Efficient Algorithms for Spatial Queries over Static and Mobile Objects on Road Networks

(道路網上の静的及び動的オブジェクトに対する
空間検索の為の高効率アルゴリズム)

2018 年 9 月

埼玉大学大学院理工学研究科（博士後期課程)

理工学専攻（主指導教員 大澤 裕）

TIN NILAR WIN

Efficient Algorithms for Spatial Queries over Static and Mobile Objects on Road Networks

TIN NILAR WIN (15DM056)

A Thesis Submitted to the

Graduate School of Science and Engineering

Saitama University

in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF ENGINEERING

in

Mathematics, Electronics and Informatics

Supervisor: Professor Yutaka Ohsawa

Saitama University

JAPAN

2018 - September

Acknowledgment

First and foremost I express special and heartily thanks to my supervisor, Professor Yutaka Oshawa, an excellent mentor, for his kindness and giving me the opportunity to carry out my thesis in his department. I have learned many things from him since I became his student under his supervision. His advice on both research as well as on my career have been invaluable.

I am deeply thankful to my committee members, Prof. Tetsuya Shimamura, Prof. Atsushi Uchida, Prof. Takashi Komuro and Prof. Yoshinori Kobayashi for accepting as committee members and investing time and providing interesting and invaluable comments and suggestions throughout the study period.

My sincere thanks also go to Assist. Prof. Htoo Htoo for sharing her knowledge on meetings, and providing all required information. She supported not only with plenty of advice for research but also academically and emotionally to make the study smooth.

I also want to take a moment to thank Saitama University for giving me a chance to study here. In addition, I am grateful to the Japanese Government (MONBUKAGAKUSHO:MEXT) scholarship foundation for providing me a generous financial support to carry out the full-time study in Saitama University. In particular, I want to thank all the members of Graduate School of Science and Engineering for providing several advice and help during my study.

I would like to acknowledge the Department of Science and Engineering at

Saitama University for making it possible for me to study here. I extend my deep thanks to the Professors and lecturers at the Department of Science and Engineering, and other members of the Department.

I want to thank the International Office members for taking responsibility our financial works and providing timely and efficient support during studying at Saitama University. I am grateful and indebted to my seniors, friends and laboratory members for sharing knowledge and giving me valuable assistance.

I also thank my family who encouraged me and prayed for me throughout the time of my life. Without their support, it was not possible to carry out study in a foreign country. Furthermore, I am thankful to sister Aye Thida Hlaing for encouraging the further studies in Saitama University, Japan.

In this lines, I would like to thank special person of my life who always beside me in every difficulties. I am very grateful to my soul mate and my best friend Nirmal for his patience, giving me constant motivation, encouragement, cheering me up to complete my studies and for his love and big heart helped me overcome all the obstacles. He spent several hours for me especially during my thesis writing to help me correction and offer valuable advice. I dedicate this thesis to him and my family for their constant support and unconditional love.

Finally, I am grateful to the readers for reading my thesis and I hope the content of the thesis will be useful for enhancing knowledge in the field of computational science.

Abstract

Spatial database is one of the active area in database research communities since last three decades, addressing the need of spatial data management and analysis in application such as Geographic Information Systems (GIS). Due to wide availability of relatively inexpensive and compact location position devices such as hand-held Geographical Positioning Systems (GPS) devices and smart-phone, users want an access to the real-time personal information anywhere in the world. To supply the end users with required information, a service provider must collect and process the geolocation information. This service is called location-based services (LBS). LBS has recently attracted significant attention due to its potential to revolutionize the mobile communications technologies by providing a personalized and context-aware services to bring unique and broad value to the end users.

There are several applications using LBS such as route assistance, tracking management, identification of objects of interest, fleet management, content management and emergency services. One of the active challenges in LBS is personalization which supports more customized and explicit services according to the users' unpredictable actions and preferences. To fulfill these requirements, we need effective and robust querying methodologies to process geographic information (so called spatial information) to utilize in the real location aware applications.

