Efficiently Monitoring Reverse k Nearest Neighbors in Spatial Networks

Shenlu Wang
( shenlu.wang@gmail.com )

Computer Science Honours Thesis

Submission Date : 3rd June, 2013
Supervisor : Dr. Muhammum Aamir Cheema
Assessor : A.Prof. Wei Wang
Acknowledgements

I would like to thank all those people who have through their support enabled me to complete this thesis.

Firstly, I would like to thank my supervisor Dr. Muhammad Aamir Cheema. He devoted a lot of time throughout my honors year to help me with thesis writing, guild me with literature review, and inspire me with problem solving. He is not just an expert in his area, but also a very good and kind supervisor.

I would also like to thank Prof. Xuemin Lin. He is the head of database research group. He contiguously asks me about my progress and encourages me to read more, think more and think deeper.

I would like to thank A.Prof. Wei Wang to be my accessor.

I would like to thank the many database research group people for their help and accompany. They gave me a great and precious memory.

Finally, I would like to thank my parents for the various things that made doing this thesis possible.
Abstract

Given a set of clients $C$, a set of facilities $F$ and a query $q \in F$, a reverse $k$ nearest neighbors (R$k$NN) query retrieves every client $c \in C$ for which $q$ is one of the $k$ closest facilities. In the past few years, R$k$NN queries have received significant research attention due to its wide range of applications. In this thesis, we study the problem of continuous monitoring of R$k$NN queries in road networks. The state-of-the-art technique is sensitive towards the movement of clients, e.g., whenever a client that is inside the so-called unpruned region changes its location, the existing technique requires expensive verification of whether the client is a R$k$NN of $q$ or not. To address this problem, we utilize the novel concept of influence zone which is a region in the network such that a client $c$ is the R$k$NN if and only if it lies inside this zone. This significantly improves the performance because the problem of continuously monitoring R$k$NN queries is reduced to the problem of continuously monitoring the clients that are inside this zone. Based on several non-trivial observations, we present an efficient algorithm to compute the influence zone. Our extensive experimental study demonstrates that our algorithm is more than an order of magnitude faster than the state-of-the-art algorithm.
Contents

1 Introduction 1
  1.1 Spatial Index 2
     1.1.1 Euclidean Space 3
     1.1.2 Spatial Network 4
  1.2 Spatial Queries 4
     1.2.1 \( k \) Nearest Neighbors Query 5
     1.2.2 Reverse \( k \) Nearest Neighbors Query 6
     1.2.3 Other Query Types 7
  1.3 Contributions 8
  1.4 Thesis Organization 8

2 Related Work 10
  2.1 Snapshot R\(k\)NN Queries in Euclidean Space 10
     2.1.1 Pre-Calculation Based Approach 11
     2.1.2 FINCH 12
  2.2 Snapshot R\(k\)NN Queries in Spatial Networks 14
     2.2.1 Network Voronoi Diagram 14
  2.3 Continuous R\(k\)NN Queries in Euclidean Space 16
     2.3.1 Safe Region and Lazy Updates 16
     2.3.2 Influence Zone 20
  2.4 Continuous R\(k\)NN Queries in Spatial Networks 21
     2.4.1 Multi-way Tree 22
     2.4.2 Safe Region, Unpruned and Monitored Network 23
CONTENTS

3 Overview 27
   3.1 Motivation ................................................. 27
   3.2 Problem Formulation ................................. 29
   3.3 Solution Overview .................................... 31

4 Influence Zone Based Techniques 33
   4.1 Terms and Notations ................................. 33
   4.2 Properties .............................................. 34
   4.3 Identifying Inner Nodes and Bounding Nodes .... 36
   4.4 Calculating Inner Edges and Inner Segments .... 38
       4.4.1 Handling Monotonic Edges ..................... 39
       4.4.2 Handling Non-monotonic Edges ............... 43
   4.5 Extension to Directed Graph ......................... 44
   4.6 Monitoring Phase .................................... 46

5 Experiments 48
   5.1 Experimental Settings ................................. 48
   5.2 Influence Zone Size .................................. 49
   5.3 Running Time ........................................... 50
   5.4 Calculating Influence Zone ......................... 52

6 Concluding Remark 54
   6.1 Conclusion ............................................... 54
   6.2 Future Work ............................................ 55
List of Figures

1.1 Gas stations near The University of New South Wales. . . . . . . 2
1.2 Reverse $k$ Nearest Neighbors Query . . . . . . . . . . . . . . . . 6

2.1 Search Region and Candidate Region . . . . . . . . . . . . . . . . 13
2.2 Network Voronoi Diagram . . . . . . . . . . . . . . . . . . . . . . 15
2.3 The exact location of $q$ on $MN$ is unknown. . . . . . . . . . . . 18
2.4 Approximation of parabola. . . . . . . . . . . . . . . . . . . . . . 18
2.5 Half-space Pruning and Dominance Pruning. . . . . . . . . . . . 19
2.6 Pruned area if exact location of query is known . . . . . . . . . . 20
2.7 Grid Based Index of Influence Zone . . . . . . . . . . . . . . . . 21
2.8 PMR Quad-tree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
2.9 Monitoring Area. . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
2.10 Monitored Network . . . . . . . . . . . . . . . . . . . . . . . . . . 26

3.1 Illustration of R$k$NN queries . . . . . . . . . . . . . . . . . . . 30
3.2 A simple spatial network example. . . . . . . . . . . . . . . . . . . 32

4.1 Influence zone $Z_1 (k = 1)$ . . . . . . . . . . . . . . . . . . . . . 38
4.2 Influence zone $Z_2 (k = 2)$ . . . . . . . . . . . . . . . . . . . . . 41
4.3 Inner segment is the segment pruned by less than $k$ facilities. . . 42
4.4 Handling non-monotonic edge. . . . . . . . . . . . . . . . . . . . . 43
4.5 A simple directed spatial network example. . . . . . . . . . . . . 45
4.6 Index of influence zones. . . . . . . . . . . . . . . . . . . . . . . 46

5.1 Influence zone size. . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
5.2 Total running time. . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
LIST OF FIGURES

5.3 Total running time. ........................................ 51
5.4 Total running time. ........................................ 51
5.5 Influence zone calculation phase CPU time cost. ....... 52
List of Tables

4.1 Notations .............................................. 33
5.1 System Parameters for Experiments ..................... 49
Chapter 1

Introduction

Spatial database has been an active research area for more than three decades, studying the problems of management and query needs for spatial applications such as Geographic Information Systems (GIS). Other applications include Computer Aided Design (CAD), Very-Large-Scale Integration (VLSI) design, and Multimedia Information System (MMIS). The current research is aiming at improving the functionality and performance of spatial database management systems.

Fig. 1.1 illustrates an example application of spatial database. The figure is obtained by entering a query of “Gas stations near The University of New South Wales” using Google Maps. Gas stations are represented as points labeled with balloons alongside the road network which is represented by lines. The University of New South Wales is represented by a polygon filled with grey color.

In a relational database, objects are represented by tuples with attributes, and indexed by some ordering of one attribute. Spatial objects (i.e. points, lines and polygons) can also be associated with id and attributes, but in essence, they are associated with geographic locations, such as x, y coordinates. And the challenges of spatial database raise from the location information. For example, two spatial objects that are close in x coordinates are not necessarily close in y coordinates. If a set of such objects are indexed on their x coordinates, then a query conditioned on their y coordinates requires a sequential scan of all data entries. Another example is about distance. If a set of spatial objects are sorted and indexed according to their distance to some point, then any query conditioned on another point will
result into a sequential scan of all data entries. These two examples demonstrate inconvenience of traditional index methods applied on spatial database.

The iconic attribute of a spatial object, geographic location, is usually defined in either Euclidean Space or Spatial Network. This difference on geographic property result in two types of spatial databases. Early years research had been mainly focused on Euclidean Space, and a trend to address issues regarding Spatial Network emerged in recent years. The later trend demonstrates its importance and necessity, specific for applications serve people’s daily life. For example, when a car driver asks for the nearest gas station, we must take into account the shortest road network distance, which is of more concerns, other than Euclidean distance.

The rest of this chapter gives a background introduction of spatial database. Section 1.1 introduces the most adopted index methods. Section 1.2 introduces the various studied query types. Section 1.3 summarize the contribution of this thesis. Section 1.4 introduces the organization of the rest of this thesis.

### 1.1 Spatial Index

Spatial indices, sometimes called spatial access methods, are data structures designed to support and optimize search operations on spatial data objects that are essential to spatial queries.
Widely used index methods in relational databases, such as B-tree [3, 2], extendible hashing [13] and linear hashing [24], are one dimensional access methods hence not suitable for spatial database simply because the search space is multidimensional.

1.1.1 Euclidean Space

R-tree [16], proposed by Antonin Guttman in 1984, is the mostly used indexing method to optimize spatial query in Euclidean Space.

In a R-tree, objects are represented by intervals in multiple dimensions. And the key idea of R-tree is to group nearby objects and index them using their minimum bounding rectangle. In this way, a search for objects is reduced to a search of intersection bounding rectangles. Intersection with a higher level bounding rectangle requires exploring of sub-tree, or, in the other hand, results in a pruned sub-tree.

Compared with traditional indexing methods, R-tree has several advantages to be applied in a spatial context.

- Unlike hash based indexing methods, which support exact match efficiently, R-tree supports range query in addition.
- Unlike one dimensional ordering of a single key value provided by B-tree, R-tree provides two dimensional ordering, and can be extended easily to higher dimensional space.
- Unlike Grid indexing, where cells shall be decided in advance, R-tree uses minimum bounding rectangles that can be updated dynamically.
- Unlike Quad tree and k-d tree, R-tree takes into account also I/O cost of secondary memory.

Due to the significant use in both theoretical and practical contexts, many variations have been proposed, trying to make it more efficient or trying to improve its worst case performance, such as R*-tree [15], R+-tree [40], Hilbert R-tree [18] and Priority R-tree [1].
Another interesting extension is Vor-Tree [41], which incorporates Voronoi diagram into R-tree. The hierarchical structure of R-tree allows search quickly arrive the right neighborhood. And the Voronoi diagram provides a tighter polygon shaped search region contrasts with coarser rectangular search region.

### 1.1.2 Spatial Network

Spatial network can be modeled as a graph, and therefore indices for graph can be used to index spatial networks (i.e. adjacency matrix and adjacency list). Since they are well known to the community, the details are omitted.

Although spatial network is gaining more serious attention in recent years, there is a lack of significant research into its indexing technologies.

Shortest path quadtree [36] is an interesting indexing method. It encodes all pair’s shortest path information based on an observation that the shortest path from a vertex to all remaining vertices can be grouped according to the first edges on the shortest paths to them. Shortest path quadtree captures this spatial coherence and reduces the storage requirement as only the first edge along the shortest paths are stored. A search for a shortest path over shortest path quadtree can thereby be reduced to recursively retrieving the first edge to the next vertex along the shortest path.

Shortest path quadtree takes advantage of the observation that shortest paths in road networks are spatially coherent. Apart from [36], [37, 39, 38] also take this advantage. Another important observation is that some vertices in a road network are more important than other vertices [14, 30].

