.3. Bichromatic R*k*NN Queries

Unlike the *monochromatic* case where all objects are of the same type, in *bichromatic* RNN, we distinguish between two types of objects O and S. For a query object of type S, the objective is to determine data objects of type O for which the query point is their closest object in the set S. With this fundamental difference between MR*k*NN and BR*k*NN, CORE-X maintains the same methodology to manage both types of query. This section presents the changes required in the proposed technique to process BR*k*NN queries.

1. In determining the UO phase, only objects of type S are considered as blocking objects to prune the network. Therefore, a closed node can be redefined as a node n for which there exists an object

$s \in S$ such that $dist(n, s) < dist(n, q)$. The remainder of Lemma 1 remains unchanged. The objects of type O that lie in the unpruned network are considered as UO objects, whereas the objects of type S in the unpruned network are called active objects.

2. Similar to MR*k*NN, UO objects are categorized into answer and non-answer objects. Answer objects for BRkNN queries can be defined as an object o for which $dist(o, q) < dist(o, s_{k+1})$ where s_{k+1} is the (k +1)th NN object of o in dataset S. Similarly, a non-answer object is defined as an object o such that $dist(o, q) > dist(o, s_k)$, where s_k is the kth NN object of o in dataset S.
3. The influence region of an answer object is defined as $IR^+ = \{p|dist(o^+, p) \leq dist(o^+, s_{k+1})\}$, where $dist(o^+, s_{k+1})$ is the distance between the answer object and (k+1)th NN in set S. Similarly, the influence region of a non-answer object is defined as $IR^- = \{p|dist(o^-, p) \leq dist(o^-, s_k)\}$, where $dist(o^-, s_k)$ is the distance between the non-answer object and kth nearest object in set S. Phase 3 remains unchanged for BR*k*NN.
4. The computation of safe exit points for the UO remains the same. The only difference is for BR*k*NN, the farthest answer object and nearest non-answer object belong to set S. Therefore, the safe exit point is the center point between s_f^+ and s_n^-.
5. The safe region of an active object can be computed in a similar fashion as q because both belong to same data type.

4.4. Analysis of Time and Space Complexities

In this section, we analyze both time and space complexities of the CORE-X algorithm. Recall that we are studying R*k*NN queries in an undirected road network that is represented as $G = (N, E, W)$. Let N_k^p and E_k^p be the sets of nodes and edges of point p within the kNN objects, respectively.

First, we analyze the time complexity of CORE-X. CORE-X retrieves a set of UO objects by traversing the road network from query location q, which has a time complexity of $T\left(2\left|E_k^q\right| + \left|N_k^q\right|logN_k^q\right)$ because an edge is visited at most twice, first to determine the answer objects and then to determine the useful non-answer objects. Here, time complexity of $T\left(\left|N_k^q\right|logN_k^q\right)$ refers to the time complexity of the shortest path algorithm from a query point to kRNN objects [29]. Next, CORE-X computes the safe exit points using the retrieved UO objects. For each data object $o \in UO$, the influence region of data object o is searched to discover safe exit points, which has a time complexity of $T\left(\left|E_k^q\right| + \left|N_k^q\right|logN_k^q\right)$. Thus, the time complexity of CORE-X is $T\left((2\left|E_k^q\right| + \left|N_k^q\right|logN_k^q) + |UO|(\left|E_k^q\right| + \left|N_k^q\right|logN_k^q)\right)$. The overall time complexity of CORE-X can be improved to $T\left((2\left|E_k^q\right| + \left|N_k^q\right|) + |UO|(\left|E_k^q\right| + \left|N_k^q\right|)\right)$ using the preprocessed ordering of nodes in static road networks. However, in the case of dynamic road networks, the preprocessing technique is naturally insignificant because updating the weight of some edges often invalidates the preprocessed information. The search space of CORE-X becomes a region within a distance of k RNNs of q and (k+1)NNs of the data objects because it is necessary to explore the (k+1)th NN to compute the influence region of an answer object. Finally, the space complexity of CORE-X is $\left|O_k^q\right| + \left|O_{k+1}^{UO}\right|$.