To accomplish the above challenges, we studied spatial query algorithms and analyzed the performance efficiencies under various settings. More specifically, we focused for two types of queries: 1) query evaluates immediately after the query is

invoked and the answer is transmitted to the user once which is called a snapshot query, and 2) query evaluates every time instant without user interference to ensure the correctness and validity of the query answer which is called a continuous query. The major challenges for these queries are focused on methods to provide an efficient query processing for either stationary or continuously moving objects with respect to the processing time, CPU time, and main memory utilization.

In this thesis, we first study the snapshot query over the static data objects and query objects on several road networks. We use object of interest and data object (point) interchangeably throughout the thesis. We present an efficient solution to answer the reverse k nearest neighbor ($RkNN$) query using pre-computation approach. we utilize the simple materialized path view ($SMPV$) structure that can calculate the road network distance with speedy performance and incremental Euclidean restriction (IER) framework to probe the fast k nearest neighbor (kNN) query. $SMPV$ structure is constructed by partitioning the whole data space into multiple subgraphs and creating a materialized distance tables for each subgraph. An extensive experimental study was conducted to evaluate the performance characteristics of our proposal for static $RkNN$ query with some road networks showed that it substantially outperforms the competitive approach that do not use any precomputed distance tables. When we used $SMPV$ framework as the underlying structure, our algorithm outperformed in processing time that was 10 to 100 times faster than competitive approach. The key characteristic of our approach is that when the data objects are distributed sparsely on the road network and the arbitrary k value ($k > 1$ more than one RNN objects) is large, our approach reduced the overall processing time drastically. Especially, we observed on the bichromatic $RkNN$ query with this approach that our approach does not depend on any distribution of rival objects ($\in S$) and objects of interest ($\in P$) and the processing time was stable.

As an expanded studies of spatial queries which evaluate continuously according to the mobile query objects have been studied in this thesis. Suppose, the query

object moves freely, the query result changes over time according to the changes of query object locations which make the query processing more complicated than the processing with the static query objects. In addition, while the location of query object is changing, the frequent requests for the up-to-date result to the server occur that cause heavy workload in the server. To address this problem efficiently, an idea of safe-region is utilized. Safe-region method restricts the mobile query object to issue a query update only if mobile object leaves the region. We present how to process the spatial continuous queries effectively in two chapters (4 and 5) in this thesis. In chapter 4, we describe the continuous vicinity queries with the aid of advanced technique called safe-region. The existing approaches for the continuous queries using safe-region require long processing time to generate safe-regions. In addition, these approaches are not applicable for all types of spatial queries, thus, require different safe-region generation mechanism for different spatial queries. To overcome all these limitation, we provide a set of techniques that can generate the safe-region with less processing time and is applicable for various types of vicinity queries. According to the conducted experimental study, the results confirm the efficiency of our proposed methodologies. The detail theoretical explanation and performance study are presented in Chapter 4.

The benefits of safe-region approach are maintaining the valid query answers for mobile query object if object is in safe-region and reducing the communication cost between the server and client (query object). This kind of approach is suitable for a kind of trip route planning query ($TRPQ$) that visits multiple data object categories and processes continuously. Generally, $TRPQ$ requires huge processing time to retrieve the best trip route for snapshot query type. In addition, if a query object (user) is mobile, the user may sometimes deviate from the optimal route while traveling. To cope with these, generally, we can monitor and update the route either at a specified time interval or distance. However, this traditional approach is not applicable for all situation, has many restrictions on the distance and time interval, and there is deterioration in server performance as the interval

is reduced. For such cases, we adopt the idea of safe-region and verify the route is optimal if the position of the current object of targeted category is in the safe-region although the user veers from the optimal route. Moreover, safe-region approach has less restrictions compared with the traditional ways.

As a first attempt, we proposed a continuous $TRPQ$ on road network using safe-region, and we investigated the generation of safe-region for $TRPQ$ with two approaches. The first approach creates an accurate safe-region and the second approach generates an approximate safe-region. We analyzed the performance of our proposed approaches and basic algorithm in the experimental study section. Our proposed first algorithm needed long processing time when the density of the data object was high. Our second approach required less processing time. However, the size of the generated safe-region by second approach became about 3% to 7% larger than the safe-region size of first approach.