Contraction Hierarchies (CH) [14] utilize the second observation and imposes a total ordering on the vertices according to their relative importance. [30] is an extension to CH.

### 1.2 Spatial Queries

Various query types were formulated and proposed to adapt to different occasions. This section briefly describes a few of them. And most of all, the closely related
query types (i.e. $k$ nearest neighbors query and reverse $k$ nearest neighbors query) are introduced with more detailed description.

1.2.1 $k$ Nearest Neighbors Query

Nearest neighbor query is one of the most fundamental research problem in the area of spatial database. Given a query point $q$, and a set of spatial objects, nearest neighbor query finds the closest object to the query point according to some distance metric. Similarly, $k$ nearest neighbors ($k$NN) query [32] finds the first $k$ closest objects to the query point. NN query is a special case of $k$NN query, where $k = 1$. More specifically, any object which is not a $k$NN of the query point, has distance to query greater than distance from any of the $k$NN to the query.

This problem is well studied, and includes several variants, such as continuous nearest neighbor query [26], aggregate nearest neighbor query, and path nearest neighbor query [10].

An example of $k$NN query could be “find out the 5 closest soldiers” issued by a nurse in a battle field. In this case, the distance is better measured by the Euclidean distance. Another example of $k$NN query could be “find out the 3 closest gas stations” asked by a car driver. In this case, the distance is better measured by the shortest road network distance.

Since $k$NN query are extensively studied, hence several variants emerged. Each variant targets a particular problem setting, and adds different requirements to the supporting indexing method.

**Snapshot $k$ Nearest Neighbor Query** Snapshot query targets static data set, where all the spatial objects as well as query point are static. This type of query generally does not update the underlying data structure or at least very rarely.

**Continuous $k$ Nearest Neighbor Query** Continuous $k$ nearest neighbor query report $k$NN for a query point over a time period. The problem arise when either the query point or other objects are constantly updating their location. The database is not static, and hence requires frequent updating. This type of query is most related to this thesis.

**Approximate $k$ Nearest Neighbor Query** Sometimes finding exact $k$NN is not necessary or even not feasible, especially when the data set is huge or in a data
stream where information can only be accessed once. In these case, many solutions to find the approximate $k$NN were proposed.

**Distributed Processing of $k$ Nearest Neighbor Query** Apart from a central server, this type of query take into account other computational modules as well. For example, if the computation and data storage can be distributed to smart phones, and utilize the CPU and storage unite on these mobile devices, the computational load on the server can be reduced, with a potential to reduce communication cost, which is also very expensive in such a system.

### 1.2.2 Reverse $k$ Nearest Neighbors Query

Reverse $k$ nearest neighbors ($Rk$NN) query is a very important variant of $k$NN query, and it is the topic of this thesis.

A $Rk$NN query finds out a set of objects, not necessarily $k$ objects, where the query point is $k$NN of any object in the set. This naturally captures the objects that influenced the most by the query point, hence have various applications in location based services include location based games, traffic monitoring, location based SMS advertising, market analysis, enhanced 911 services and army strategic planning etc. These applications may require continuous monitoring of reverse nearest moving clients.

![Figure 1.2: Reverse $k$ Nearest Neighbors Query](image)

As illustrated in Fig. 1.2, there are two gas stations, $f_1$ and $f_2$, and there are two cars, $c_1$ and $c_2$. Assume $f_2$ is the query point. The nearest neighbor of $f_2$ is $c_1$ since $c_1$ is closer to $f_2$ than $c_2$, but the reverse nearest neighbor of $f_2$ is $c_2$ since the nearest neighbor of $c_2$ is $f_2$, but the nearest neighbor of $c_1$ is not $f_2$.

Although $c_1$ is the nearest neighbor of $f_2$, but since the nearest gas station of $c_1$ is $f_1$ and the nearest gas station of $c_2$ is $f_2$, hence $f_2$ has a higher influence on
\( c_2 \) than \( c_1 \). Assume both \( c_1 \) and \( c_2 \) would like to go to their nearest gas station and refuel, then \( c_1 \) prefers \( f_1 \) and \( c_2 \) prefers \( f_2 \).

The owner of the gas station \( f_2 \) may want to continuously monitor its \( R_k \) cars and may decide to send them promotional offers. Such promotional offers are expected to influence car drivers more than promotional offers send to its \( k \) cars.

1.2.3 Other Query Types

Optimal Location Query Optimal location (OL) query is particularly useful for strategic planning of resources [51]. Given a set of facilities, a set of clients and a set of candidate locations, an optimal location query finds out the candidate location which optimize a certain cost metric. For example, a city government plans to build a new hospital (i.e. a new facility). Given the locations of all the existing hospitals (i.e. a set of facilities), the government may seek a candidate location that minimize distance from any resident (i.e. client) to its nearest hospital.

Top-\( k \) Spatial Keyword Query Given a set of spatial objects with textual description, a query point with search keywords, a top-\( k \) spatial keyword query seeks the highest ranked \( k \) objects in terms of both spatial and textual similarity of those objects with the query [31]. For example, a user may want to find out the nearest 5 restaurants that serve Italian or French foods. A good index to assist this kind of query may be a combination of R-tree and inverted index.

Maximizing Range Sum Query Given a set of weighted points, maximizing range sum query finds out a rectangular or circular location of given size such that the sum of the weights of all covered points is the maximum possible [12]. For example, a coffee chain holds some data about their members, including their home address and how often they would love to have a cup of coffee in store within a limited distance. Then the manager may want to issue a maximizing range sum query to find out the most profitable location to open a new site.
1.3 Contributions

With the popularity of cheap mobile devices, continuously monitoring of spatial queries over moving objects became an important issue to many location-based services and applications.

$R_k$NN query is of significant concern due to its natural capturing of the influence of the query point, hence can have many applications such as decision support, location based services, resource allocation, profile-based management, etc.

This thesis focuses on the problem of efficiently monitoring of reverse $k$ nearest neighbor query in spatial network. The contributions can be summarized as follows.

- We are the first to study the influence zone in road networks that have applications in $R_k$NN queries, marketing and decision making. We propose an efficient influence zone computation algorithm based on several novel observations and pruning rules.

- We demonstrate the usefulness of influence zone by developing an efficient algorithm to continuously monitor reverse $k$ nearest neighbor queries in spatial networks. Our techniques can be applied to both directed and undirected networks.

- We conduct extensive experimental study and demonstrate that our continuous $R_k$NN monitoring algorithm is more than an order of magnitude more efficient than the state-of-the-art algorithm.

1.4 Thesis Organization

The rest of this thesis is organized as follows.

- Chapter 2 summarizes a few most related works about snapshot $R_k$NN query and continuous $R_k$NN query in both Euclidean Space and Spatial Network.
Chapter 3 presents an overview of our solution, includes the motivation behind this thesis, a formal definition of the thesis problem (i.e. continuously monitoring R$k$NN query in spatial network), and followed by the outline of our solution.

Chapter 4 presents our influence zone computation algorithm that based on several novel observations and pruning rules and discusses the monitoring phase where the influence zone is to be used to efficiently monitor continuous R$k$NN query.

Chapter 5 provides an extensive experimental study using a real word road network with various settings of several parameters, and shows that the presented algorithm more than an order of magnitude more efficient than the state-of-the-art algorithm.

Chapter 6 concludes this thesis and explores remaining problems and potential future works.
Chapter 2

Related Work

This chapter examines related work on snapshot R\(k\)NN query and continuous monitoring of R\(k\)NN query in both Euclidean Space and Spatial Network. Influential works are introduced with a summary of their advantages and disadvantages, including the reason why they are not applicable for the problem studied through this thesis. Two representative works from each section are described in detail.

2.1 Snapshot R\(k\)NN Queries in Euclidean Space

RNN queries were first introduced by Korn et. al [22] who answer the RNN query by using preprocessing. Specifically, for each data object \(p\), they pre-calculate its nearest neighbor and assign a circle for \(p\) such that \(p\) is the center of the circle and the distance to its nearest neighbor is the radius. Then, RNN of a query \(q\) can be computed by returning every point that contains \(q\) in its circle. Yang et. al [52] and Lin et. al [23] present techniques to improve the work done in [22]. [22] is explained in detail in Section 2.1.1.

Stanoi et. al [42] handle R\(k\)NN queries by partitioning the space centered at the query point \(q\) into six regions of 60 degrees each. It can be proved that, for each partition, any client that is further from query than the \(k^{th}\) closest facility of \(q\) cannot be the R\(k\)NN of \(q\). They utilize this observation to prune the search space.

Cheema et. al [6, 8] introduce the concept of influence zone for answering
reverse $k$ nearest neighbors queries. To compute the R$k$NN queries, initially the influence zone is computed then each client that lies inside the influence zone is returned as answer. As stated earlier, their work is only applicable to Euclidean space and it is non-trivial to extend the techniques for road networks.

Sharifzadeh et. al [41] introduced an index structure VoR-tree to efficiently answer various spatial queries including reverse $k$ nearest neighbors queries.

Tao et. al [45] propose TPL that utilize the property of perpendicular bisectors to prune the search space. The perpendicular bisector between the query point $q$ and a pruner $p$ splits the space into two half-spaces. The half-space that contains $p$ is denoted as $H_{pq}$, every point in this half-space is closer to $p$ than $q$, and cannot be the RNN. Further, every point that lies in $k$ such half-spaces cannot be the R$k$NN. The limitation of this approach is that to prune the entries, TPL uses $m$ (i.e., the number of facility points for which the bisectors are considered) combinations of $k$ bisectors which is expensive.

Wu et. al [49] propose algorithm FINCH that overcomes the limitation of TPL. Instead of using bisectors to prune the objects, they use a convex polygon that approximates the unpruned area. Clearly, the containment checking is easier than TPL as this can be done in linear time via convex polygons. However, since FINCH approximates the unpruned area, it prunes an area smaller then the area that actually can be pruned. [49] is explained in detail in Section 2.1.2.

2.1.1 Pre-Calculation Based Approach

Korn et. al identify a problem in various marketing and decision support systems that to determine the influence of a query point on the database [22]. This notion is natural and broadly applicable. For example, to decide if a new store outlet should be opened at a spot (i.e. a query point), the manager may need to know how many residents will be influenced by a store outlet at the spot.

The set of points that are closest to the query point (i.e. the nearby residents of the potential new store outlet) may not be the set of points that have the query point as their nearest neighbor (i.e. the residents shall be influenced by the potential new store outlet). The set of points that are the RNN of a query point is defined as Influence Set. A two step approach of finding the influence set is proposed as
CHAPTER 2. RELATED WORK

follows.

1. For each data point \( p \), generate a circle \((p, \text{dist}(p, N(p)))\), where \( p \) is its center and \( \text{dist}(p, N(p)) \) is the distance from \( p \) to its nearest neighbor \( N(p) \) as the radius.

2. For any query point \( q \), determine all the circles that contain \( q \), return their centers.

The proposed approach also tolerates dynamic insertion and deletion of data points (note, dynamic does not mean data points are moving). A two step approach for insertion of a point \( q \) is as follows.