5. Performance Evaluation

In this section, we evaluate the performance of the proposed algorithm (CORE-X) through simulation experiments. We describe our experimental settings in Section 5.1 and present our experimental results for static and dynamic road networks in Sections 5.2 and 5.3, respectively.

5.1. Experimental Settings

Our experiments were performed using a real road network [30] for San Joaquin County, California, USA, which contains 18,263 nodes and 23,874 edges. The road network contains a set of queries and a set of data objects that can move randomly in the network. We simulate moving objects (both query and data objects) using the network-based moving-object generator [31]. In each experiment, we evaluated the performance by varying a single parameter within the range indicated; all other parameters were set to the default values indicated in bold. All queries were monitored continuously for 600 timestamps. Table 5 lists the default parameters used in our experiments.

Table 5. Experimental parameter settings.

Parameter	Range
Number of data objects (N_{Data})	10, 50, **100**, 150, 200, 250 ($\times$1000)
Number of queries (N_{qry})	400, **800**, 1200, 1600, 2000
Number of requested RNNs (k)	5, 10, **15**, 20, 25
Speed of objects (V_{obj})	20, 40, **60**, 80, 100 (km/h)
Percentage of moving objects (R_{obj})	20, 40, **60**, 80, 100
Percentage of updated edges (R_{upd})	0, 5, **10**, 25, 30

We evaluated the performance of CORE-X using the following measures: (1) total amount of server CPU time per timestamp, which indicates the query processing time and (2) total communication cost as the total number of points (i.e., the location updates sent by data and query objects, and the query results and safe exit points returned by the server) transferred between clients and the server. The total query processing time is dominated by the computations (i.e., determining UO, computation influence region and safe exit points) performed at the server end, therefore we only measure the CPU time on the server. The battery power and wireless bandwidth consumption typically increase with the amount of data transferred between objects(clients) and servers, thus we use the amount of transferred data as a metric to evaluate the communication cost. For static road networks, we compared CORE-X with the state-of-the-art algorithm SAC [8] and a baseline algorithm. SAC continuously monitors R*k*NN queries by assigning a safe region to query and data objects, whereas baseline algorithm recomputes the results at each timestamp. For dynamic road networks, we compared the proposed algorithm with only the baseline because SAC does not address dynamic road networks. All algorithms were implemented in Java and executed on a desktop PC, 2.80 GHz, Intel Core i5 with 4 GB memory.

5.2. Experimental Results for Static Road Networks

Figure 14a presents the performance of baseline, SAC, and CORE-X with respect to the number of objects. Observe that all the three algorithms are sensitive to the increase in the number of objects because the algorithm must address updates from a large number of objects. The performance of SAC degrades with a large number of data objects because filtering and verification of more data objects are required. The query processing time of CORE-X increases with the increase in data objects mainly because the influence region and ω for a large number of objects must be calculated. However, CORE-X scales better than baseline and SAC.

Figure 14b displays the comparison of computation cost of CORE-X, SAC, and baseline with different values of N_{qry}. Experimental results reveal that the computation time of all algorithms increases as the number of queries increases. However, CORE-X and SAC consumed considerably less CPU time than baseline because the query results are valid as long as the query and data objects remain in their respective safe region. Figure 14c, studies the effect of the proportion of query and data objects that are moving. Performance of all algorithms degrades as the percentage of moving objects increases. The computation time of CORE-X increases mainly because with data mobility, the safe region expires more frequently and the algorithm must recompute the results frequently.

Figure 14d demonstrates the influence of the speed of data objects and query on the performance of CORE-X, SAC, and baseline algorithms. The experimental results reveal that the baseline and SAC algorithms incur constant computation costs. The performance of SAC is not significantly affected by the speed because candidate objects must be verified at each timestamp, regardless of their speed. Conversely, the performance of CORE-X gradually decreases as the speed of the objects increases because objects leave their respective safe regions more frequently. Figure 14e shows the query processing time of CORE-X, SAC, and baseline as a function of k number of requested RNNs. Experimental results indicate that the computation time of all algorithms increases with an increase of the k value. This is expected because with an increase in k, more data objects are required to be explored and verified. However, CORE-X outperforms SAC and baseline.

