A Pruning Algorithm for Reverse Nearest Neighbors in Directed Road Networks

Rizwan Qamar¹, Muhammad Attique², Tae-Sun Chung³

¹ Computer Engineering, Ajou University, Paldal Hall 913-2, Suwon, Gyeonggi-do 443-749, South Korea E-mail: Rizwan.Qamar@outlook.com
² Computer Engineering, Ajou University, Paldal Hall 913-2, Suwon, Gyeonggi-do 443-749, South Korea E-mail: Attiq85@gmail.com
³ Computer Engineering, Ajou University, Paldal Hall 913-2, Suwon, Gyeonggi-do 443-749, South Korea E-mail: tschung@ajou.ac.kr

Abstract

In this paper, we studied the problem of continuous reverse k nearest neighbors (RkNN) in directed road network, where a road segment can have a particular orientation. A RNN query returns a set of data objects that take query point as their nearest neighbor. Although, much research has been done for RNN in Euclidean and undirected network space, very less attention has been paid to directed road network, where network distances are not symmetric. In this paper, we provided pruning rules which are used to minimize the network expansion while searching for the result of a RNN query. Based on these pruning rules we provide an algorithm named SWIFT for answering RNN queries in continuous directed road network.

Keywords: reverse nearest neighbors, spatial query, directed road network, continuous.

1. Introduction

Spatial databases offer large number of services such as nearest neighbor, resource allocation, and preferential search etc. These services have not only changed peoples daily life but also scientific research. For example, people now rely on location-based services to plan and manage their trips. This new demand for location-aware services has resulted in development of efficient algorithms and many novel query types for spatial databases. One of them is reverse nearest neighbor (RkNN). While a lot of attention has been given to this problem because of its practical applicability [1–3], most of it exclusively focuses on Euclidian space or undirected road network. In this paper, we study safe region of a reverse nearest neighbor query for a moving query and static data objects in a directed road network (i.e.,
A safe region is a region in which result of the query doesn’t change when the query object moves within safe region.

Consider a query point \( q \) and a set \( O \) of points of interest (POIs) e.g. offices, universities, schools etc. We use \( d(q, o) \) to denote network distance from query object \( q \) to data object \( o \). Given a query point \( q \), RkNN query returns a set of data points \( o \in O \) such that \( q \) is one of there \( k \) nearest neighbor i.e. \( RkNN(q) = o \in d(o, q) \leq d(o, o_k) \) where \( o_k \) is the \( k^{th} \) nearest neighbor of data point \( o \). Many real-life scenarios exist to illustrate the importance of continuous reverse nearest neighbor queries. Consider a first person shooting game in which the goal of each player is to shoot the other player. Naturally, all players try to avoid getting shot and for this they continuously monitor their own reverse nearest neighbor as chances of getting shot from the reverse nearest neighbor are highest. Fig. 1 illustrates such game, where \( p_1 \) is closest to \( q \) but RNN of \( q \) is \( p_2 \) as nearest neighbor of \( p_1 \) is \( p_3 \).

RkNN has received a lot of attention [4–6] from the research community for applications like emergency response team and taxi providing services. In general, RkNN query can further be classified into two groups monochromatic and biochromatic. Our example above is of monochromatic RkNN query where all objects belong to the same set of objects. Consider the example of taxi and customers where they belong to the set of taxis and customers respectively.

In general, the main problem in continuous reverse nearest neighbor query is how to maintain the freshness of the query result, as the query object is moving freely and arbitrarily. A naïve technique for finding RkNN of a moving query object \( q \) is to increase the frequency of updates. However, just increasing the frequency of queries doesn’t not guarantee freshness of result and the it may become invalid in between two timestamps. Moreover this not only increases the load on server but also increases the use of communication channel between client and the server. The contributions of this paper are as following:

- We present an algorithm for calculation of moving RkNN in a directed road networks.
- We provide a pruning rule that minimizes the network expansion for finding RkNN in directed road networks.
- We discuss why RkNN algorithm for undirected road is not applicable to directed road networks.
- We conduct experiments to study the effects of various parameters and show superiority of SWIFT over naïve algorithm.