1. Determine all the RNN \( p \) of \( q \), for each such point \( p \), replace all the circles of the form \((p, \text{dist}(p, N(p)))\) to \((p, \text{dist}(p, q))\).

2. Find the nearest neighbor of \( q' \) of \( q \), and add a circle \((q, \text{dist}(q, N(q)))\).

Similarly, deletion of a point \( q \) can also be done in two steps, shown as follows.

1. Remove the circle \((q, \text{dist}(q, N(q)))\).

2. Determine all the RNN \( p \) of \( q \), for each such point \( p \), find its current nearest neighbor \( N(p) \), and replace its circle \((p, \text{dist}(p, q))\) to \((p, \text{dist}(p, N(p)))\).

Yang et. al [52] and Lin et. al [23] present techniques to improve [22].

2.1.2 FINCH

INCH and FINCH is proposed by Wu et. al in [49]. In contrast with [22], INCH and FINCH do not rely on pre-calculation.

Three steps based on the filter-refinement paradigm is required to process R\&NN query. INCH (i.e. INtersections’ Convex Hull) is the first step, which computes a search region where all candidates are inclosed. FINCH (i.e. Fast R\&NN processing using INCH) is the second step, where the search region is used to find candidates, and discovered candidates are used to gradually tighten
the search region until no candidate could be found. The last step is a refinement process to produce the final result from the set of candidates.

Given the query point \( q \) and a data point \( p \), for RNN query \((k = 1)\) the perpendicular bisector between them can prune objects in the half-plane that contains \( p \), since all such objects is closer to \( p \) than to \( q \). For RKNN query \((k > 1)\), a set of such bisectors can be used to prune objects in a certain area. This is the observation behind INCH and FINCH, and this observation is first introduced by Tao et. al in their proposed algorithm named TPL [45].

But the problem of the observation is that it is computationally expensive when \( k > 1 \). Instead, INCH calculate the convex hull of the region which can not be pruned (candidate region) in that way. The key advantage of using convex hull of candidate region is that the filtering process can be done with only one polygon other than many smaller polygons. Fig. 2.1 illustrates the candidate region (black line) and the search region (red line) for RKNN query when \( k = 2 \).

![Figure 2.1: Search Region and Candidate Region](image)

FINCH has two parts, namely FINCH-Filter and FINCH-Refine. FINCH-Filter computes a set of candidates, and FINCH-Refine refine the candidate set to produce the final result. The general case of FINCH-Filter algorithm is an iterative approach that starts with an empty set and gradually adds candidate objects to the set and shrinks the search region after each candidate object is identified. A more
efficient version of FINCH-Filter algorithm is based on R-tree index and follows
the best first search paradigm with respect to their distance from \( q \). FINCH-Refine
has two steps. The first step is efficient but approximate, and the second step is
computational expensive but precise.

## 2.2 Snapshot R\(k\)NN Queries in Spatial Networks

Spatial queries in spatial networks received significant research attention. Among
the various query types studied, the shortest path queries \([10, 36, 39]\), \(k\)NN queries
\([11, 21, 20]\) and range queries \([29, 25, 47]\) are among the most popular ones.

Papadias et al. [29] propose an architecture that integrates network with Eu-
clidean space. Based on this architecture, they introduce an approach to utilize Eu-
clidean restriction. They also developed a network expansion framework which
prunes search space efficiently.

Safar et al. [35] are the first to propose a solution for snapshot RNN queries
in spatial networks. They use a network Voronoi diagram to process RNN queries
efficiently. In their follow-up work [46], they extend their technique to process
R\(k\)NN queries in spatial networks. Taniar et al. [44] also study the RNN queries
problem in spatial network. They use order-2 network Voronoi diagram combined
with Euclidean distance based lower bound to provide an efficient solution. [35]
is explained in detail in Section 2.2.1.

### 2.2.1 Network Voronoi Diagram

Safar et al. solves RNN query with the aid of Network Voronoi Diagram (NVD)
in [35]. Their approach is based on properties of NVD, and their previous work,
namely Progressive Incremental Network Expansion (PINE) [33, 34], which fol-
lows the incremental network expansion (INE) [29] paradigm.

A Voronoi diagram divides a space into disjoint polygons (i.e. Voronoi cells),
where the generator of a polygon is the nearest neighbor of any point inside the
polygon. NVD is defined as graphs. It divides the network into disjoint sub-
networks. In a Voronoi diagram, Voronoi cells are of convex polygon shape, but
in contrast, Voronoi cells in NVD are of irregular shape. Fig. 2.2 illustrates an
example network Voronoi diagram.

![Network Voronoi Diagram](image)

Figure 2.2: Network Voronoi Diagram

Safar et. al classify RNN into four types depending on if the query point is generator or not and interest objects are generators or not. The algorithm is based on the existence of a NVD accompanied with pre-computed data, such as border-to-border and border-to-generator distances. The four types are listed as follows.

**$RNN_{GP}$**: Both query point and interest objects are generators.

In this case, the candidate interest objects are generators of query adjacent Voronoi cells.

**$RNN_{GP}$**: Only query point is generator.

In this case, the result is the set of interest objects that enclosed by the Voronoi cell where the generator is the query point.

**$RNN_{GP}$**: Both query point and interest objects are not generators.

In this case, the NVD helps reducing network distance calculation time. The idea is to explore the neighborhood of the query point using PINE expansion. And if an interest object is found, PINE expansion is used to process a NN query of the interest object to determine if it is the RNN of the query point.
**RNN\_GP(GP):** **Only interest objects are generators.**

In this case, the NVD helps reducing network distance calculation time. The idea is to explore the neighborhood of the query point using network expansion. Generator of the Voronoi cell where the query point belongs to, and the generator of neighboring Voronoi cells are examined.

### 2.3 Continuous R\_kNN Queries in Euclidean Space

Benetis *et. al* [4] are the first to present an algorithm for continuous RNN queries in Euclidean space. However, they assume that velocities of the objects are known. Xia *et. al* [50] do not assume any knowledge about objects’ movement and propose a solution based on the six region based approach. Kang *et. al* [19] propose an algorithm based on perpendicular bisectors based pruning approach for continuous monitoring. Wu *et. al* [48] propose the first solution to monitor R\_kNN queries based on the six region based approach. All of these algorithms monitor RNN or R\_kNN queries by first conducting the pruning phase to short list candidates, and then verifying those candidates to report the results. Whenever the query or a candidate object changes the location, the expensive pruning phase is needed to be reinvoked.

Cheema *et. al* [7] introduce the algorithm named *Lazy Updates* to continuously monitor R\_kNN queries. In contrast to the previous approaches, Lazy Updates saves computation time by reducing the number of times the pruning phase has to be invoked. This is achieved by assigning a rectangular region to each moving object such that the pruning remains valid as long as each object remains inside its respective rectangular region. They also proposed Influence Zone based algorithms [6, 8] to continuously monitor R\_kNN queries in Euclidean space. [7] is explained in detail in Section 2.3.1, [6] is explained in detail in Section 2.3.2.

#### 2.3.1 Safe Region and Lazy Updates

Cheema *et. al* introduce the algorithm *Lazy Updates* [7], which improves the computation cost and reduces the communication cost in client-server architecture.
by assigning each object and query a rectangular safe region such that location
updates may not be required as long as the query and objects remain in their
respective safe regions.

The server recommends the side lengths of the safe regions, and a client as-
signees itself a safe region such that it lies at the center of the safe region. If an
object stops moving (e.g. a car is parked), it notifies the server and the server
reduces its safe region to a point.

The communication between server and clients is classified into two types. A
client reports its location to the server when it moves out of its safe region (i.e.
source-initiated updates). In order to verify results, the server may need to request
the exact location of a client (i.e. server-initiated updates). Although the query is
also assigned with a safe region, it reports its location at every timestamp because
its location is important to verify results.

The algorithm has two phases, listed as follows.

1. **Initial computation:** When a query is issued, the server computes the set
   of candidate objects by applying pruning rules (i.e. filtering phase). Then,
   for each candidate object, the server verifies if the query is $k$NN of it (i.e.
   verification phase).

2. **Continuous monitoring:** The server maintains the set of candidate objects.
   Upon receiving location updates, the server updates the candidate set if it
   is affected. Otherwise, the server invokes verification module and report
   results.

   There are four pruning rules used in filtering phase, listed as follows.

**Half-space Pruning**

Fig. 2.3 illustrates a simpler case where the exact location of a filtering object (i.e.
an object that is used for pruning other objects) $p$ is known, and the exact location
of $q$ on a line $MN$ is unknown. The grey area is the pruned area, and the boundary
consists of every point $x$ that satisfies $\text{mindist}(x, MN) = \text{dist}(x, p)$.

The parabola between $L_M$ and $L_N$ can be approximated by a line $AB$ as il-
lustrated in Fig. 2.4(a). Alternatively, it can be approximated by moving $H_{p:N}$
CHAPTER 2. RELATED WORK

(i.e. half-space created by perpendicular bisector of \(p\) and \(N\)) and \(H_{p:M}\) to \(H'_{p:M}\) and \(H'_{p:N}\) respectively such that they both go through a point \(c\) that satisfies \(\text{mindist}(c, MN) \geq \text{dist}(c, p)\) as illustrated in Fig. 2.4(b). The moved half-space is called normalized half-space.

Fig. 2.5(a) illustrates the pruning area by half-space pruning. \(R_q\) is the safe region of query, and \(R_{fq}\) is the safe region of a filtering object. The four normal-
ized half-space is derived from four pairs of antipodal corners. For example, \( A \) and \( P \) are antipodal corners, \( A \) has the biggest \( x \) coordinate among corners of \( R_q \) while \( P \) in the other hand has the smallest \( x \) coordinate among corners of \( R_{fid} \). Similarly, \( A \) has the smallest \( y \) coordinate while \( P \) has the biggest \( y \) coordinate. The shaded area are pruned by half-space pruning.

![Diagram showing half-space pruning and dominance pruning](image)

**Figure 2.5: Half-space Pruning and Dominance Pruning.**

**Dominance Pruning**

Note in Fig. 2.5(a), the angle between the half-spaces that define the pruned area is greater than \( 90^\circ \). This is always true, and the space dominated by \( c \) (i.e. the dotted area) can be pruned.

In general, let \( f \) be the furthest corner of \( R_{fid} \) from \( R_q \) and \( n \) be the nearest corner of \( R_q \) from \( f \), as illustrated in Fig. 2.5(b), the frontier point \( F_p \) (i.e. the middle point of line segment \( fn \)) dominates and prunes the shaded area. Fig. 2.5(b) demonstrates four frontier points and the four corresponding pruned areas.

**Metric Based Pruning**

Let \( R_{end} \) be the safe region of a candidate object, it can be pruned if the maximum distance between \( R_{end} \) and \( R_{fid} \) is strictly smaller than the minimum distance
between $R_{\text{end}}$ and $R_q$.