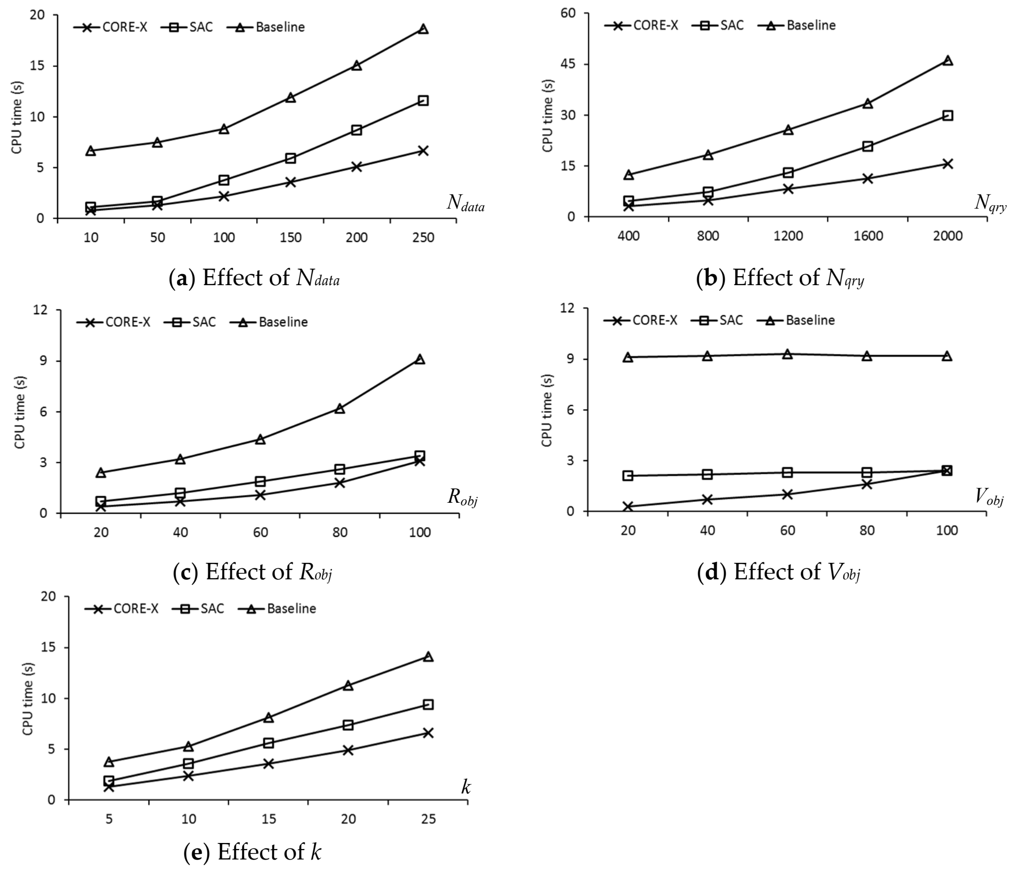
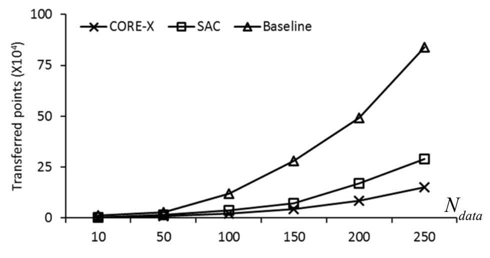


Figure 14. Comparison of computational cost. (**a**) Effect of N_{data}; (**b**) Effect of N_{qry}; (**c**) Effect of R_{obj}; (**d**) Effect of V_{obj}; (**e**) Effect of k.

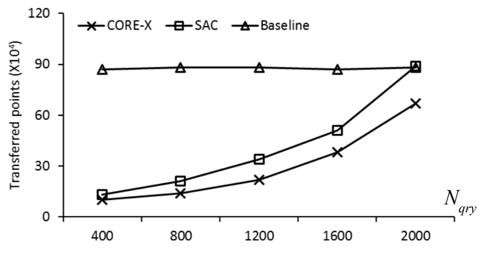
In Figure 15a, we study the effect of the number of objects on the communication cost. This figure illustrates that the number of messages sent by all algorithms tended to increase as the number of objects increased. It is clear, however, that the safe region-based algorithms SAC and CORE-X significantly outperformed baseline. The proposed algorithm demonstrates superior performance compared to SAC because the client-server communication is not required when the query and data objects remain within the safe exit points, whereas in SAC, candidate objects are required to report their location to the server for verification whenever they change their location.