The remainder of this paper is structured as follows. Section 2 surveys related work and limitation of undirected algorithms. Section 3 explains terms and definitions used in this paper as well as gives explanation about the problem and some pruning rules. Section 4 explains the working of SWIFT algorithm. Section 5 discusses SWIFT in continuous road network. Section 6 discusses the experimental results. Section 7 gives concluding remarks and future way through.

2. Related Work

Section 2.1 surveys RkNN in spatial databases and Section 2.2 discusses why undirected algorithms are not applicable to directed road networks.

2.1. Reverse Nearest Neighbor in Euclidean and Road Networks

RkNN was first introduced by Korn et al. [1] where they used pre-computations to answer the RNN for a query object \( q \). Drawback of this technique was that they were only able to answer RkNN query for a fixed value of \( k \). Stanoi et al. [7] proposed an algorithm that did not require preprocessing. They proposed partitioning algorithm that divides the whole
space into six equal regions of $60^\circ$. It can be verified that the possible RNN of $q$ can only be the nearest point to $q$ found in each of the six regions. This proves that in 2D space, there can be at most six RNNs for a query point $q$.

Sun et al. [3] presented a continuous monitoring in bichromatic RkNN queries. They used multiway tree with each query to assign it a monitoring region and only updates in the monitoring region affects the result. However, it is only applicable to bichromatic queries and can only answer queries when $k = 1$.

Cheema et al. [5] presented continuous reverse nearest neighbor queries for both Euclidian and spatial road network and suggested pruning rules for road networks. Similarly, Man L Yiu. [4] proposed a method for finding RkNN in an undirected road network. They proposed two algorithms eager & lazy, both algorithms are quite similar to Dijkstra algorithm. In eager approach they try to prune nodes that cannot be reverse nearest neighbor of query point $q$ proactively. In lazy approach they exploit the verification phase of their algorithm to prune nodes for their future searches. However, both these algorithms work for undirected road network only as the exploit symmetric property of undirected graphs which is not applicable to directed road networks.

In this paper we propose an algorithm SWIFT that can efficiently find reverse nearest neighbors of a moving query in a directed road network. We present a technique based on safe exit point which is a mean to overcome excessive computation and communication cost associated with timestamps based continuous monitoring.

SWIFT is based on the technique of safe exit points. Safe exit points are the boundary points of the safe region and since safe region is comprised of road segments and safe exit points are just points in the road network, as a result less network bandwidth is consumed during communication.

2.2. Limitations of Undirected Algorithms in Directed Graphs

For better presentation we create Table 1 that summarizes all the notations we use in this paper. SWIFT is closest to Man L Yiu. [4] algorithm. Fig. 2a shows an undirected road network where there are two data objects $o_1, o_2$ a query object $q$ which are denoted by rectangles and triangle, respectively. Here, $k = 1$ (number of RNN to find). The distances between $q$ to $o_1$ and $o_2$ is 1 and 5 respectively. Thus, the query result $O_1 = o_1$ as $d(q, o_1) = 1$ and $d(o_1, q) = d(q, o_1)$ due to symmetric nature of undirected graphs. However, we can clearly see in Fig. 2b the query result $O_1 = o_2$ as $d(q, o_1) = 1$ but $d(o_1, q) = 9$ which is greater than $d(o_2, q) = 5$. Thus $o_2$ is the correct RNN for query object $q$.

3. PRELIMINARIES AND PROBLEM DESCRIPTION

Section 3. A defines the terms and notations used overall in the paper.
3.1. Definitions and Notations

3.1.1. Directed Road Network

A directed road network is a type of road network where each edge has a particular orientation. For example $nsne$ is a sequence which has no orientation assigned to it. In case of such sequences we are allowed to travel them in any direction either it be $ns$ to $ne$ or $ne$ to $ns$. However, in case of $\rightarrow nsne$ we can only travel it from $ns$ to $ne$. Consider segment $\rightarrow n_1n_2$ in Fig. 2b, it is only travelable from $n_1$ to $n_2$.

3.1.2. Nodes Classification

A node can belong to one of the following categories.

- A node is referred as terminal node if the degree of node is 1.
- A node is referred as intermediate node if the degree of node is exactly 2.
- Finally, if the degree of node is larger than 2, it is referred as intersection node.