Pruning if Exact Location of Query is Known

As illustrated in Fig. 2.6, any point $p$ that lies in the shaded area can be pruned, but $p'$, which is not in the shaded area, is the RNN of $q$ if the filtering object is exactly at corner $P$.

![Figure 2.6: Pruned area if exact location of query is known](image)

2.3.2 Influence Zone

Cheema et. al introduce the novel concept influence zone in [6], which is the area such that every point inside it is R$k$NN of $q$ and every point outside of it is not R$k$NN of $q$. This approach does not require a verification phase.

The main idea is using R-tree to index data points. Since data points that are close to the query $q$ are expected to prune larger area, data points and intermediate R-tree nodes are explored iteratively in increasing order of their minimum distance to $q$.

Initially, the influence zone is the whole data space, and a min-heap $h$ is initialized with the root of the R-tree. If a de-heaped entry $e$ completely lies outside of all convex vertices of the current influence zone, it is ignored. Otherwise, its
children are inserted in $h$ if $e$ is a R-tree node, or it is used to prune the space and update the influence zone if $e$ is a data point.

After the influence zone is obtained, the monitoring is equivalent with checking containment of data points in the influence zone.

![Figure 2.7: Grid Based Index of Influence Zone](image.png)

To assist the containment checking, the influence zone is indexed by a grid based data structure as illustrated by Fig. 2.7. A cell $c$ of the grid is marked as an interior cell (i.e. black cells) if it is completely resides in the influence zone. A cell $c'$ is marked as a border cell (i.e. grey cells) if it intersects with the boundary of the influence zone. For each border cell, they record the edges of the polygon (i.e. boundary of influence zone) that intersect it.

### 2.4 Continuous R$k$NN Queries in Spatial Networks

Sun et. al [43] are the first to study the continuous monitoring of RNN queries in spatial networks. A multi-way tree is created for each query that helps in defining the monitoring region, and only the updates in the monitoring region affect the results. [43] is explained in detail in Section 2.4.1.

Cheema et. al [9] are the first to present a continuous R$k$NN monitoring algorithm for moving objects and queries in spatial networks. They assign each
moving object a safe region such that the pruning remains valid as long as the objects remain in their respective safe regions. Although this avoids frequent calls to the pruning phase, unfortunately the algorithm needs to verify a client whenever it changes its location. The verification of a client is expensive because it requires determining whether the query is one of the $k$ closest facilities of the client or not. We utilize the concept of influence zone to cheaply conduct the verification. Specifically, a client is confirmed as R$k$NN if and only if it is inside the influence zone. [9] is explained in detail in Section 2.4.2.

### 2.4.1 Multi-way Tree

The multi-way tree [43] approach is introduced by Sun et. al. With the aid of PMR quad-tree [27, 28, 17] to index the network, a multi-way tree is stored and maintained for a query as its monitored area.

PMR quad-tree partitions the data space. It is constructed by inserting edges to intersected nodes. When splitting threshold is exceeded, a cell splits into four cells as a normal quad-tree. Fig. 2.8 illustrates an example where the splitting threshold is 5.

![PMR Quad-tree](image)

**Figure 2.8: PMR Quad-tree**

Fig. 2.9(a) illustrates an example of monitoring area in the network. Fig. 2.9(b) illustrates the corresponding multi-way tree.
CHAPTER 2. RELATED WORK

(а) Monitoring area in the network  (b) Monitoring area in the multi-way tree

Figure 2.9: Monitoring Area.

The multi-way tree is built while iterative expanding the network. First, the nodes $n_2$ and $n_4$ of the edge where $q_1$ resides are inserted. Then, adjacent nodes of $n_2$, which is closer to $q_1$ than $n_4$, are inserted, namely $n_1$, $n_3$ and $n_8$.

Object updates are handled in three categories: the incoming objects (i.e. objects move into the monitoring area from outside), the outgoing objects (i.e. objects move out of the monitoring area from inside), and the objects moving within the monitoring area.

Multi-way tree can also handle updates of edge weight, either increases or decreases.

Edge weight updating is processed before objects updating to guarantee correctness and save as much computation as possible. After these information are updated, invalid subtree is deleted, and the remaining multi-way tree is re-expanded from new leaf nodes.

2.4.2 Safe Region, Unpruned and Monitored Network

As an extension of[7], Cheema et. al use safe region to assist processing of continuous R$k$NN query in spatial network, where they assume both the query and objects are moving.

Each object and query is assigned a safe region, which is basically a segment
in the network, and could also be extended to contain more than one edges and segments. Moreover, no two safe regions can intersect each other.

Each object and query reports its location to the server when it leaves its safe region (i.e. source-initiated updates). In order to update results, the server might need to request the exact locations of some objects (i.e. server-initiated updates).

The proposed algorithm has two phases, listed as follows.

1. **Filtering**: The filtering phase is to retrieve a set of candidates. This is done by pruning edges and segments of the network that cannot contain any R$k$NN, and the remaining network, unpruned network, contains the set of candidates. The candidate set remains valid unless either the query or a candidate leaves its safe region or an object enters in the unpruned network.

2. **Verification**: With a valid unpruned network, the verification phase is invoked at each timestamp. For Every candidate object, the query $q$ is checked of its eligibility to be a $k$NN.

**Filtering Phase**

The filtering phase is to incrementally expand the network in a way similar to Dijkstra’s algorithm. The edges and segments that are explored during this process form the unpruned network. Six lemmas are used to prune unnecessary networks, listed as follows.

1. An object cannot be the R$k$NN if its shortest path to the query contains $k$ other objects.

2. An object cannot be the R$k$NN if its shortest path to the query $q$ contains a dead vertex. A dead vertex $v$ is a vertex such that there exist at least $k$ other objects $o$ have $dist(v, o) < dist(v, q)$, where $dist(a, b)$ is the network distance (i.e. shortest path) from $a$ to $b$.

3. A vertex is a dead vertex if its shortest path to the query contains a dead vertex.
CHAPTER 2. RELATED WORK

4. For an edge \((v_1, v_2)\) (\(v_1\) and \(v_2\) are the two end vertices of the edge) that contains at least \(2k\) objects and does not contain the query \(q\), the edge cannot contain any R\(k\)NN if both \(dist(v_1, q)\) and \(dist(v_2, q)\) are longer than the length of the edge.

5. Only the extreme objects of an edge \((v_1, v_2)\) can be the R\(k\)NN of a query if \(q\) does not resides on the edge. An extreme object \(o\) is an object such that at least one of the two segments \([o, v_1]\) and \([o, v_2]\) contains at most \(k - 1\) other objects.

6. Regardless of the number of queries in the system, an edge that does not contain any query has at most \(2k\) objects that can be the R\(k\)NNs of any of the queries.

_verification phase_

A straightforward but costly approach to verify a candidate is to issue a boolean range query on the network with range set to the distance (i.e. network distance, the length of the shortest path between two points) from the candidate to the query.

One way to avoid boolean range query is by applying the forth lemma. If a candidate resides on an edge \((v_1, v_2)\) such that the edge length is shorter than both \(dist(v_1, q)\) and \(dist(v_2, q)\), a counter that records the number of objects on the edge is maintained. Objects are verified only when the counter shows that there are at most \(k\) objects on the edge.

Another way to avoid time consuming boolean range query is based on the concept of monitored network. Once the monitored network for an object is computed, the verification becomes computationally cheap. The monitored network is computed during the filtering phase and remains valid as long as the unpruned network remains valid.

Monitored network of an object \(o\) is the part of the network such that for any point \(p\) that does not resides on it, minimum network distance from \(p\) to the safe region of \(o\) is greater than maximum \(dist(o, q)\). Fig. 2.10 illustrates an example of monitored network of object \(o_3\).

Since for any object \(o'\) that out of the monitored network of object \(o\), the e-
Equation $\text{dist}(o, o') > \text{dist}(o, q)$ holds, hence only objects resides in the monitored network need to be considered at the time of verifying the eligibility of $o$ to be R$k$NN of the query $q$. 

Figure 2.10: Monitored Network
Chapter 3

Overview

First, we present our motivation of this thesis in Section 3.1. Then, we formalize the problem in Section 3.2. And last, we present an outline of our solution to the problem in Section 3.3.

3.1 Motivation

Given a set of clients $C$ and a set of facilities $F$, a client $c \in C$ may find its nearby facilities by issuing a $k$ nearest neighbors ($k$NN) query that returns $k$ closest facilities of $c$. Consider the example of a car driver who may issue a $k$NN query to find his $k$ closest gas stations and then may choose one of these gas stations for refueling. In contrast to a $k$NN query, a reverse $k$ nearest neighbors ($Rk$NN) query returns every client $c \in C$ for which the query facility $q \in F$ is one of the $k$ closest facilities. Since $q$ is close to such clients, $q$ is said to have high influence on these clients, i.e., $q$ is likely to be used by such clients. Hence, the set of such clients is also called influence set of $q$ [22]. For instance, the owner of a gas station may issue a $Rk$NN query to find the cars for which his gas station is one of the $k$ closest gas stations. These car drivers are his potential customers and are likely to be influenced by any promotional offer sent to them.

Throughout this thesis, we use RNN queries to refer to $Rk$NN queries for which $k = 1$. The reverse $k$ nearest neighbors ($Rk$NN) query [22, 42, 52, 7, 6, 4, 50, 19] has received significant attention ever since it was introduced in [22].
R\(k\)NN queries have various applications such as decision support, location based services, resource allocation, profile-based management, etc. With the availability of inexpensive mobile devices, position locators and cheap wireless networks, location based services are gaining increasing popularity. An example of such location based services is zhiing\(^1\). Consider that a user needs a taxi and she sends her location to a taxi company’s dispatch center. The company notifies to a taxi for which she is the closest passenger (the taxi is a RNN of the user).

Other examples of location based services include location based games, traffic monitoring, location based SMS advertising, enhanced 911 services and army strategic planning etc. These applications may require continuous monitoring of reverse nearest moving clients. For instance, the owner of a gas station may want to continuously monitor its R\(k\)NN cars and may decide to send them promotional offers from time to time. In this thesis, we study the problem of continuously monitoring R\(k\)NN queries in a road network where the clients (e.g., cars) are continuously moving and the facilities (e.g., gas stations) are static.

The existing best known solution (Lazy Updates [9]) monitors the R\(k\)NN queries in two major steps. In *pruning* phase, pruning rules are used to prune the part of the road network such that any client in the pruned region cannot be the R\(k\)NN of the query. The clients that are in the unpruned network are called the candidate objects. In *monitoring* phase, the server conducts *verification* for each candidate object \(c\) to verify whether it is a R\(k\)NN or not. Specifically, a candidate object \(c\) is reported as a R\(k\)NN if and only if \(q\) is one of its \(k\) closest facilities.

At each time stamp (e.g., after every \(t\) time units), the server needs to conduct the verification for the clients that are inside the unpruned network and have changed their locations during the last time stamp. This is quite expensive because i) verifying a candidate \(c\) requires checking whether \(q\) is one of the \(k\) closest facilities of \(c\) or not and ii) the verification is required as soon as a candidate changes its location.