Figure 15b illustrates the effect of the number of queries on the communication cost. Baseline incurs constant communication cost, regardless of different values of N_{qry}. Conversely, the communication cost of CORE-X and SAC increases with an increase in the number of queries. The communication cost of SAC increases mainly because more data objects are required to be verified at each timestamp, which increases the number of server-initiated updates. CORE-X only verifies the useful object when it leaves its safe region, which significantly reduces the communication cost.

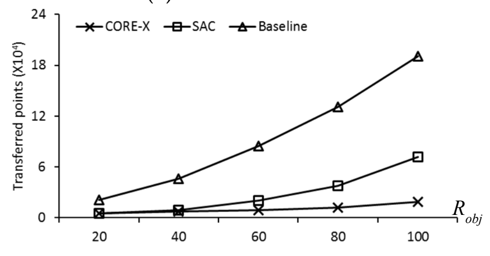
Figure 15c presents the performance trends of CORE-X, SAC, and baseline as a function of query and data object mobility. As expected, baseline performs most poorly because it must update each data object at every timestamp. The communication cost of SAC is higher than CORE-X because of its expensive verification method. Figure 15d shows the total communication cost of CORE-X, SAC, and baseline with respect to speed of data and query objects. This figure indicates similar trends to Figure 14d. SAC incurs constant communication cost because the server-initiated request to verify the candidate objects does not depend on the speed. For CORE-X, the objects reach a safe exit point earlier when the speed is increased, which increases the communication cost. As indicated in Figure 15e, the communication costs of CORE-X, SAC, and the baseline algorithm increase with *k*. Both CORE-X and SAC perform better than the baseline method. However, CORE-X significantly outperforms SAC for all cases because of low verification cost.



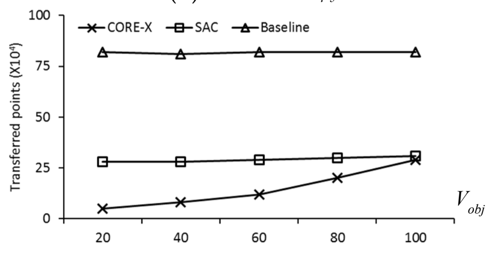
(**a**) Effect of N_{data}



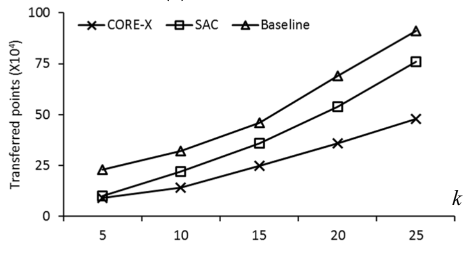
(**b**) Effect of N_{qry}



(**c**) Effect of R_{obj}



(**d**) Effect of V_{obj}



(**e**) Effect of *k*

Figure 15. Comparison of the communication cost. (**a**) Effect of N_{data}; (**b**) Effect of N_{qry}; (**c**) Effect of R_{obj}; (**d**) Effect of V_{obj}; (**e**) Effect of *k*.

5.3. Experimental Results for Dynamic Road Networks

Figure 16 demonstrates a comparison of the query processing time for a dynamic road network where the R_{upd} (%) values of all edges change their weight at each timestamp. The updated edges are selected randomly, irrespective of the locations of data and query objects. The length of an updated edge is selected randomly from 0.1 to 10 times the original length.

Figure 16a presents the query processing time as a function of the percentage of updated edges (R_{upd}) per timestamp. Here, $R_{upd} = 0$ indicates a static road network. Naturally, the query processing time of baseline is almost constant regardless of the value of R_{upd} because query objects in baseline issue R*k*NN queries at each timestamp. The query processing time of CORE-X increases with R_{upd}; this is mainly because the safe region of a query and data object must be updated if an edge is associated with the critical region of the data or query object. Observe that for $R_{upd} \leq 25$, CORE-X performs considerably better than baseline, whereas for $R_{upd} = 50$, the query processing time of CORE-X is greater than baseline.

