International Journal of Networked and Distributed Computing, Vol. 3, No. 4 (November 2015), 261-272

A Pruning Algorithm for Reverse Nearest Neighbors in Directed Road Networks

Rizwan Qamar [1] , Muhammad Attique [2] , Tae-Sun Chung [3]

[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.,

each road is either directed or undirected). A safe region is a region in which result of the query doesn't change when the query object moves within safe region.

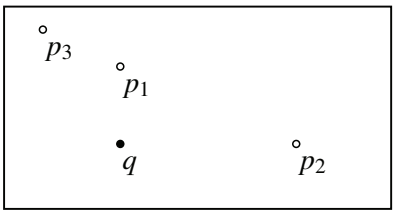

Fig. 1. Example Road Network

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. $RkNN(q) = o \in d(o,q) \leqslant d(o,o_k)$ 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

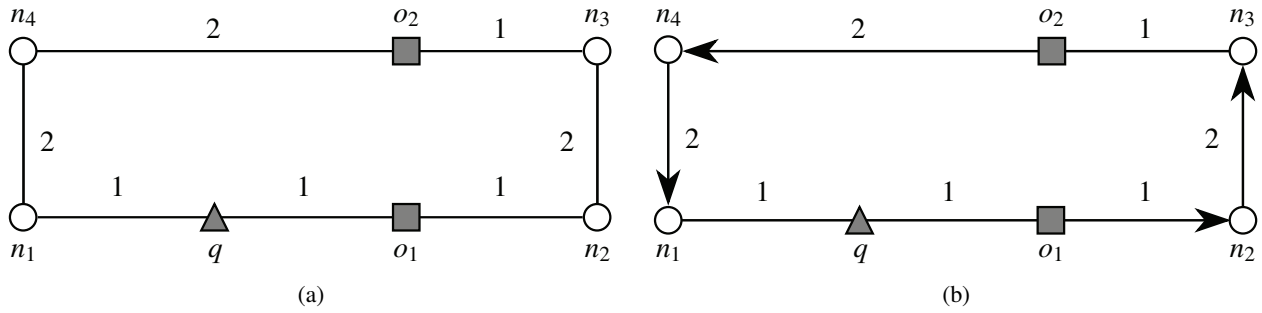

Figure 2: Examples (a) undirected road network (b) directed road network

space into six equal regions of 60°. 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 is $O_1^q = o_1$ 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 $O_1^q = o_2$ 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.

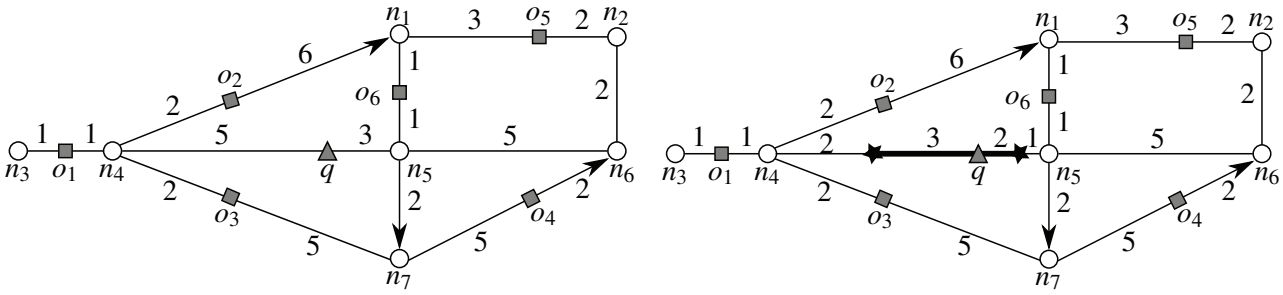

Figure 3: Examples (a) Example Road Network (b) Example Road Network with Safe Exits