To address the above limitation, we use the novel concept of *influence zone* that we introduced in [6, 8] for answering R\(k\)NN queries in Euclidean space. The influence zone \(Z_k\) of a query \(q\) is the region such that \(q\) is one of the \(k\) closest facilities of a client \(c\) if and only if \(c\) is inside this region, i.e., \(c\) is a R\(k\)NN if and

\(^1\)http://www.zhiing.com/how.php
only if \( c \) is inside the influence zone. Our solution also has two major steps. In the first phase, we compute the influence zone \( Z_k \) of the query. Once the influence zone is computed, it is straightforward to monitor the R-kNNs of query by monitoring the clients that are inside the influence zone. Hence, in the second phase, we report every client that lies inside the influence zone. At each time stamp, the clients that move inside the influence zone are reported as new R-kNNs and the clients that move out of the influence zone are removed from the answer. This significantly reduces the computation cost because we only need to monitor the clients that enter or leave the influence zone. Our extensive experimental evaluation demonstrates that our influence zone based approach is more than an order of magnitude better than the state-of-the-art algorithm [9].

Although the focus of this thesis is on continuous R-kNN queries, we remark that the influence zone has applications also in marketing and decision support systems. Consider the example of a restaurant. Its influence zone may be used as market analysis as well as targeted marketing. For instance, the demographics of its influence zone may be used by the market researchers to analyse its business. The influence zone can also be used for marketing, e.g., advertising bill boards or posters may be placed in its influence zone because the people in this area are more likely to be influenced by the marketing. Similarly, the people in its influence zone may be sent SMS advertisement.

In our previous work [6, 8], we presented efficient algorithm to compute influence zone in Euclidean space. In this thesis, we address the applications that use road network distance as underlying distance metric and propose efficient techniques to compute the influence zone in road networks. Unfortunately, the existing algorithms for Euclidean space cannot be applied or extended for road networks due to inherently different characteristics of Euclidean space and road networks.

### 3.2 Problem Formulation

**Reverse k nearest neighbors (RkNN) query:** Given a set of points \( C \) representing client objects, a set of points \( F \) representing facility objects, and a query point \( q \in F \), a RkNN query retrieves every point \( c \in C \) such that \( q \) is one of the \( k \) closest points to \( c \).
closest facilities of \( c \). In this thesis, we focus on \( R_k \text{NN} \) queries considering the road network distance.

**Example 1** Consider the example of Fig. 3.1 that includes three facilities (shown as square) \( f_1, f_2 \) and \( q \) on a road network along with three clients (shown as triangles) \( c_1, c_2 \) and \( c_3 \). \( c_1 \) is the \( R \text{NN} \) (\( k = 1 \)) of the query \( q \) because \( q \) is its closest facility. On the other hand, \( c_2 \) and \( c_3 \) are not its \( R \text{NN} \) because the closest facility for both \( c_2 \) and \( c_3 \) is \( f_2 \). Similarly, \( R_2 \text{NNs} \) (\( k = 2 \)) of the query \( q \) include the clients \( c_1 \) and \( c_3 \). The client \( c_2 \) is not the \( R_2 \text{NN} \) because \( q \) is not one of its 2 closest facilities.

![Figure 3.1: Illustration of \( R_k \text{NN} \) queries](image)

**Snapshot \( R_k \text{NN} \) Queries vs Continuous \( R_k \text{NN} \) Queries:** In a snapshot \( R_k \text{NN} \) query, the results are to be computed only once. On the other hand, in a continuous \( R_k \text{NN} \) query the results of the query are to be continuously monitored as the location of underlying data objects (e.g., clients and/or facilities) change. In this thesis, we focus on continuous monitoring of \( R_k \text{NN} \) queries. Like many real world scenarios, we assume that the clients (e.g., cars, pedestrians etc.) are moving whereas the facilities (e.g., gas stations, restaurants etc.) are static. At each time stamp (i.e., after every \( t \) time units), the server receives a set of updates that contains the new locations of the clients that have moved during the last time stamp. Upon receiving these location updates, the server updates the results of the \( R_k \text{NN} \) queries accordingly.
3.3 Solution Overview

The key idea is to compute the influence zone for the query. The influence zone of a query $q$ is a part of the network such that every client $c$ that lies inside the zone is the R$k$NN of $q$ and every client $c$ that lies outside the zone is not its R$k$NN. Once the influence zone is computed, the problem is reduced to monitoring the clients that are inside the influence zone. We give a formal definition of the influence zone below.

**Definition 3.3.1** Given a spatial network $G$, a set of facilities $F = \{f_1, f_2, \cdots, f_n\}$ where $f_i$ denotes the location of the $i^{th}$ facility on $G$, and a query facility $q \in F$, the influence zone $Z_k$ is a sub network $(Z_k \subseteq G)$ such that for every point $p \in Z_k$, $q$ is one of the $k$ closest facilities of $p$, and for every point $p' \notin Z_k$, $q$ is not one of the $k$ closest facilities of $p'$.

**Example 2** In Fig. 3.2(a), the influence zone $Z_1$ (i.e., $k = 1$) is the part of the network shown using thick lines. Note that any point (e.g., $c_1$) that lies in the influence zone is the RNN of $q$ whereas any point (e.g., $c_2$ and $c_3$) that do not lie on the influence zone are not the RNNs of $q$. Fig. 3.2(b) illustrates the influence zone $Z_2$ (e.g., $k = 2$) using the thick lines. Since $c_1$ and $c_3$ lie inside the influence zone, these two clients are the R2NNs of $q$. On the other hand, $c_2$ is not the R2NN of $q$ because it lies outside the influence zone.

Note that the influence zone does not change as long as the locations of the facilities do not change. As stated earlier, in our problem settings, the facilities (e.g., gas stations) do not change their locations. Therefore, once the influence zone has been computed, it is not required to be updated.

Based on the concept of the influence zone, we develop our continuous R$k$NN monitoring algorithm. It consists of two phases namely influence zone calculation phase (Chapter 4) and monitoring phase (Section 4.6). As is obvious from the name, in the influence zone calculation phase, the influence zone of the query is computed. Since a client is the R$k$NN of $q$ if and only if it is inside the influence zone, once the influence zone has been computed, we only need to monitor the clients that lie inside the influence zone. In the monitoring phase, the algorithm
monitors the clients that are inside the influence zone. More specifically, whenever a client moves, the server checks whether it lies inside the influence zone or not. If it is inside the influence zone, it is reported as the R$k$NN.

In the next Chapter, we present techniques to efficiently calculate the influence zone. In Section 4.6, we demonstrate efficient techniques to determine whether a client lies inside the influence zone or not.
Chapter 4

Influence Zone Based Techniques

For the ease of presentation, we treat the query point \( q \) as a node. For example, in Fig. 4.1, we assume that \( q \) is a node and \( n_1 \) and \( n_3 \) are its adjacent nodes in the network. Note that this assumption is made only for the simplification of the presentation and is not a requirement for our algorithm to work correctly.

Before we present our solution, we first define some terms and notations in Section 4.1. We present problem characteristics in Section 4.2 followed by our techniques in Section 4.3 and Section 4.4. We present the extension of our techniques to directed graph in Section 4.5. We explain how to use the influence zone to efficiently monitor RkNN query in Section 4.6.

4.1 Terms and Notations

Table 4.1 defines the terms and notations used throughout this thesis.

**Inner Node**: A node that resides in the influence zone is called an *inner node*.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G )</td>
<td>spatial network</td>
</tr>
<tr>
<td>( n_i )</td>
<td>( i^{th} ) node</td>
</tr>
<tr>
<td>( E(n_1,n_2) )</td>
<td>An edge between nodes ( n_1 ) and ( n_2 )</td>
</tr>
<tr>
<td>( Z_k )</td>
<td>influence zone</td>
</tr>
<tr>
<td>( Dist(p_1,p_2) )</td>
<td>length of the shortest path between ( p_1 ) and ( p_2 )</td>
</tr>
<tr>
<td>( S[p_1,p_2] )</td>
<td>the part of an edge between ( p_1 ) and ( p_2 )</td>
</tr>
</tbody>
</table>
CHAPTER 4. INFLUENCE ZONE BASED TECHNIQUES

Outer Node: A node that resides outside of the influence zone is called an outer node.

Bounding Node: An outer node is called a bounding node if at least one of its adjacent nodes is an inner node.

Example 3 In Fig. 3.2(a) \((k = 1)\), \(n_1\) is an inner node because it lies in the influence zone. On the other hand, \(n_2\), \(n_3\) and \(n_4\) are the outer nodes. Since \(n_1\) is an inner node and is an adjacent node of \(n_2\), \(n_2\) is a bounding node. Similarly, \(n_3\) is also a bounding node whereas \(n_4\) is not a bounding node. In the example of Fig. 3.2(b) \((k = 2)\), \(n_1\), \(n_2\) and \(n_3\) are inner nodes whereas \(n_4\) is an outer node as well as a bounding node.

Inner Edge: An edge that completely lies inside the influence zone is called an inner edge, i.e., an edge \(E(n_1, n_2)\) is an inner edge if, for every point \(p \in E\), \(p \in Z_k\).

Outer Edge: An edge that completely lies outside the influence zone is called an outer edge, i.e., an edge \(E(n_1, n_2)\) is an outer edge if, for every point \(p \in E\), \(p \notin Z_k\).

Partial Edge: An edge that only partially lies inside the influence zone is called a partial edge, i.e., an edge \(E(n_1, n_2)\) is a partial edge if there exists \(p \in E\) such that \(p \in Z_k\) and there exists \(p' \in E\) such that \(p' \notin Z_k\).

Inner Segment: An inner segment is the part of a partial edge that lies in the influence zone. A segment between two points \(p_1\) and \(p_2\) is denoted as \(S[p_1, p_2]\).

Example 4 In Fig. 3.2(a), the edge \(E(n_1, n_2)\) is a partial edge and the edge \(E(n_3, n_4)\) is an outer edge. In Fig. 3.2(b), the edge \(E(n_1, n_2)\) is an inner edge whereas the edge \(E(n_3, n_4)\) is a partial edge. In Fig. 3.2(b), the segment \(S[n_3, f_2]\) is an inner segment.

Note that an influence zone can be described by a set of inner nodes, a set of inner edges and a set of inner segments.

4.2 Properties

In this section, we present two properties that support our basic algorithm.
CHAPTER 4. INFLUENCE ZONE BASED TECHNIQUES

Lemma 4.2.1 A node \( n \) is an inner node iff the \( k \) closest facilities of \( n \) include \( q \).

The proof is straightforward hence omitted.

Lemma 4.2.2 An edge \( E(n_1, n_2) \) is an outer edge if both \( n_1 \) and \( n_2 \) are the outer nodes\(^1\).