Figure 16b illustrates the effect of the number of data objects on the query processing time. The computation time for both algorithms typically increases with N_{data}. In this case, CORE-X significantly outperforms baseline. In Figure 16c, we study the effect of N_{qry} on the computation cost. Experimental results reveal that both algorithms are sensitive to an increase in the number of query objects. The query processing time of CORE-X increases mainly because the computation of the critical region and safe region for more objects is required. Figure 16d indicates the trend that computation cost of both algorithms increases with an increase in query and data object mobility. However, CORE-X scales significantly better than baseline.

Figure 16e is the query processing time of CORE-X and baseline as a function of the speed of the data and query objects. Baseline has a nearly stable query processing time. However, the computation cost of CORE-X increases with V_{obj} because, as the objects move faster, they reach their respective safe exit points earlier, resulting in more frequent updates of the query results. Figure 16f demonstrates the influence of the k number of requested RNNs on the performance of both algorithms. As expected, the computation of both algorithms increases with an increase of the k value. With CORE-X, the critical region of more data objects needing to be computed increases the size of the critical region. Consequently, the number of edges associated with the critical region also increases, which elevates the query processing time.

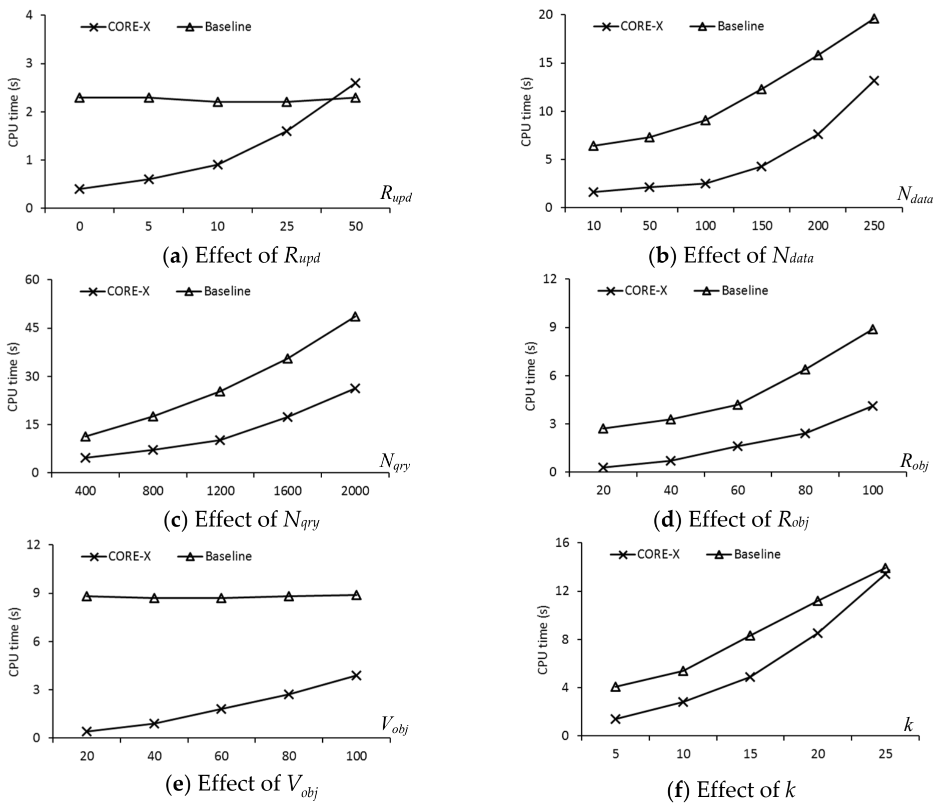
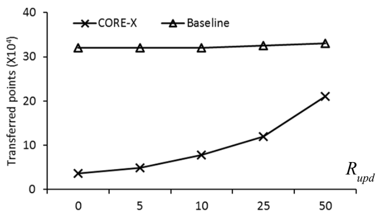


Figure 16. Comparison of the computational cost. (**a**) Effect of R_{upd}; (**b**) Effect of N_{data}; (**c**) Effect of N_{qry}; (**d**) Effect of R_{obj}; *(e)* Effect of V_{obj}; (**f**) Effect of k.