3.1.3. Sequence

A sequence is a set of node such as $ns, ns+1, ns+2...ne-1, ne$ denotes a path from $ns$ to $ne$, such that $ns$ and $ne$ must either be an intersection node or a terminal node and all nodes in-between $ns$ and $ne$ must be intermediary nodes. For the sake of simplicity we assume that all edges in a sequence have same direction. Note that in a directed road network $d(ns, ne) \neq d(ne, ns)$ as it is not symmetric like an undirected road network. Here $d(ns, ne)$ denotes the weight of the sequence.

3.1.4. Spatial Network

A graph is represented by $G(N,E)$, where $N$ is a set of nodes and $E$ is a set of edges. Each edge in a graph has exactly two distinct nodes and weight is assigned to it, which is the cost of traveling the edge. In case of road networks this cost can be considered as time taken to travel the edge which may change many times during a day depending upon the traffic on road. It also has an orientation assigned to it, which can be directed or undirected. Throughout this paper we use $\overline{nsne}$ to represent an undirected edge where $ns$ and $ne$ are boundary nodes of the sequence, whereas $\rightarrow nsne$ is used to show directed sequence. Of course the direction of arrow represents the direction of the sequence. We use $\rightarrow nin_j$ to denote a directed segment. A segment is a single edge in a graph so, a sequence can be compromised of many edges. Similarly, we use $\overline{nin_j}$ to represent an undirected edge in the graph where $i$ and $j$ are some positive integers.

3.2. Problem Description

To provide clear explanation of SWIFT algorithm, we use a directed road network shown in the Fig. 3a. It has following characteristics; it has six data objects denoted by squares, i.e. $o_1, o_2...o_6$, three directed edges, i.e. $\rightarrow n_4n_1$, $\rightarrow n_5n_7$ and $\rightarrow n_7n_6$, it has seven undirected edges, i.e. $\overline{n_3n_4}$, $\overline{n_4n_5}$, $\overline{n_5n_6}$, $\overline{n_5n_7}$, $\overline{n_2n_1}$,
Let's assume we have a node such that \( d(n, q) > d(n, \{ o_k \}) \) holds true where \( o_k \) can be any object.

**Lemma 1.** If a choke node is part of shortest path from \( o \) to \( q \), then such object cannot be RNN of query object \( q \).

**Proof.** Let's assume we have a node such that \( d(n, q) > d(n, \{ o_k \}) \) holds true. Let \( o_1 \) and \( o_2 \) be the nodes closer to the node \( n \) than query object \( q \). Let \( o_3 \) be the object which has node \( n \) in its shortest path to \( q \). Distance from \( o_1 \) to \( q \) will be equal to \( d(o_1, n) + d(n, q) \) but we already know that \( d(n, q) > d(n, o_1, o_2) \). Hence \( \{ o_1, o_2 \} \) will automatically become NN for \( o_3 \) so it cannot be RNN of \( q \).

\( \square \)

**Lemma 2.** All intermediate nodes of a sequence become choke if both \( n_s, n_e \) are choke nodes.

**Proof.** Let's assume a sequence with infinite length. When both \( n_s \) and \( n_e \) are choke nodes all intermediate nodes will also be choke as the shortest path of any object in this sequence will either pass through \( n_s \) or \( n_e \) and using Lemma 1 they cannot be RNN for query point. Hence, all intermediate nodes become choke nodes.

\( \square \)

In Fig. 3a when \( k = 2 \), \( n_4 \) is a choke node as \( d(n_4, q) = 5 \) while \( d(n_4, o_2) = 2 \), \( d(n_4, o_1) = 1 \) and \( d(n_4, o_3) = 2 \).

### 4. SWIFT REVERSE NEAREST NEIGHBOR

Figure. 3b shows the final result with safe exit points. SWIFT algorithm has three phases. In phase 1, we find reverse nearest neighbors while in phase 2 we find influence region of objects found in phase 1. In phase 3 we find safe exit points. Section 4.1 describes our SWIFT algorithm used to find RNN in a directed road network. Section 4.2 explains running example of Fig. 3a.