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 $\overline{n_s n_e}$ 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 n_s to n_e or n_e to n_s. However, incase of $\overrightarrow{n_s n_e}$ we can only travel it from n_s to n_e. Consider segment $\overrightarrow{n_1 n_2}$ 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 $\overline{n_s, n_{s+1}, n_{s+2} ... n_{e-1}, n_e}$ denotes a path from n_s to n_e, such that n_s and n_e must either be an intersection node or a terminal node and all nodes in-between n_s and n_e 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 $d(n_s, n_e) \neq d(n_e, n_s)$ as it is not symmetric like an undirected road network. Here $d(n_s, n_e)$ 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 $\overline{n_s n_e}$ to represent an undirected edge where n_s and n_e are boundary nodes of the sequence, whereas $\overrightarrow{n_s n_e}$ is used to show directed sequence. Of course the direction of arrow represents the direction of the sequence. We use $\overrightarrow{n_i n_j}$ 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 $\overline{n_i n_j}$ 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. $\overrightarrow{n_4 n_1}$, $\overrightarrow{n_5 n_7}$ and $\overrightarrow{n_7 n_6}$, it has seven undirected edges, i.e. $\overline{n_3 n_4}$, $\overline{n_4 n_5}$, $\overline{n_5 n_6}$, $\overline{n_4 n_7}$, $\overline{n_2 n_1}$,

$\overline{n_6n_2}$, and $\overline{n_1n_5}$, a query point q denoted by a triangle. The numbers on the line denote the distances e.g. $d(n_3,o_1)=1$ while $d(n_3,n_4)=2$.

Road network in Fig. 3a is comprised of nine sequences i.e. $\overline{n_1n_2n_6}$, $\overline{n_1n_5}$, $\overrightarrow{n_4n_1}$, $\overline{n_3n_4}$, $\overline{n_4n_7}$,$\overline{n_4n_5}$, $\overline{n_5n_6}$, $\overrightarrow{n_5n_7}$, $\overrightarrow{n_7n_6}$. Here, $O_2^q=\{o_5,o_6\}$ as both o_5 and o_6 have q as their nearest neighbor when $k=2$. Notice o_1 is not in the query result as its two nearest neighbors are $\{o_2,o_3\}$ and this makes n_4 a choke node.

Before we present our pruning lemmas we need to define the term choke node. A node is said to be a choke if there exists at least k objects 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 $d(n,q)>d(n,\{o_1,o_2\})$ 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_3 to q will be equal to $d(o_3,n)+d(n,q)$ 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. □

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. □

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

Steps	State of Min-heap
1	$\overline{qn_5},\overline{n_4q}$
2	$\overline{n_4q},\overline{n_1n_5},\overline{n_5n_6}$
3	$\overline{n_1n_5},\overline{n_3n_4},\overline{n_5n_6},\overline{n_4n_7}$
4	$\overline{n_3n_4},\overline{n_5n_6},\overline{n_4n_7},\overline{n_1n_2n_6},\overrightarrow{n_4n_1}$
5	$\overline{n_5n_6},\overline{n_4n_7},\overline{n_1n_2n_6},\overrightarrow{n_4n_1}$
6	$\overline{n_4n_7},\overline{n_1n_2n_6},\overrightarrow{n_4n_1},\overrightarrow{n_7n_6}$
7	$\overline{n_1n_2n_6},\overrightarrow{n_5n_7},\overrightarrow{n_4n_1},\overrightarrow{n_7n_6}$
8	$\overrightarrow{n_5n_7},\overrightarrow{n_4n_1},\overrightarrow{n_7n_6}$
9	$\overrightarrow{n_4n_1},\overrightarrow{n_7n_6}$
10	$\overrightarrow{n_7n_6}$
11	$\overline{n_4n_7}$

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 $\overline{n_sn_e}$ 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^q query result, ω_{safe_exit}

1: *min-heap* $\leftarrow \emptyset$ /* min-heap is min priority queue */
2: *explored* $\leftarrow \emptyset$ /* keeps information of visited sequences and their corresponding base nodes */
3: *choke-nodes* $\leftarrow \emptyset$ /* keeps track of choke nodes */