Proof We prove this by contradiction. Assume a point \( p \in E \) that lies inside the influence zone, i.e., the \( k \) closest facilities of \( p \) include \( q \). Without lose of generality, assume that the shortest path from \( p \) to \( q \) goes through the node \( n_1 \), i.e., \( \text{Dist}(p, q) = \text{Dist}(p, n_1) + \text{Dist}(n_1, q) \). Consider a facility \( f \) such that \( \text{Dist}(n_1, f) < \text{Dist}(n_1, q) \). For such facility \( f \), \( \text{Dist}(p, f) \leq \text{Dist}(p, n_1) + \text{Dist}(n_1, f) \). Since \( \text{Dist}(n_1, f) < \text{Dist}(n_1, q) \), \( \text{Dist}(p, f) < (\text{Dist}(p, n_1) + \text{Dist}(n_1, q) = \text{Dist}(p, q)) \). In other words, \( f \) is closer to \( p \) than \( q \). Since \( n_1 \) is an outer node, there are at least \( k \) such facilities and this implies that there are at least \( k \) facilities closer to \( p \) than \( q \). Hence, \( p \) does not lie inside the influence zone which is a contradiction to our assumption.

Example 5 In Fig. 3.2(a), both \( n_2 \) and \( n_3 \) are the outer nodes and the edge \( E(n_2, n_3) \) is an outer edge.

Remark 1 A reader may incorrectly assume that an edge \( E(n_1, n_2) \) is always an inner edge if both \( n_1 \) and \( n_2 \) are the inner nodes. However, this is not true. For instance, in Fig. 3.2(b), \( n_2 \) and \( n_3 \) both are inner nodes but \( E(n_2, n_3) \) is not an inner edge (it is a partial edge).

As aforementioned, influence zone consists of inner edges and inner segments. Therefore, to calculate influence zone, all the inner edges and inner segments must be identified. According to Lemma 4.2.2, an edge between two outer nodes is always an outer edge. Hence, to compute the inner edges and inner segments, we only need to identify inner nodes and bounding nodes (Section 4.3). Based on the inner and bounding nodes, we can then determine the inner edges and the inner segments (Section 4.4).

\(^1\)Recall that, for the ease of presentation, we treat the query point \( q \) as a node in the network. Hence, if a query \( q \) lies on an edge \( E(n_1, n_2) \), the edge is considered to be divided into two edges \( E(n_1, q) \) and \( E(q, n_2) \) and the lemma is applied on both of the edges.
4.3 Identifying Inner Nodes and Bounding Nodes

A naïve approach to identify inner nodes is to apply Lemma 4.2.1 for all the nodes in the spatial network. However, this is quite expensive because the lemma is to be applied for every node in the network. Next, we present a lemma which allows us to identify all the inner nodes by applying Lemma 4.2.1 only on the inner nodes and the bounding nodes.

**Lemma 4.3.1** A node \( n \) is an outer node if the shortest path from \( n \) to \( q \) goes through another outer node \( n' \).

**Proof** As the shortest path from \( n \) to \( q \) goes through \( n' \), \( \text{Dist}(n, q) = \text{Dist}(n, n') + \text{Dist}(n', q) \). Since \( n' \) is an outer node, we know that there are at least \( k \) facilities such that for each such facility \( f \), \( \text{Dist}(n', f) < \text{Dist}(n', q) \). For each of these \( k \) facilities \( f \), \( \text{Dist}(n, f) \leq \text{Dist}(n, n') + \text{Dist}(n', f) \). Since \( \text{Dist}(n', f) < \text{Dist}(n', q) \), \( \text{Dist}(n, f) < (\text{Dist}(n, n') + \text{Dist}(n', q) = \text{Dist}(n, q)) \). Thus, each of these \( k \) facilities \( f \) satisfies \( \text{Dist}(n, f) < \text{Dist}(n, q) \). Hence \( q \) is not one of the \( k \) closest facilities of \( n \), i.e., \( n \) is an outer node.

We remark that, in the case where there are multiple shortest paths from \( n \) to \( q \), Lemma 4.3.1 holds as long as at least one of these shortest paths goes through an outer node.

**Example 6** In Fig. 3.2(a), \( n_3 \) is an outer node. The shortest path from \( n_4 \) to \( q \) goes through \( n_3 \). Therefore, \( n_4 \) is also an outer node (as can be verified because \( f_2 \) is also closer to \( n_4 \) than \( q \)).

We explore the network starting from \( q \) using network expansion similar to Dijkstra’s algorithm. As per Lemma 4.3.1, as soon as an outer node is encountered, further expansion from this node is not required. Algorithm 1 presents the details.

Algorithm 1 explores the network starting from \( q \) in an iterative expansion manner. Line 1 initializes a min heap \( h \) by inserting \( q \) with key set to zero. Recall that we treat \( q \) as a node in the network. The key of each node \( n \) in the heap is its network distance to \( q \). For each de-heaped node \( n \), we mark \( n \) as an inner
Algorithm 1 Inner Nodes and Bounding Nodes

**Input:** $G$, a set of facilities $F$, a query point $q \in F$, $k$

**Output:** a set of inner nodes, a set of bounding nodes

1. insert $q$ in a min-heap $h$
2. while $h$ is not empty do
3. deheap an node $n$, mark $n$ as visited
4. if $k$ closest facilities of $n$ include $q$ then
5. mark $n$ as inner node and mark it visited
6. for each unvisited adjacent node $n'$ of $n$ do
7. if $n'$ not in $h$ then
8. insert $n'$ in $h$
9. else if a shorter path from $n'$ to $q$ found then
10. update $n'$ in $h$
11. else
12. mark $n$ as bounding node and mark it visited

node if $q$ is one of its $k$ closest facilities (lines 4 and 5). Furthermore, for each adjacent node $n'$ of $n$, the algorithm inserts (or updates) $n'$ in $h$ along with its minimum network distance to $q$ (lines 6 to 10). On the other hand, $n$ is marked as a bounding node if $q$ is not one of the $k$ closest facilities of $n$ (line 12). According to Lemma 4.3.1, the network is not required to be expanded beyond any outer (or bounding) node. Therefore, the adjacent nodes of $n$ are not inserted in $h$ in this case. The algorithm terminates when the heap becomes empty. It can be easily verified that the algorithm identifies (and marks) all the inner and bounding nodes before it terminates.

A straightforward way to check if the $k$ closest facilities of $n$ include $q$ (line 4) is to explore the network starting from the node $n$ with a search area bounded by $Dist(n, q)$ and counting the number of facilities encountered during the search. We remark that a more efficient approach is to utilize the Euclidean distance bound to quickly conduct this check in some cases. More specifically, a Euclidean space circular range query [5] centered at $n$ with range set as $Dist(n, q)$ can be issued. If the Euclidean range query returns less than $k$ facilities, it guarantees that $q$ is one of the $k$ closest facilities of $n$ in terms of network distance. This is because Euclidean distance serves as a lower bound network distance and if there are less than $k$ facilities with Euclidean distance from $n$ smaller than $Dist(n, q)$ then there
are less than \( k \) facilities closer to \( n \) than \( q \) in terms of network distance. Euclidean range query can be efficiently conducted using an index structure such as R-tree if available.

### 4.4 Calculating Inner Edges and Inner Segments

In this section, we demonstrate how to compute inner edges and inner segments once the inner and bounding nodes have been identified using Algorithm 1. To facilitate the presentation, we first introduce the concept of monotonic edges.

**Monotonic Edge**: An edge \( E(n_1, n_2) \) is called a monotonic edge if the shortest path from \( n_2 \) to \( q \) goes through \( n_1 \) or the shortest path from \( n_1 \) to \( q \) goes through \( n_2 \). A monotonic edge \( E(n_1, n_2) \) is denoted as \( E_m(n_2, n_1) \) if the shortest path from \( n_2 \) to \( q \) goes through \( n_1 \), and it is denoted as \( E_m(n_1, n_2) \) if the shortest path from \( n_1 \) to \( q \) goes through \( n_2 \).

**Example 7** Fig. 4.1 shows two monotonic edges \( E_m(n_2, n_1) \) and \( E_m(n_4, n_3) \). \( E_n(n_2, n_3) \) and \( E_n(n_2, n_4) \) are non-monotonic edges.

Next, we present techniques on how to compute inner edge and inner segment for a monotonic edge. Later in Section 4.4.2, we show that a non-monotonic edge
can be easily broken into two monotonic edges and the techniques for monotonic edges can then be applied on each of the two monotonic edges.

### 4.4.1 Handling Monotonic Edges

Before we present our techniques, we present a few terms and two properties of monotonic edges.

**Pruner:** A facility $f$ is said to prune a point $p$ if $\text{Dist}(p, f) < \text{Dist}(p, q)$ and $f$ is called a pruner of $p$. Note that a point $p$ is the R$k$NN of a query $q$ if and only if there exist less than $k$ pruners of $p$. In other words, a point $p$ that has less than $k$ pruners lies in the influence zone, and a point $p'$ that has at least $k$ pruners lies outside the influence zone.

**Example 8** Consider the example of Fig. 4.1, $f$ is a pruner of $n_3$. Furthermore, $n_3$ is not a RNN of $q$ because it has one pruner.

**Lemma 4.4.1** Given a monotonic edge $E_m(n_2, n_1)$, a facility $f$ that prunes $n_1$ also prunes every point $p \in E_m(n_2, n_1)$.

**Proof** Assume that there is a point $p \in E_m(n_2, n_1)$ such that $f$ is not a pruner of $p$, i.e., $\text{Dist}(p, f) \geq \text{Dist}(p, q)$. Since $f$ is a pruner of $n_1$, we know that $\text{Dist}(n_1, f) < \text{Dist}(n_1, q)$. Furthermore, since $E_m(n_2, n_1)$ is a monotonic edge, the shortest path from $p$ to $q$ goes through $n_1$. This means $\text{Dist}(p, q) = \text{Dist}(p, n_1) + \text{Dist}(n_1, q)$. Since $\text{Dist}(n_1, f) < \text{Dist}(n_1, q)$, $\text{Dist}(p, q) > (\text{Dist}(p, n_1) + \text{Dist}(n_1, f))$. This contradicts the assumption that $\text{Dist}(p, f) \geq \text{Dist}(p, q)$.

**Example 9** Consider the monotonic edge $E_m(n_4, n_3)$ in Fig. 4.1, $f$ is a pruner of $n_3$ and it can be verified that $f$ is also a pruner of every point on $E_m(n_4, n_3)$.

**Lemma 4.4.2** Given a monotonic edge $E_m(n_2, n_1)$, a facility $f$ that does not prune $n_2$ cannot prune any point $p \in E_m(n_2, n_1)$.

**Proof** Assume that $f$ does not prune $n_2$ but prunes a point $p \in E_m(n_2, n_1)$, i.e., $\text{Dist}(p, f) < \text{Dist}(p, q)$. Since $E_m(n_2, n_1)$ is a monotonic edge, $\text{Dist}(n_2, q) \geq$
\( Dist(n_2, n_1) \). Since \( f \) does not prune \( n_2 \) (i.e., \( Dist(n_2, f) \geq Dist(n_2, q) \)), it is immediate that \( Dist(n_2, f) \geq Dist(n_2, n_1) \). This implies that if \( f \) lies on the edge \( E \), it lies at the node \( n_1 \) otherwise it lies outside the edge. Hence, the shortest path from \( p \) to \( f \) must go through either \( n_1 \) or \( n_2 \).