#### Table 2. Summary of Notations

<table>
<thead>
<tr>
<th>Steps</th>
<th>State of Min-heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( qn_5, n_4q )</td>
</tr>
<tr>
<td>2</td>
<td>( n_4q, n_3q_6, n_5n_6 )</td>
</tr>
<tr>
<td>3</td>
<td>( n_1n_5, n_5n_6, n_3n_7 )</td>
</tr>
<tr>
<td>4</td>
<td>( n_1n_5, n_5n_6, n_3n_7, n_1n_6, n_4n_7 )</td>
</tr>
<tr>
<td>5</td>
<td>( n_5n_6, n_3n_7, n_1n_6, n_4n_7 )</td>
</tr>
<tr>
<td>6</td>
<td>( n_3n_7, n_1n_6, n_4n_7 )</td>
</tr>
<tr>
<td>7</td>
<td>( n_1n_6, n_3n_7, n_1n_6, n_4n_7 )</td>
</tr>
<tr>
<td>8</td>
<td>( n_5n_6, n_3n_7, n_1n_6, n_4n_7 )</td>
</tr>
<tr>
<td>9</td>
<td>( n_5n_6, n_3n_7, n_1n_6, n_4n_7 )</td>
</tr>
<tr>
<td>10</td>
<td>( n_5n_6 )</td>
</tr>
<tr>
<td>11</td>
<td>( n_4n_7 )</td>
</tr>
</tbody>
</table>

#### 4.1. Overview

SWIFT transverses the road network incrementally, similar to Dijkstra's algorithm. It starts expanding the network from the location of query object \( q \) in an increasing order of distance from the query object. Whenever, a sequence \( n_5n_6 \) is popped, it is examined for data objects and if any data object is found, it becomes a candidate answer object and a range-NN [8, 9] query is issued to verify if \( k \) nearest neighbors of object under consideration contains query point \( q \). This process continues until the min-heap is empty.

**Algorithm: SWIFT**

**Input:** \( q \) query location, \( k \) no of RNN

**Output:** \( O_k \) query result, \( \omega_{\text{safe,exit}} \)

1: \( \text{min-heap} \leftarrow \emptyset \) /* min-heap is min priority queue */
2: \( \text{explored} \leftarrow \emptyset \) /* keeps information of visited sequences and their corresponding base nodes */
3: \( \text{choke-nodes} \leftarrow \emptyset \) /* keeps track of choke nodes */
4: \texttt{min-heap.insert}$(\text{seq}_{\text{active}}, n_s, d(n, q))$ /* here $n_s$ denotes $\text{seq}_{\text{base}}$ base node */
5: $\text{NA} \leftarrow \emptyset$ /* List of non answer objects */
6: \textbf{while} $\text{min-heap} \neq \emptyset$ do
7: \textbf{if} $< seq, d(q, n) > \leftarrow \text{min-heap.pop}()$ then
8: $\text{explored} \leftarrow \text{explored} \cup \{\text{seq}\}$
9: \textbf{for all} data objects $o$ do
10: $\text{knn}($seq$_{\text{base}}$) $\leftarrow$ range-NN$(n_s, k, d(n, q))$
11: \textbf{if} $q \in \text{knn}($seq$_{\text{base}}$) then
12: $O_k^l \leftarrow O_k^l \cup o$
13: \textbf{else if} $\text{knn}($seq$_{\text{base}}$) $\geq k$ and $q \notin$\text{choke-nodes} then
14: \text{choke-nodes} $\leftarrow$ \text{choke-nodes} $\cup$ seq$_{\text{base}}$
15: $\text{min-heap.update}$(adj seq of $n_e, d(q, n_s) + d(\text{seq}))
16: \text{NA} $\leftarrow$ NA $\cup$ o
17: \textbf{else}
18: $\text{NA} $\leftarrow$ NA $\cup$ o
19: \textbf{end if}
20: \textbf{end for}
21: \textbf{end if}
22: \textbf{end while}
23: $\omega_{\text{safe.exit}} \leftarrow \text{ComputeSafeRegion}(O_k^l, \text{NA}, q, k)$
24: \textbf{return} $O_k^l$, $\omega_{\text{safe.exit}}$