Table 1. Summary of Notations

Notations	Explaination
$G(N,E)$	The graph model of road network where,denotes nodes and,denotes edges in graph.
n_i	A node in the graph, where,is integer e.g. n_1
n_s	Start node of a sequence.
n_e	End node of a sequence.
$\overline{n_s,n_{s+1}...n_e}$	A sequence in graph, where n_s, is start node and n_e is end,node of a sequence.
$d(n_i,n_j)$	Network distance between node n_i and n_j.
k	Represent number of NN to be found.
$\overrightarrow{n_jn_j}$	Represent directed edge between n_i and n_j.
$\overline{n_in_j}$	Represent undirected edge between n_i and n_j.
O_k^q	Represent answer objects for RkNN(q) and k denotes number of nearest neighbors.
seq_{base}	Represent start point of network expansion for the sequence.
$I(o)^+q$	Represents influence region of answer objects
$I(o)^-q$	Represents influence region of non-answer objects

4: *min-heap.insert*$(seq_{active}, n_s, \overline{d(n,q)})$ /* here n_s denotes seq_{base} base node */
5: $NA \leftarrow \emptyset$ /* List of non answer objects */
6: **while** *min-heap* $\neq \emptyset$ **do**
7: $< seq, d(q,n) > \leftarrow$ *min-heap.pop*()
8: **if** $< seq, d(q,n) > \notin$ *explored* **and** $seq_{base} \notin$ *choke-nodes* **then**
9: *explored* $\leftarrow$ *explored* $\cup \{seq\}$
10: **for all** data objects o **do**
11: $knn(seq_{base}) \leftarrow$ *range-NN*$(n_s, k, \overrightarrow{d(n,q)})$
12: **if** $q \in kNN(seq_{base})$ **then**
13: $O_k^q \leftarrow O_k^q \cup o$
14: **else if** $kNN(seq_{base}) \geqslant k$ **and** $q \notin knn(seq_{base})$ **then**
15: *choke-nodes* $\leftarrow$ *choke-nodes* $\cup$ seq_{base}
16: *min-heap.update*$(adj\_seq\_of\_n_e, d(q,n_s) + d(seq))$
17: $NA \leftarrow NA \cup o$
18: **else**
19: $NA \leftarrow NA \cup o$
20: **end if**
21: **end for**
22: **end if**
23: **end while**
24: $\omega_{safe\_exit} \leftarrow ComputeSafeRegion(O_k^q, NA, q, k)$
25: **return** O_k^q, ω_{safe_exit}

Fig. 4. Algorithm: SWIFT

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_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 $KNN(seq_{base}) \geqslant k$ and $k \in KNN(seq_{base})$ 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_{base} node for these adjacent sequences. When the min-heap is exhausted the algorithm stops and O_k^q 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