If the shortest path from \( p \) to \( f \) goes through \( n_1 \) then \( Dist(p, f) \geq Dist(n_2, f) - Dist(n_2, p) \). This is because \( Dist(n_2, f) \leq Dist(n_2, p) + Dist(p, f) \). On the other hand, if the shortest path from \( p \) to \( f \) goes through \( n_2 \) then \( Dist(p, f) \geq Dist(p, n_2) + Dist(n_2, f) \). Therefore, in any case, \( Dist(p, f) \geq Dist(n_2, f) - Dist(n_2, p) \). Since \( Dist(n_2, f) \geq Dist(n_2, q) \) (because \( f \) does not prune \( n_2 \)), \( Dist(p, f) \geq Dist(n_2, q) - Dist(n_2, p) \). Since \( E_{m}(n_2, n_1) \) is a monotonic edge, \( Dist(p, q) = Dist(n_2, q) - Dist(n_2, p) \). Hence, \( Dist(p, f) \geq Dist(p, q) \) which contradicts the assumption.

Example 10 Consider the example of the monotonic edge \( E_{m}(n_2, n_1) \) in Fig. 4.1. Since \( f \) does not prune \( n_2 \), it can be verified that \( f \) does not prune any point on \( E_{m}(n_2, n_1) \).

Based on the above lemmas, next we show how to compute inner edge and inner segment for a monotonic edge.

**Identifying inner edge.** Recall that an inner edge is an edge that completely resides in the influence zone. The next lemma shows that a monotonic edge is an inner edge if and only if its nodes are inner nodes.

**Lemma 4.4.3** A monotonic edge \( E_{m}(n_2, n_1) \) is an inner edge iff both \( n_1 \) and \( n_2 \) are inner nodes.

**Proof** Assume that both \( n_1 \) and \( n_2 \) are inner nodes. Since \( n_2 \) is an inner node, there are less than \( k \) facilities that prune \( n_2 \), i.e., at least \( |F| - k \) facilities cannot prune \( n_2 \) where \( |F| \) is the total number of facilities. According to Lemma 4.4.2, a facility \( f \) that does not prune \( n_2 \) cannot prune any point \( p \in E_{m}(n_2, n_1) \). Hence, there are at least \( |F| - k \) facilities that do not prune any point \( p \in E_{m}(n_2, n_1) \). Hence, every such point \( p \) is inside the influence zone which implies that the edge is an inner edge.

If at least one of \( n_1 \) or \( n_2 \) is not an inner node, then the edge is not an inner edge (it is a partial edge because at least one point is outside the influence zone).
Lemma 4.4.3 can be applied to identify whether a monotonic edge is an inner edge or not. Next, we show how to compute the inner segment of a monotonic edge if the edge is not an inner edge (i.e., is a partial edge).

**Identifying inner segments.** Recall that an inner segment is the part of a partial edge that lies in the influence zone. For instance, in Fig. 4.2, \( S[n_1, p_1] \) is an inner segment. Below, we show how to identify inner segments.

![Figure 4.2: Influence zone \( Z_2 \) \( (k = 2) \)](image)

Given an edge \( E_m(n_2, n_1) \), for each facility \( f \) that prunes at least one point of this edge, we first identify the segment of the edge that is pruned by \( f \). As implied by Lemma 4.4.2, a facility \( f \) that prunes at least one point of a monotonic edge \( E_m(n_2, n_1) \) must always prune \( n_2 \). Therefore, for each facility, the pruned segment always includes \( n_2 \), i.e., the pruned segment is \( S[p, n_2] \) where \( p \) is a point on the edge. Once we identify all segments that are pruned by the facilities, the inner segment of the edge corresponds to the part of the edge that is pruned by less than \( k \) facilities. The example below illustrates this.

**Example 11** Fig. 4.3 illustrates how to compute the inner segment of \( E_m(n_2, n_1) \) for the example of Fig. 4.2. For each of the facilities \( f_1, f_2 \) and \( f_3 \), it demonstrates the segment (shown thick) pruned by the corresponding facility. For instance, \( f_1 \) prunes the whole edge whereas \( f_2 \) prunes the part of the edge between \( p_1 \) and \( n_2 \). Note that the segment \( S[n_1, p_1] \) is pruned by less than 2 facilities so it lies inside
CHAPTER 4. INFLUENCE ZONE BASED TECHNIQUES

Figure 4.3: Inner segment is the segment pruned by less than $k$ facilities.

the influence zone and is the inner segment (considering $k = 2$). Note that such point $p_1$ can be easily identified given the pruned segments for each facility.

Next, we show how to identify the segment of a monotonic edge $E_m(n_2, n_1)$ pruned by a facility $f$. The facility prunes the whole edge if it prunes $n_1$ (Lemma 4.4.1) and it does not prune any point of the edge if it does not prune $n_2$ (Lemma 4.4.2). For the case when $f$ prunes $n_2$ but does not prune $n_1$, it prunes a segment $S[p, n_2]$ of the edge where $p$ is a point on the edge that lies at a distance $d$ from $n_2$ and $d$ is calculated by Equation (4.1).

$$d = \begin{cases} & \frac{\text{Dist}(n_2, q) + \text{Dist}(n_2, f)}{2} \quad \text{if } f \text{ is on } E_m(n_2, n_1) \\ & \frac{\text{Dist}(n_2, q) - \text{Dist}(n_2, f)}{2} \quad \text{if } f \text{ is not on } E_m(n_2, n_1) \end{cases} \quad (4.1)$$

The proof is omitted for brevity, however, it can be verified easily.

Example 12 In Fig. 4.2, the segment pruned by $f_2$ is $S[p_1, n_2]$ where $p_1$ is a point with distance $d = \frac{1}{2}(\text{Dist}(n_2, q) + \text{Dist}(n_2, f_2)) = \frac{1}{2}(8 + 2) = 5$ from $n_2$. The segment pruned by $f_3$ is $S[p_2, n_2]$ where $p_2$ is a point with distance $d = \frac{1}{2}(\text{Dist}(n_2, q) - \text{Dist}(n_2, f_3)) = \frac{1}{2}(8 - 6) = 1$ from $n_2$.

A final issue to be addressed with the above approach is that it requires computing the pruned segment for every facility $f$ regardless of whether it prunes a part of the edge or not. This can be easily addressed as follows. As per Lemma 4.4.2, if a facility $f$ does not prune $n_2$, it cannot prune any part of the edge. Hence, the pruned segment is required to be computed only for the facilities that
prune $n_2$. Therefore, we retrieve and store the facilities that prune $n_2$ when we check whether $n_2$ is an inner node or not (at line 4 of Algorithm 1). These facilities are then used to determine the inner segment as described above.

4.4.2 Handling Non-monotonic Edges

We handle a non-monotonic edge $E_n(n_1, n_2)$ by introducing an imaginary node $x$ on this edge that divides it into two monotonic edges $E_m(x, n_1)$ and $E_m(x, n_2)$. Then, the techniques presented in previous section can be applied on each of these monotonic edges to compute the corresponding inner segments.

![Figure 4.4: Handling non-monotonic edge.](image)

Example 13 Fig. 4.4 illustrates an example of a non-monotonic edge $E_n(n_1, n_2)$. $\text{Dist}(n_1, q) = 6$ and $\text{Dist}(n_2, q) = 7$. It also demonstrates a point $x$ on the edge such that $E_m(x, n_1)$ is a monotonic edge, i.e., the shortest path from $x$ to $q$ goes through $n_1$. Similarly, $E_m(x, n_2)$ is also a monotonic edge.

The point $x$ is a point such that the length of its shortest path to $q$ passing through $n_1$ is the same as the length of its shortest path to $q$ passing through $n_2$ (i.e., $\text{Dist}(x, q) = \text{Dist}(x, n_1) + \text{Dist}(n_1, q) = \text{Dist}(x, n_2) + \text{Dist}(n_2, q)$). Note, $\text{Dist}(x, q) = \frac{1}{2}(\text{Dist}(n_1, n_2) + \text{Dist}(n_1, q) + \text{Dist}(n_2, q))$. It can be verified that the point $x$ is a point that lies at distance $d$ from $n_2$ where $d$ is calculated as follows:
\[ d = \text{Dist}(x, q) - \text{Dist}(n_2, q) \]
\[ = \frac{\text{Dist}(n_1, n_2) + \text{Dist}(n_1, q) - \text{Dist}(n_2, q)}{2} \]  
(4.2)

**Example 14** In Fig. 4.4, \( x \) is a point that lies at distance \( d = (5 + 6 - 7)/2 = 2 \) from \( n_2 \).

A final issue that remains to be addressed is that, in order to efficiently compute inner segment of a monotonic edge \( E_m(x, n_1) \), we need to know the facilities that prune \( x \) and \( n_1 \). As stated earlier, the facilities that prune a node \( n_1 \) are obtained during the execution of Algorithm 1. However, since \( x \) is an imaginary node, the facilities that prune \( x \) are not obtained by Algorithm 1. A straightforward way to obtain these facilities is to explore the network starting at \( x \) with range set as \( \text{Dist}(x, q) \) and recording the facilities that are closer to \( x \) than \( q \). However, this is not efficient. Below we describe a more efficient approach.

We calculate pruners of \( x \) separately for facilities on the edge \( E_n(n_1, n_2) \) and facilities that are not on the edge. First, if a facility \( f \) is on the edge, then \( \text{Dist}(x, f) \) can be easily obtained. Second, if a facility \( f' \) is not on the edge, then the shortest path from \( x \) to \( f' \) goes through either \( n_1 \) (i.e., \( \text{Dist}(x, f') = \text{Dist}(x, n_1) + \text{Dist}(n_1, f') \)) or \( n_2 \) (i.e., \( \text{Dist}(x, f') = \text{Dist}(x, n_2) + \text{Dist}(n_2, f') \)), which ever is shorter. Since for \( \text{Dist}(x, f') < (\text{Dist}(x, q) = \text{Dist}(x, n_1) + \text{Dist}(n_1, q) = \text{Dist}(x, n_2) + \text{Dist}(n_2, q)) \), we must have either \( \text{Dist}(n_1, f') < \text{Dist}(n_1, q) \) or \( \text{Dist}(n_2, f') < \text{Dist}(n_2, q) \), hence we check the pruners of both \( n_1 \) and \( n_2 \). For each of such facilities, once the length of the shortest path from \( x \) is calculated, we compare it with \( \text{Dist}(x, q) \) to know if the facility prunes \( x \) or not.

### 4.5 Extension to Directed Graph

Previous sections focus on \( R_k \)NN queries in spatial networks that are represented by undirected graphs. Our proposed techniques can be easily extended for directed graphs. In this section, we show the changes needed for \( R_k \)NN queries in directed graphs.
Since it can be verified that lemmas 4.2.1, 4.2.2 and 4.3.1 still hold, Algorithm 1 can be used in a similar way. The only difference is that for unidirectional edges the expansion follows only the reverse direction (i.e., at line 6 of Algorithm 1, a node $n'$ is considered the adjacent node of $n$ if and only if the edge is either bidirectional or the direction of the edge is from $n'$ to $n$).