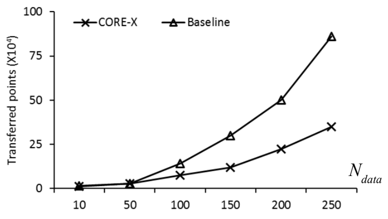
Figure 17 presents a comparison of communication cost of CORE-X and baseline using the same conditions as those in Figure 16. Figure 17a illustrates the effect of R_{upd} on the communication cost of both algorithms. The performance of baseline is not significantly affected by R_{upd}. However, the communication cost of CORE-X increases with R_{upd} because the critical regions of data and query objects are updated more frequently as R_{upd} increases. Consequently, the safe regions of the query and data objects are reevaluated.

In Figure 17b, we study the influence of the cardinality of the data objects on the communication costs. Experimental results indicate that CORE-X performs better than baseline because in CORE-X, server-client communication is only required when the query and data objects leave their safe regions or when the critical regions of the data and query objects are updated. Figure 17c shows the communication cost of CORE-X and baseline with respect to the number of query objects. Observe that the communication cost of baseline does not depend on the number of query objects because each data object reports its location whenever it changes its location. On the other hand, the communication cost of CORE-X increases because for a large number of query objects, more data objects are required to be verified.

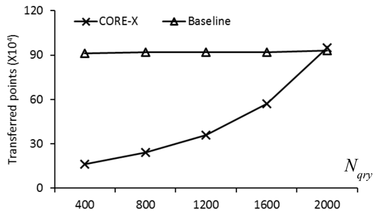
Figure 17d exhibits the effect of query and data object mobility on the communication costs. Clearly, the number of transferred points increases with an increase in R_{obj}. However, CORE-X consistently provides improved performance compared to baseline. Figure 17e illustrates the effect of the speed of the data and query objects on communications costs. In CORE-X, the number of transferred points increases with V_{obj} for the same reason as discussed earlier in the case of Figure 15d. As indicated in Figure 17f, the communication costs of CORE-X and the baseline algorithm increases with k. This is mainly because the number of data objects that require verification and monitoring increases with k.



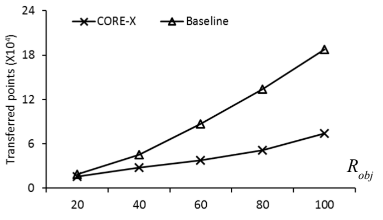
(**a**) Effect of R_{upd}



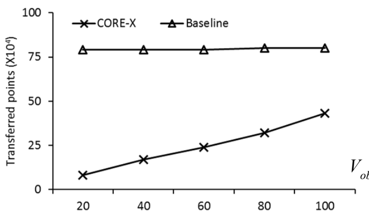
(**b**) Effect of N_{data}



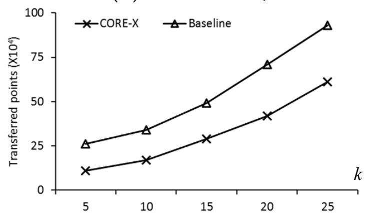
(**c**) Effect of N_{qry}



(**d**) Effect of R_{obj}



(**e**) Effect of V_{obj}



(**f**) Effect of k

Figure 17. Comparison of the communication cost. (**a**) Effect of R_{upd}; (**b**) Effect of N_{data}; (**c**) Effect of N_{qry}; (**d**) Effect of R_{obj}; (**e**) Effect of $V_{obj\ j}$; (**f**) Effect of k.

6. Conclusions

We proposed a new algorithm called CORE-X for the efficient processing of continuous reverse k nearest neigbor (RkNN) queries in road networks where both query and data objects are moving. The proposed approach is based on safe exit points, which can significantly improve not only the computation cost but also the communication cost between server and query object. Moreover, we presented pruning techniques and influence region-based monitoring to avoid processing of irrelevant data objects. CORE-X can effectively construct a safe region in a dynamic road network by introducing a compressed safe region and critical region. The results of experiments conducted using real datasets confirm that the proposed algorithm significantly reduces the computation cost and the communication cost compared to a baseline and state-of-the art algorithm (SAC).