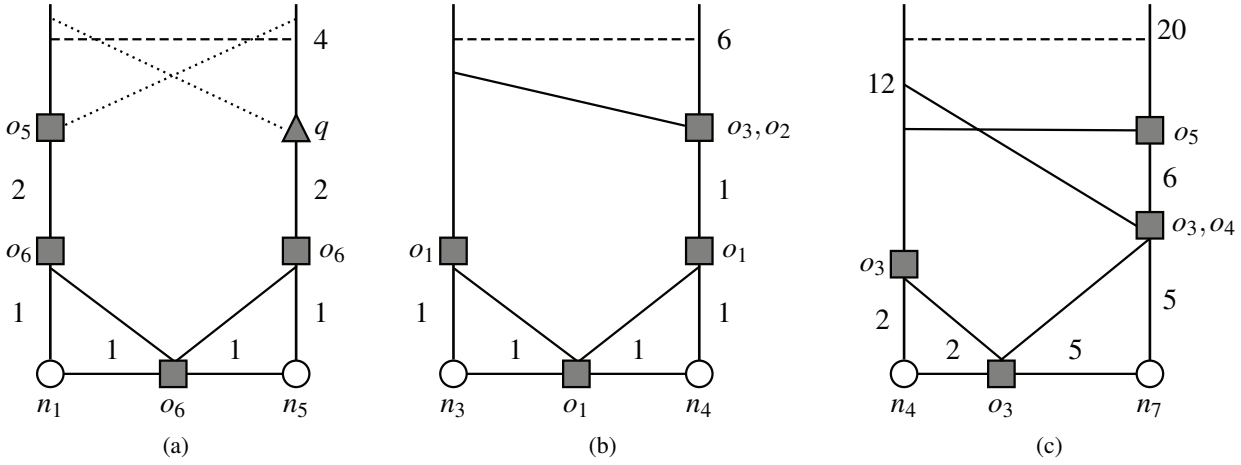

Figure 5: Examples (a) $\overline{n_1 n_5}$ (b) $\overline{n_3 n_4}$ (c) $\overline{n_4 n_7}$

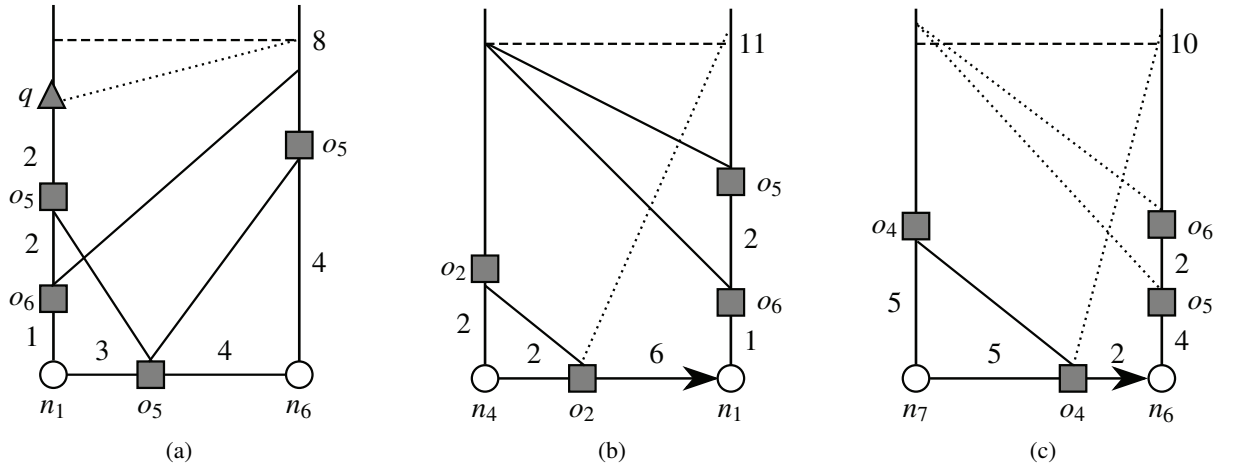

Figure 6: Examples (a) $\overline{n_1 n_2 n_6}$ (b) $\overrightarrow{n_4 n_1}$ (c) $\overrightarrow{n_7 n_6}$

small.

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 $\overline{n_4 n_5}$ where the query object q exists. Sequence $\overline{n_4 n_5}$ is broken into two sequences $\overline{n_4 q}$ and $\overline{q n_5}$, each is added into min-heap to be evaluated separately. First pop results in $\overline{q n_5}$, but no data points are found in this sequence. Thus, adjacent sequences $\overline{n_1 n_5}$ and $\overline{n_5 n_6}$ of n_5 is added to min-heap. Next, $\overline{n_4 q}$ is popped and since no data points are found on this sequence, adjacent sequences $\overline{n_3 n_4}$ and $\overline{n_4 n_7}$ of n_4 are added to min-heap.

As shown in Fig. 5a we discover our first candidate object o_6 on our sequence $\overline{n_1 n_5}$, 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 $\overrightarrow{n_4 n_1}$ and $\overline{n_1 n_2 n_6}$ are added to the min-heap.