Lemmas 4.4.1, 4.4.2 and 4.4.3 hold for bidirectional edges. Once inner and bounding nodes are identified, bidirectional edges can be handled the same as in an undirected graph. Although these lemmas are not applicable for unidirectional edges, this issue can be easily addressed as below.

Let $E_u(n_1, n_2)$ be an unidirectional edge goes from $n_1$ to $n_2$. The shortest path from $n_1$ to a facility $f$ or $q$ may not go through $n_2$. Apart from $n_1$, the shortest path from every other point $p \in E_u(n_1, n_2)$ to a facility $f \notin S[p, n_2]$ or $q$ must go through $n_2$. This means, on one hand, a facility $f \in E_u(n_1, n_2)$ prunes a segment $S[n_1, f]$ (note, $n_1$ is excluded), and on the other hand, a facility $f' \notin E_u(n_1, n_2)$ such that $\text{Dist}(n_2, f') < \text{Dist}(n_2, q)$ prunes every point $p \in E_u(n_1, n_2)$ (note, $n_1$ is included). Recall that the inner segment can be easily identified given the pruned segments for each facility.

![Figure 4.5: A simple directed spatial network example.](image)

**Example 15** Fig. 4.5 gives a simple example on directed spatial network. Direction of edges are marked by arrows. For example, bidirectional edge $E(n_1, n_3)$
has arrows at both ends, and unidirectional edge $E_u(n_2, n_1)$ has arrow pointing at $n_1$, which means it goes from $n_2$ to $n_1$.

Since $n_1$ has no pruner, $E_u(n_2, n_1)$ (n_2 excluded) has no pruner. $f_1 \in E_u(n_2, n_4)$ prunes $S[n_2, f_1]$, and since $n_2$ is pruned by $f_1$, $E_u(n_3, n_2)$ (n_3 excluded) is pruned as well. Similarly, $f_2 \in E_u(n_4, n_3)$ prunes $S[n_4, f_2]$, and also prunes the whole edge $E_u(n_2, n_4)$. Fig. 4.5(a) shows the influence zone when $k = 1$, and Fig. 4.5(b) shows the influence zone when $k = 2$.

### 4.6 Monitoring Phase

Once the influence zone is computed, we only need to monitor the clients that lie in the influence zone. In real world applications, there may be multiple R$k$NN queries issued by different users. In this section, we show how to efficiently monitor the results of all R$k$NN queries.

For each edge $E$ of the network, we maintain two lists. The first list called edge list contains query ID of every query $q$ for which $E$ is an inner edge. The second list called segment list contains query ID of every $q$ for which $E$ contains an inner segment (the inner segment is also stored as shown in Fig. 4.6).

Whenever a client moves from a location $loc_{old}$ to $loc_{new}$, we update the results of the affected queries as follows. Without loss of generality assume that $loc_{old}$ (resp. $loc_{new}$) lies on an edge $E_{old}$ (resp. $E_{new}$). Note that the results of every query $q$ are changed if the client leaves or enters its influence zone. Note that if a client leaves (resp. enters) an influence zone, its $loc_{old}$ (resp. $loc_{new}$) must be inside its influence zone. Hence, we look in the edge list and segment list of $E_{old}$ and identify every query $q$ for which the client $c$ was in its influence zone, i.e., $q$ is present in the edge list of $E_{old}$ or $q$ is present in the segment list of $E_{old}$ and $c$ lies in the inner segment. For each such query $q$, the client $c$ is removed from its
R$k$NN. Then, we look in the edge list and inner segment list of $E_{new}$ and identify the queries for which $c$ is now in its influence zone. For each such query $q$, $c$ is added as a R$k$NN.
Chapter 5

Experiments

In this section, we evaluate the performance of our algorithm for \( R_k \)NN queries. We first illustrate the size of influence zone in section 5.2, and then evaluate the total running time of our approach in section 5.3, and finally we evaluate the computation cost of the influence zone calculation phase in section 5.4.

5.1 Experimental Settings

All the experiments were conducted on Intel Xeon 2.4 GHz dual CPU with 4 GBytes memory. All the algorithms (including the competitors) were implemented in C++. Our algorithm is named \textbf{IZone}. We compare our algorithm with the state-of-the-art algorithm for \( R_k \)NN monitoring in spatial networks, Lazy Updates which is also known as \textbf{SAC} [9].

We use the road network of California that consists of 21048 nodes, 21693 edges and 87635 points of interest. For the experiment where we vary the number of facilities, we randomly select facilities from the 87635 points of interest. The moving clients follow a random moving path, e.g., a client moves on an edge with a given speed and when it reaches the end node, it randomly selects a neighboring edge and continues moving on it.

Table 5.1 shows the system parameters used in our experiments, default values are shown in bold. The server reports the results continuously at each time stamp. We assume that the time stamp length is 1 second. All queries are continuously
Table 5.1: System Parameters for Experiments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of clients (×1000)</td>
<td>50, 75, <strong>100</strong>, 125, 150</td>
</tr>
<tr>
<td>Percentage of clients moving</td>
<td>20, 40, 60, 80, <strong>100</strong></td>
</tr>
<tr>
<td>Speed of clients (km/h)</td>
<td>60, 70, <strong>80</strong>, 90, 100, 110, 120</td>
</tr>
<tr>
<td>Number of queries</td>
<td>1, 10, <strong>100</strong>, 1000</td>
</tr>
<tr>
<td>Number of facilities (×1000)</td>
<td>2, 4, <strong>8</strong>, 16, 32, 64</td>
</tr>
<tr>
<td>( k )</td>
<td>1, 2, 4, 8, <strong>16</strong>, 32, 64</td>
</tr>
</tbody>
</table>

monitored for 5 minutes (e.g., 300 time stamps) and the results correspond to the total running time of all queries monitored for the 5 minutes duration.

5.2 Influence Zone Size

In this section, we illustrate how the size of influence zone is affected by different system parameters. The size of the influence zone is important because the total running cost is directly proportional to the size of the influence zone (e.g., the larger the influence zone is, the greater the number of R\( k \)NNs of \( q \) is). The size of the influence zone is the total length of all the inner edges and inner segments.

![Figure 5.1: Influence zone size.](image)

**Figure 5.1: Influence zone size.**

Fig. 5.1(a) shows the effect of the number of facilities on the size of influence zone. The influence zone becomes smaller when the number of facilities increases. This is because the distance from a client to its \( k \) closest facilities is inversely proportional to the number of facilities.
Fig. 5.1(b) shows the effect of $k$ on the size of influence zone. As expected, the size of influence zone increases with the value of $k$. This is because, for a larger value of $k$, a client is influenced by facilities located farther.

### 5.3 Running Time

In this section, we illustrate how the total running time is affected by various system parameters. Here, the total running time includes the CPU time of influence zone calculation phase and monitoring phase.

Fig. 5.2(a) shows the effect of number of facilities on the total running time. When the number of facilities is bigger, the computational cost is lower. This is because, as shown in Fig. 5.1(a), when there are more facilities on the network, the influence zone is relatively smaller. Hence, the running time is inversely proportional to the number of facilities because the influence zone calculation time as well as the monitoring time is reduced. Also, note that IZone is more than an order of magnitude faster than SAC.

![Graph](image-url)

(a) Effect of number of facilities.

![Graph](image-url)

(b) Effect of $k$.

Figure 5.2: Total running time.

As noted in Fig. 5.2(a), the scenario with fewer number of facilities is more challenging for both of the algorithms. Therefore, for the rest of our experiments, we choose a relatively smaller number of facilities, i.e., we set 8000 as the default number of facilities.

Fig. 5.2(b) shows the effect of $k$. The running times of both algorithms increase with the value of $k$. However, IZone is significantly more efficient and
scales much better.

Fig. 5.3(a) shows the effect of number of queries on both of the algorithms. As with other experiments, IZone is more than an order of magnitude faster and scales better.\footnote{The cost does not grow exponentially}

Fig. 5.3(b) shows the effect of number of moving clients. The running times of both algorithm increases with the increase in number of clients, however, IZone scales much better.

Fig. 5.4(a) shows the effect of the proportion of clients (in percentage) that are moving, i.e., $m\%$ means $m$ out of 100 clients are moving and the others are static.
CHAPTER 5. EXPERIMENTS

This figure shows similarity with Fig. 5.3(b), this is because when a client does not move, it does not report to the server about its location and location updates, hence the server does not need to calculate any thing for these clients. From this perspective, when a larger number of clients are moving, it is equivalent to a larger number of clients present in the network.

Fig. 5.4(b) shows the effect of the speed of moving clients. The performance of the algorithms is not affected by the speed of moving clients and IZone consistently outperforms SAC.

5.4 Calculating Influence Zone

Figure 5.5: Influence zone calculation phase CPU time cost.

In this section, we demonstrate only the cost of first phase, i.e., influence zone computation time for IZone and pruning phase cost of SAC. We remark that the unpruned network is a superset of influence zone, i.e., it includes every points of the influence zone. We also remark that computing the influence zone is a more complicated task than computing the unpruned network due to the following reason. An unpruned network is a sub-network such that every client $c$ that lies outside it can be pruned, however, every client $c'$ that lies inside it may or may not be pruned. In contrast, influence zone is a sub-network such that every client $c$ that lies outside it can be pruned and every client $c'$ that lies inside it cannot be
pruned. This additional feature of influence zone that significantly reduces the total running time makes the computational task of the first phase more challenging.

Fig. 5.5 shows the costs of first phase for both the algorithms for increasing number of facilities and increasing value of $k$. As stated earlier, influence zone poses more complicated requirements, which implies that additional checks are required to compute the influence zone. Nevertheless, Fig. 5.5 shows that the cost of influence zone computation is almost the same as the cost of computing unpruned network. This indicates that our proposed technique computes a more complex region by incurring almost the same cost as incurred by SAC.
Chapter 6

Concluding Remark

6.1 Conclusion

In this thesis, we study the problem of continuous monitoring of reverse $k$ nearest neighbors (R$k$NN) queries in spatial networks. R$k$NN query have received significant research attention in the past few years due to its wide range of applications, and the continuous monitoring problem raise when inexpensive mobile devices are widely used and popular.

We are the first to use the novel concept of influence zone in spatial networks which has many applications such as in location based services, marketing and decision support systems. Since a client is the R$k$NN of the query point if and only if it is inside the influence zone, once the influence zone has been computed, we only need to monitor the clients that lie inside the influence zone, this significantly improves the efficiency of monitoring.

Based on several non-trivial observations, we present an efficient algorithm to compute the influence zone for both directed and undirected graphs. Our extensive experimental study demonstrates that our algorithm is more than an order of magnitude faster than the state-of-the-art R$k$NN monitoring algorithm.
6.2 Future Work

In this thesis, we assume that the network is static, whilst in some real world cases it may change dynamically. For example, the expected travelling time required on roads is generally different during daytime and nighttime. In this case, the weight of edges (i.e. expected time required to go through) in the network is of interest, and when the condition changes, the influence zone becomes invalid and requires re-calculation. How to avoid such expensive re-calculation could be an interesting research topic.
Bibliography