Fig. 4 present details of the SWIFT algorithm used to find RkNN for a query point $q$. It initializes a min-heap with an empty set and starts expanding the network from the active sequence where the query object $q$ is found. When a query object is found in a sequence it is broken into two sequences and query object $q$ is made their base node seq$_{\text{base}}$, while for any other sequence either $n_s$ or $n_e$ becomes its base node. Whenever, a new candidate object $o'$ is found in a sequence a range-NN query is issued, where range is equal to $d(q, o')$. Recall that in case of directed road network $d(q, o') \neq d(o', q)$. If $\text{KNN(seq}_{\text{base}}) \geq k$ and $k \in \text{KNN(seq}_{\text{base}})$ then, the base node of active sequence is marked as choke node using Lemma 1. However, if Lemma 1 is not applicable then adjacent sequences of the sequence end node $n_e$ are added to min-heap and $n_e$ is made seq$_{\text{base}}$ node for these adjacent sequences. When the min-heap is exhausted the algorithm stops and $O_k^l$ is returned. It should be noted that the network expansion is in opposite direction of the direction of roads.

Fig. 10 present details of algorithm used to find safe exit points in the graph while Fig. 7 shows how to calculate influence region of objects.

During the execution of algorithm we cache the results of all nodes and data objects, so multiple requests from same nodes and objects can be avoided. After the completion of algorithm the cache is cleared in order to make memory foot print.
4.2. Running Example

We now discuss the working of SWIFT for given query point \( q \) in Fig. 3a. We are evaluating the result for \( O_2^q \), here \( k = 2 \). Table 2 shows min-heap states during the execution of algorithm.

As discussed earlier in Section 4.1 SWIFT will start from the active sequence \( n_4n_5 \) where the query object \( q \) exists. Sequence \( n_4n_5 \) is broken into two sequences \( n_4q \) and \( qn_5 \), each is added into min-heap to be evaluated separately. First pop results in \( qn_5 \), but no data points are found in this sequence. Thus, adjacent sequences \( n_1n_5 \) and \( n_5n_6 \) of \( n_5 \) is added to min-heap. Next, \( n_4q \) is popped and since no data points are found on this sequence, adjacent sequences \( n_3n_4 \) and \( n_4n_7 \) of \( n_4 \) are added to min-heap.

As shown in Fig. 5a we discover our first candidate object \( o_6 \) on our sequence \( n_1n_5 \), and when a range-NN query of 4 is issued it is determined that it is in fact our RKNN. As 2NN of \( o_6 = \{ q, o_5 \} \). Thus, adjacent sequences \( n_4n_1 \) and \( n_1n_2n_6 \) are added to the min-heap.

As show in Fig. 5b we discover our second

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.png}
\caption{Examples (a) \( n_1n_3 \) (b) \( n_3n_4 \) (c) \( n_4n_7 \)}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.png}
\caption{Examples (a) \( n_1n_2n_6 \) (b) \( n_4n_1 \) (c) \( n_1n_6 \)}
\end{figure}

small.
candidate object $o_1$ on our sequence $\pi_5n_6$, a range-NN of 6 query is issued, and it is determined that $KNN(n_4) = \{o_2, o_3\}$ as $d(n_4, o_2) = 2$, $d(n_4, o_3) = 2$, while $d(n_4, q) = 5$. Hence, node $n_4$ is marked as choke node. As there are no adjacent sequences of node $n_3$ no new sequence is added to min-heap and the execution continues.

As the algorithm continues its execution, next it pops $\pi_5n_6$ but as no data object is found, the adjacent sequence $\pi_5n_7$ is added to min-heap. As shown in Fig. 5c next, $\pi_5n_7$ is popped, but recall that $n_4$ was marked as choke so the sequence is immediately pruned and after adding adjacent sequence $\pi_5n_7$ to the min-heap the algorithm moves to next sequence.

As shown in Fig. 6a, next $\pi_7n_6$ is popped and object $o_5$ is found which takes the path $n_1n_5q$ and becomes the second answer object. Since, no new adjacent sequences are found nothing is added to min-heap.

Next, sequence $\pi_7n_5$ is popped from min-heap and since no data object is found on this sequence the algorithm continues to next sequence. As shown in Fig. 6b, next sequence $d(n_4, n_1)$ is popped and 2NN of data object $o_2$ are found to be $\{o_5, o_6\}$, therefore node $n_1$ is marked as choke node, and no new sequence is added to min-heap.