As show in Fig. 5b we discover our second

candidate object o_1 on our sequence $\overline{n_3n_4}$, a range-NN of 6 query is issued, and it is determined that $KNN(n_4) = \{o_2, o_3\}$ as $\overrightarrow{d(n_4,o_2)} = 2$, $\overrightarrow{d(n_4,o_3)} = 2$, while $\overrightarrow{d(n_4,q)} = 5$. 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 $\overline{n_5n_6}$ but as no data object is found, the adjacent sequence $\overrightarrow{n_7n_6}$ is added to min-heap. As shown in Fig. 5c next, $\overline{n_4n_7}$ is popped, but recall that n_4 was marked as choke so the sequence is immediately pruned and after adding adjacent sequence $\overrightarrow{n_5n_7}$ to the min-heap the algorithm moves to next sequence.

As shown in Fig. 6a, next $\overline{n_1n_2n_6}$ 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 $\overrightarrow{n_5n_7}$ 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 $\overrightarrow{d(n_4,n_1)}$ 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 $\overrightarrow{n_7n_6}$ and it is found that 2NN of data object o_4 are $\{o_5, o_6\}$. Thus, node n_6 is marked as coke node and adjacent sequence $\overline{n_4n_7}$ is again added to min-heap with n_7 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 $O_2^q = \{o_6, o_5\}$.

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 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:

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

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:

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

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 kNN 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

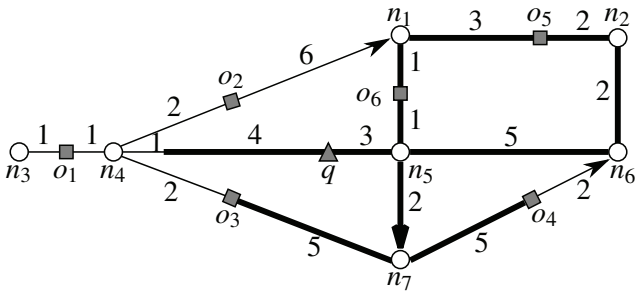

Fig. 8. Influence Region for o_6

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 $d(o_6,o_5) = 4$.

For computing influence region of o_6 we need to compute the distance of o_6 to $k+1^{th}$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 $\overline{n_1n_5}$, $\overline{n_2n_1}$,$\overline{n_6n_2}$,$\overline{n_5n_6}$, $\overrightarrow{n_5n_7}$, $\overrightarrow{n_7o_4}$, $\overrightarrow{n_7o_3}$ complete segments and $\overline{n_4n_5}$ 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 $o_1=(o_2,o_3)$ where $d(o_1,o_2)=3$ and $d(o_1,o_3)=3$. 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 compromises road segments $\overline{n_3n_4}$, $\overline{n_4o_3}$, $\overrightarrow{n_4o_2}$ complete and $\overline{n_4n_5}$ with distance of 2. Figure. 9 shows the influence region for object o_1.

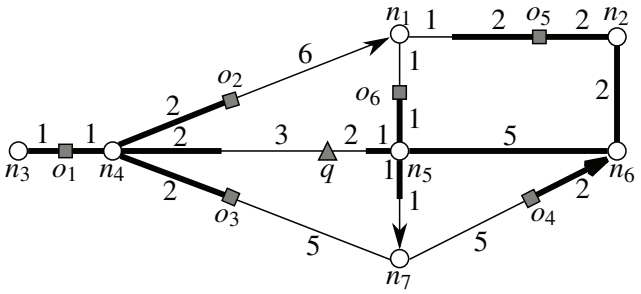

Fig. 9. Influence Region for o_4 and o_1

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 $\overrightarrow{o_4n_6}$, $\overline{n_5n_6}$, $\overline{n_6n_2}$ complete and $\overline{n_2n_1}$, $\overline{o_6n_5}$, $\overrightarrow{n_5n_7}$ and $\overline{n_4n_5}$ 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: ω_{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** $A \neq \emptyset$ **do**
4: $o \leftarrow A.pop()$
5: $I(O)^+ \leftarrow CalcInfluence(o,q,k) \cap I(O)^+$
6: **end while**
7: **while** $NA \neq \emptyset$ **do**
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** ω_{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.