There are several promising directions of future research. First, RNN queries can be studied for privacy-aware systems to ensure the location privacy of a query object from an attacker. Further, this study can be extended for uncertain data objects, which may not have exact locations. It is important to determine their approximate locations and develop accuracy bounds on the query results.

Acknowledgments: We thank anonymous reviewers for their valuable comments and suggestions. Muhammad Attique, Rize Jin and Tae-Sun Chung were supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2013R1A1A2A10012956 and NRF-2012R1A1A2043422). Hyung-Ju Cho was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIP) (No. NRF-2016R1A2B4009793). Finally, this work was partially supported by Leaders Industry-University Cooperation Project.

Author Contributions: All authors significantly contributed to the manuscript. Muhammad Attique initiated the idea, implemented the experiments, and wrote the manuscript. Muhammad Attique and Hyung-Ju Cho designed the solution and experiments. Rize Jin critically reviewed the paper and revised the manuscript. Tae-Sun Chung reviewed the manuscript and supervised the research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ciuonzo, D.; Buonanno, A.; D'Urso, M.; Palmieri, F. Distributed classification of multiple moving targets with binary wireless sensor network. In Proceedings of the 14th International Conference on Information Fusion (FUSION), Chicago, IL, USA, 5–8 July 2011; pp. 1–8.
2. Buonanno, A.; D'Urso, M.; Prisco, G.; Felaco, M.; Meliadò, E.F.; Mattei, M.; Palmieri, F.; Ciuonzo, D. Mobile sensor networks based on autonomous platforms for homeland security. In Proceedings of the Tyrrhenian Workshop on Advances in Radar and Remote Sensing (TyWRRS), Naples, Italy, 12–14 September 2012; pp. 80–84.
3. Cho, H.; Kwon, S.; Chung, T. A safe exit algorithm for continuous nearest neighbor monitoring in road networks. *Mob. Inf. Syst.* **2013**, *9*, 37–53. [CrossRef]
4. Cho, H.; Ryu, K.; Chung, T. An efficient algorithm for computing safe exit points of moving range queries in directed road networks. *Inf. Syst.* **2014**, *41*, 1–19. [CrossRef]
5. Wang, H.; Zimmermann, R. Processing of continuous location-based range queries on moving objects in road networks. *IEEE Trans. Knowl. Data Eng.* **2011**, *23*, 1065–1078. [CrossRef]
6. Yung, D.; Yiu, M.; Lo, E. A safe-exit approach for efficient network-based moving range queries. *Data Knowl. Eng.* **2012**, *72*, 126–147. [CrossRef]
7. Zhang, J.; Zhu, M.; Papadias, D.; Tao, Y.; Lee, D. Location-based spatial queries. In Proceedings of the 2003 ACM SIGMOD International Conference on Management of data, San Diego, CA, USA, 10–12 June 2003; pp. 443–454.
8. Cheema, M.; Lin, M.X.; Zhang, Y.; Zhang, W.; Li, X. Continuous reverse k nearest neighbors queries in Euclidean space and in spatial networks. *VLDB J.* **2012**, *21*, 69–95. [CrossRef]
9. Korn, F.; Muthukrishnan, S. Influence sets based on reverse nearest neighbor queries. In Proceedings of the 2000 ACM SIGMOD international conference on Management of data, Dallas, TX, USA, 16–18 May 2000; pp. 201–212.
10. Stanoi, I.; Agrawal, S.; Abbadi, A. Reverse nearest neighbor queries for dynamic databases. In Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Dallas, TX, USA, 14 May 2000; pp. 44–53.
11. Tao, Y.; Papadias, D.; Lian, X. Reverse *k*NN search in arbitrary dimensionality. In Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Japan, 31 August–3 September 2004; pp. 744–755.
12. Kolahdouzan, M.; Shahabi, C. Voronoi-based *k* nearest neighbor search for spatial network databases. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Japan, 31 August–3 September 2004*; ACM: New York, NY, USA; pp. 840–851.