As shown in Fig. 6c that the next sequence popped is $\pi_7n_6$ and it is found that 2NN of data object $o_4$ are $\{o_5, o_6\}$. Thus, node $n_6$ is marked as choke node and adjacent sequence $\pi_7n_7$ is again added to min-heap with $n_1$ as $seq_{base}$.

As shown in Fig. 5c, 2NN of $o_3$ are $\{o_4, o_5\}$ but as node $n_6$ is a choke node execution terminates on it. At this step it is found that $R_2 = \{o_6, o_5\}$.

5. Extension of SWIFT to continuous directed road networks

5.1. Influence Region

Before understanding SWIFT algorithm in continuous road networks it is necessary to understand the concept of influence region of answer and non-answer objects. Answer objects are those objects which are RNN of query object $q$. We observe that object is said to be answer object if $d(o, q) < d(o, o_{k+1})$ where $o_{k+1}$ is $k + 1^{th}$NN of object $o$. Similarly an object is said to be non-answer object if $d(o, q) > d(o, o_k)$ where $o_k$ is $k^{th}$NN of object $o$.

The influence region of answer object can be described as follows:

$$I(o)^+ = \{p|d(o, p) \leq d(o, o_{k+1})\}$$

In simple words for any object $o$ influence region is equal to the distance between object $o$ and its $k^{th}$NN object.

The influence region of non-answer object can be described as follows:

$$I(o)^- = \{p|d(o, p) > d(o, o_k)\}$$

**Computation of Influence Region**

**Input:** $o$ object, $k$ no of RNN  
**Output:** $I(O)$ Influence region

1: if $o$ is answer object then  
2: $d \leftarrow d(o, o_{k+1})$ /* distance from object $o$ to $k + 1^{th}$NN of $o$ */  
3: $I(O) =$ Expand region of object $o$ by $d$  
4: else  
5: $d \leftarrow d(o, o_k)$ /* distance from object $o$ to $k$NN of $o$ */  
6: $I(O) =$ Expand region of object $o$ by $d$  
7: end if  
8: return $I(O)$

**Fig. 7. CalcInfluence**

5.2. Influence Region in Running Example

In Fig. 3a object $o_6$ is an answer object and 2NNs of $o_6 = \{q, o_5\}$ where $d(o_6, q) = 3$ and $d(o_6, o_5) = 4$. 

![Fig. 8. Influence Region for o6](image-url)
For computing influence region of $o_6$ we need to compute the distance of $o_6$ to $k^{th}$NN, which is $o_4$ and $d(o_6, o_4) = 8$. So, influence region for object $o_6$ will be distance of 8 from $o_6$. The influence region comprises road segments $n_5n_7$, $n_2n_5$, $n_6n_5$, $n_7n_6$, $n_4n_7$, $n_3o_4$, $n_7o_3$ complete segments and $n_4n_5$ with distance of 7. Figure. 8 shows influence region for object $o_6$. Similarly, the influence region for $o_5$ can be determined.

Considering Fig. 3a object $o_1$ is non-answer object and 2NNs of $o_1 = (o_2, o_5)$ where $d(o_1, o_2) = 3$ and $d(o_1, o_5) = 3$. For computing influence region of non-answer object we use distance of $k^{th}$NN which is 3 in this case. So, influence region for object will distance of 3 from $o_1$. The influence region comprises road segments $n_2n_5$, $n_3n_7$, $n_5o_1$ complete and $n_4n_5$ with distance of 2. Figure. 9 shows the influence region for object $o_1$.

![Influence Region for $o_2$ and $o_1$](image)

Figure. 9 shows the influence region for object $o_4$ which is a non-answer object. 2NN for $o_4$ are $o_6$ and $o_5$. Here, $k^{th}$NN of $o_4$ is $o_6$ and $d(o_4, o_6) = 8$. So, influence region will be the distance of 8 from $o_4$. The influence region comprises road segments $o_4n_6$, $n_2n_5$, $n_6n_7$ complete and $n_2n_7$, $o_4n_5$, $n_5n_3$ and $n_4n_5$ with distances of $(4, 1, 1, 1)$ respectively.