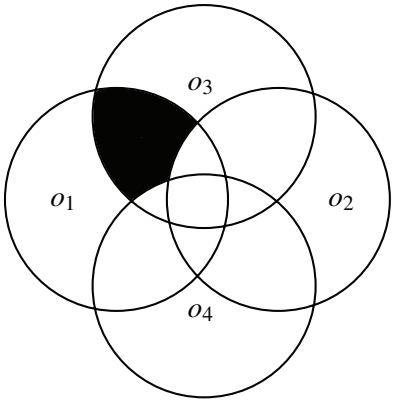

Fig. 11. Euclidean Safe Region

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 $5000 \times 5000 km^2$, 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 of 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

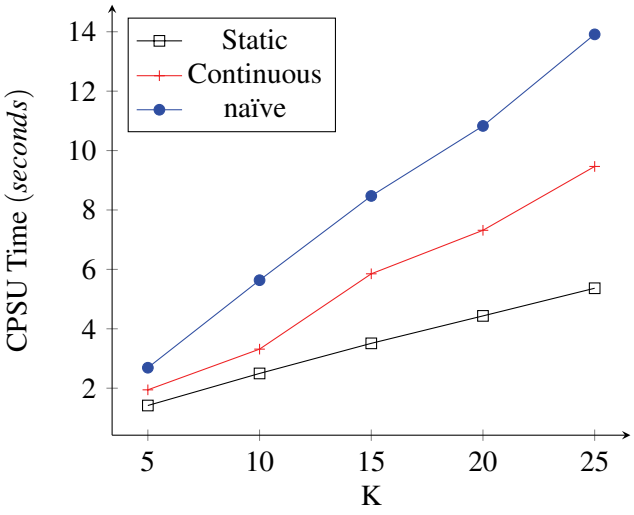

Fig. 12. Effect of k on CPU

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.

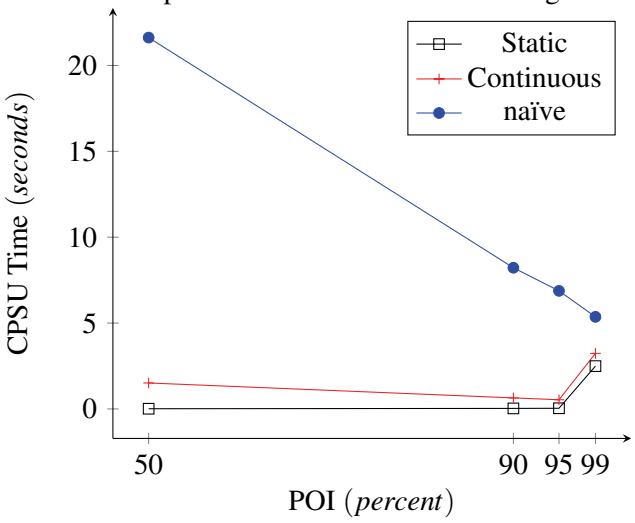

Fig. 13. Effect of *POI* on CPU

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 transverse 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-

tween static and continuous decreases.

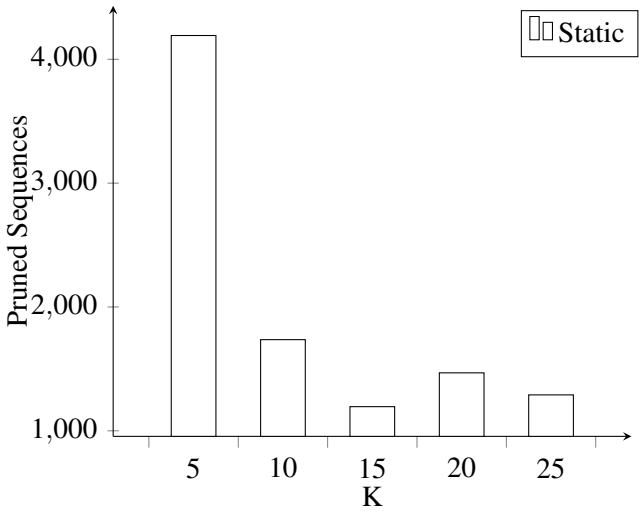

Fig. 14. Effect of k on node pruning

Fig. 14 shows the effect of k on the number of pruned sequences. 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 chocked sequences.