13. Gao, Y.; Zheng, B.; Chen, G.; Lee, W.; Lee, K.; Li, Q. Visible reverse *k*-nearest neighbor queries. In Proceedings of the IEEE 25th International Conference on Data Engineering, Shanghai, China, 29 March–2 April 2009; pp. 1314–1327.
14. Li, G.; Fan, P.; Li, Y.; Du, J. An efficient technique for continuous k-nearest neighbor query processing on moving objects in a road network. In Proceedings of the IEEE 10th International Conference on Computer and Information Technology (CIT), Bradford, UK, 29 June–1 July 2010; pp. 627–634.
15. Song, Z.; Roussopoulos, N. K-nearest neighbor search for moving query point. In Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD), Redondo Beach, CA, USA, 12–15 July 2001; pp. 79–96.
16. Sun, H.; Jiang, C.; Liu, J.; Sun, L. Continuous reverse nearest neighbor queries on moving objects in road networks. In Proceedings of the Ninth International Conference on Web-Age Information Management (WAIM), Zhangjiajie, Hunan, China, 20–22 July 2008.
17. Yiu, M.; Mamoulis, N.; Papadias, D.; Tao, Y. Reverse nearest neighbor in large graphs. *IEEE Trans. Knowl. Data Eng.* **2006**, *18*, 540–553.
18. Benetis, R.; Jensen, C.; Karciauskas, G.; Saltenis, S. Nearest neighbor and reverse nearest neighbor queries for moving objects. In Proceedings of the International Database Engineering and Applications Symposium, Edmonton, AB, Canada, 17–19 July 2002; pp. 44–53.
19. Xia, T.; Zhang, D. Continuous reverse nearest neighbor monitoring. In Proceedings of the 22nd International Conference on Data Engineering (ICDE), Atlanta, GA, USA, 3–7 April 2006; p. 77.
20. Kang, J.; Mokbel, M.; Shekhar, S.; Xia, T.; Zhang, D. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In Proceedings of the IEEE 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey, 16–20 April 2007; pp. 806–815.
21. Wu, W.; Yang, F.; Chan, C.; Tan, K. Continuous reverse *k*-nearest-neighbor monitoring. In Proceedings of the Ninth International Conference on Mobile Data Management (MDM), Beijing, China, 27–30 April 2008; pp. 132–139.
22. Cheema, M.; Lin, X.; Zhang, W.; Zhang, Y. Influence zone: Efficiently processing reverse *k* nearest neighbors queries. In Proceedings of the IEEE 27th International Conference on Data Engineering (ICDE), Hannover, Germany, 11–16 April 2011; pp. 577–588.
23. Safar, M.; Ebrahimi, D.; Taniar, D. Voronoi-based reverse nearest neighbor query processing on spatial networks. *Multimed. Syst.* **2009**, *15*, 295–308. [CrossRef]
24. Li, G.; Li, Y.; Li, J.; Shu, L.; Yang, F. Continuous reverse *k* nearest neighbor monitoring on moving objects in road networks. *Inf. Syst.* **2010**, *35*, 860–883.
25. Gotoh, Y. A simple routing method for reverse *k*-nearest neighbor queries in spatial networks. In Proceedings of the IEEE 17th International Conference on Network-Based Information Systems, Salerno, Italy, 10–12 September 2014; pp. 615–620.
26. Wang, S.; Cheema, M.; Lin, X. Efficiently monitoring reverse *k*-nearest neighbors in spatial networks. *Comput. J.* **2015**, *58*, 40–56. [CrossRef]
27. Attique, M.; Hailu, Y.; Ayele, S.; Cho, H.; Chung, T. A safe exit approach for continuous monitoring of reverse *k* nearest neighbors in road networks. *Int. Arab J. Inf. Tech.* **2015**, *12*, 540–549.
28. Attique, M.; Cho, H.; Chung, T. CORE: Continuous monitoring of reverse k nearest neighbors on moving objects in road networks. *Stud. Comput. Intell.* **2016**, *2015*, 109–124.
29. Cormen, T.; Leiserson, C.; Rivest, R.; Stein, C. *Introduction to Algorithms*, 3rd ed.; MIT Press and McGraw-Hill: Cambridge, MA, USA, 2009.
30. Real Datasets for Spatial Databases. Available online: https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm (accessed on 2 April 2016).
31. Brinkhoff, T. A framework for generating network-based moving objects. *GeoInformatica* **2002**, *6*, 153–180. [CrossRef]