5.3. Safe Exit

Computation of Safe Exit Points

**Input:** A answer objects, NA non answer objects, $q$ query location, $k$ no of RNN

**Output:** $\omega_{safe, exit}$ Safe exit points

1: $I(O)^+ \leftarrow \emptyset$ /* Influence region of answer objects */
2: $I(O)^- \leftarrow \emptyset$ /* Influence region of non-answer objects */
3: while $A \neq \emptyset$
4: $o \leftarrow A.pop()$
5: $I(O)^+ \leftarrow CalcInfluence(o,q,k) \cap I(O)^+$
6: end while
7: while $NA \neq \emptyset$
8: $o \leftarrow NA.pop()$
9: $I(O)^- \leftarrow CalcInfluence(o,q,k) \cup I(O)^-$
10: end while
11: $\omega_{safe, exit} = I(O)^+ - I(O)^-$
12: return $\omega_{safe, exit}$

Fig. 10. ComputeSafeRegion

A safe region in euclidean space is defined as follows:

$$S(q, r) = \{ \cap I(O)^+ - \cup I(O)^- \}$$

Here, $q$ is the query object and $r$ is the range of the query. Figure. 11 shows how a safe region is computed in euclidean space. In the fig. 11 we assume we have a dataset of four objects $o_1, o_2, o_3, o_4$. The answer objects of query $q$ are $o_1, o_3$ while non-answer objects are $o_2, o_4$. The intersection of all answer objects is considered to be part of influence region and subtracting union of non-answer objects influence region from this region gives us the correct safe region of the query $q$.

![Euclidean Safe Region](image)
6. EXPERIMENT AND PERFORMANCE EVALUATION

Section 6.1 explains our experimental settings and in Section 6.2 we discuss the results of our experiment by comparing them to naïve algorithm.

6.1. Experimental Setup

We use a real world data road map [10] of North America (NA). It is a world of $5000 \times 5000km^2$, comprising of 175,813 nodes and 179,179 edges. It has been used in many performance evaluations as well [8, 11]. The direction to the edges is assigned randomly and skewed data points distribution is opted. For comparing the performance of this algorithm we also report the performance of baseline algorithm, which answers the query by finding nearest neighbors for all data points in the road network.

6.2. Results Evaluation

Fig. 12 shows the effect of $k$ on the query processing time. The query processing time increases for both naïve and SWIFT algorithm with $k$. However, the query time of naïve algorithm is affected linearly, as the value of $k$ is increasing linearly and naïve algorithm finds kNN for every data object in the network. Hence, its time also increases linearly. We observe that when value of $k$ is small continuous takes almost similar amount time as static algorithm but as the value of $k$ increases the time for continuous increases much rapidly as it has to compute more and more influence regions.

Fig. 13 shows the effect POIs on query processing time. The query processing time of naïve algorithm decreases dramatically, because when the number of edges with data points are few, naïve algorithm has to traverse more sequences in order to find kNN for each data point in the road network. However for SWIFT the time increases as the number of edges having data points increase. This is because when there are many data points the calls to range-NN also increases which make the algorithm expand the network many times. Continuous performs worse when there are fewer POIs but as the number of POIs increase more and more influence regions are needed to be calculated and but when POIs are more influence regions tend to be smaller so the performance improves a little and the gap be-
Fig. 14 shows the effect of $k$ on node pruning. Number of pruned sequences decreases as the value $k$ is increased. It is because when $k$ is large more data objects become RNN of query point $q$. Hence, fewer nodes are marked as choke nodes and more sequences are expanded which results in lower number of choked sequences.

Fig. 15 shows the effect of speed on number of queries. The naïve algorithm incurs constant number of queries as we query at specific timestamps but on the other hand queries for SWIFT increases as the speed of the query object increases as the object starts leaving the safe region more quickly.

7. Conclusion

In this paper we studied RkNN in a directed road network and proposed a new algorithm called SWIFT. Experimental results indicate that the algorithm with pruning rules reduced the computation time significantly. In future we want to study RkNN in a directed road network where query object is moving in the road network and the cost of traveling an edge is not constant.

References

References are to be listed in the order cited in the text. Use the style shown in the following examples. For journal names, use the standard abbreviations. Typeset references in 9 pt Times Roman.