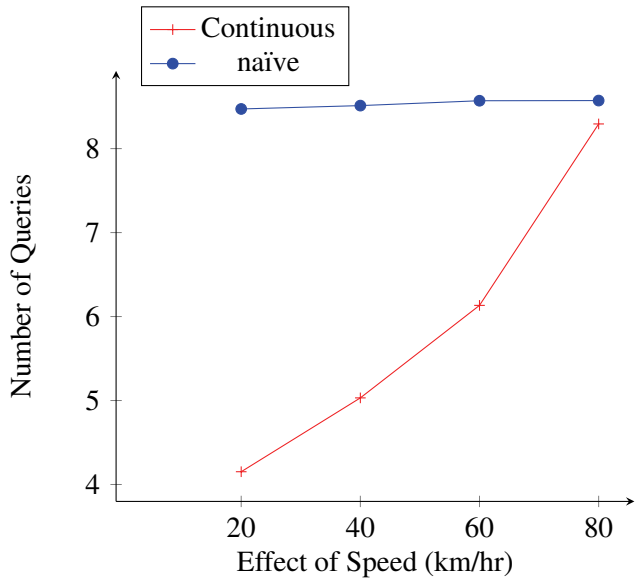

Fig. 15. Effect of speed in km/hr on queries

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.

1. F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 201–212, Jun. 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=335191.335415
2. R. Benetis, C. Jensen, G. Karciauskas, and S. Saltenis, "Nearest neighbor and reverse nearest neighbor queries for moving objects," in *Proceedings International Database Engineering and Applications Symposium*. IEEE Comput. Soc, 2002, pp. 44–53. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1029655
3. W. Wu, F. Yang, C. Y. Chan, and K.-L. Tan, "Continuous Reverse k-Nearest-Neighbor Monitoring," in *The Ninth International Conference on Mobile Data Management (mdm 2008)*. IEEE, Apr. 2008, pp. 132–139. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4511444
4. M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao, "Reverse Nearest Neighbors in Large Graphs," pp. 540–553, Apr. 2006. [Online]. Available: http://www.computer.org/csdl/trans/tk/2006/04/k0540.html

5. M. A. Cheema, W. Zhang, X. Lin, Y. Zhang, and X. Li, "Continuous reverse k nearest neighbors queries in Euclidean space and in spatial networks," *The VLDB Journal*, vol. 21, no. 1, pp. 69–95, May 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2124885.2124903
6. A. Singh, H. Ferhatosmanoglu, and A. c. Tosun, "High dimensional reverse nearest neighbor queries," in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, ser. CIKM '03. New York, NY, USA: ACM, 2003, pp. 91–98. [Online]. Available: http://doi.acm.org/10.1145/956863.956882
7. I. Stanoi, D. Agrawal, and A. E. Abbadi, "Reverse Nearest Neighbor Queries for Dynamic Databases," in *In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000, pp. 44–53.
8. H.-J. Cho, K. Ryu, and T.-S. Chung, "An efficient algorithm for computing safe exit points of moving range queries in directed road networks," *Information Systems*, vol. 41, pp. 1–19, May 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437913001464
9. N. Ishii, I. Torii, T. Nakashima, and H. Tanaka, "Generation and mapping of multi-reducts based on nearest neighbor relation," *IJNDC*, vol. 2, no. 1, p. 1, 2014. [Online]. Available: http://dx.doi.org/10.2991/ijndc.2014.2.1.1
10. "Real Datasets for Spatial Databases: Road Networks and Category Points." [Online]. Available: http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm
11. M. A. Cheema, L. Brankovic, X. Lin, W. Zhang, and W. Wang, "Continuous Monitoring of Distance-Based Range Queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 8, pp. 1182–1199, Aug. 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5669315