In addition, we addressed the solution to solve the processing time problem of the trip route query when the distribution of densities of data object categories are substantially different for each other. For this issue, we introduced a new method which defines the sparse data category as the first visiting data category to shorten the processing time. Because the processing time has a great effect on the visiting order of the data object categories. Our proposed approach dynamically retrieves the sparse category to define the visiting order of the data categories. The theoretical explanation and performance study are presented in chapter 5.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Abbreviations

$A*$	A Star
$AVOS$	Auxiliary Vicinity Object Set
$BBDT$	Border to Border Distance Table
$BRkNN$	Bichromatic Reverse k Nearest Neighbor
CS	Closed Set
$CTRPQ$	Continuous Trip Route Planning Query
FM	Full Materialized
GB	GigaByte
GIS	Geographic Information Systems
GPS	Geographical Positioning Systems
H	Heap
$CTRPQ$	Continuous Trip Route Planning Query
$IBDT$	Inner to Border Distance Table
IER	Incremental Euclidean Restriction
INE	Incremental Network Expansion
kNN	k Nearest Neighbor

kVD	k^{th} order Voronoi Diagram
LBS	Location-Based Services
LBC	Lower Bound Constraint
LM	Light Materialized
$LORD$	Light Optimal Route Discoverer
LPR	Length of Partial Route
MB	MegaByte
MBR	Minimum Bounding Rectangle
MM	Medium Materialized
MO	Moving Object
MPV	Materialized Path View
$MRkNN$	Monochromatic Reverse k Nearest Neighbor
$MRPSR$	Multi Rule Partial Sequence Route
$MTNN$	Multi Type Nearest Neighbor
NN	Nearest Neighbor
$NNDT$	Node to Node Distance Table
NVD	Network Voronoi Diagram
OSR	Optimal Sequence Route
$OSRA$	Optimal Sequence Route with A*
$PINE$	Progressive Incremental Network Expansion
PNE	Progressive Neighbor Exploration

PR	Partial Route
PRA	Preceding Rival Addition
QR	Query Result
RER	Range Euclidean Restriction
$RkNN$	Reverse k Nearest Neighbor
RNN	Reverse Nearest Neighbor
RNE	Range Network Expansion
RO	Rival Object
ROS	Rival Object Set
$SCFA$	Sparse Category First Algorithm
$SDBMS$	Spatial Database Management System
SG	Sub-Graph
$SMPV$	Simple Materialized Path View
SPF	Shortest Path Finder
SR	Safe-Region
$SSMTA*$	Single Source Multi Target A*
TP	Time Parameterized
TPQ	Trip Planning Query
TRA	Tardy Rival Addition
$TRPQ$	Trip Route Planning Query
VD	Voronoi Diagram

VP	Voronoi Polygon
VR	Voronoi Region

CHAPTER 1

Introduction

In computing world, a database is a system to collect related information that permits the entry, storage, output and processing of data. Among the various types of database, our study in this thesis is focused on spatial database. Unlike the ordinary database, a spatial database or geo-database is a database that is especially built to store and access the spatial data representing a geometric space. The spatial database systems are aiming to serve the underlying database technology for geographic information systems and other applications. The main applications that are driving research in spatial database systems are the applications of Geographical Information System (GIS). A spatial database provides a reliable foundation for accessing, storing, managing and querying the spatial data.

A spatial database can handle complex spatial data such as higher dimensional objects including time reference. Due to the rapid increase in the availability of data from a wide variety of sources such as satellite images, mapping agencies and independent data collection agencies, we have witnessed in recent year an increase in the demand for systems that can model, manipulate, and interpret the spatial data. In addition, spatial information can represent the current status of real objects.

Therefore, the study about the spatial objects is increasingly popular in research areas of computer science. The spatial data is associated with geographical locations and features such as points, lines, polygons, and surfaces, volumes. A common example of spatial data is a street map showing roads and popular locations. A real road map is composed of several two-dimensional objects such as points, lines, and polygons, that represent cities, roads and regions respectively. Such database technology makes possible to provide location-based services to the web and mobile applications.

In a common database system such as relational database management system, objects are represented by tuples with some attributes and indexed by ordering one of the attribute. Indexes are used for more efficient search and providing relationship. However, traditional database systems have not been designed to support frequent update due to object agility, predicative and spatio-temporal based query processing. Spatial data is logically represented by a road network data model and stored in a road network data structure. Spatial database uses a unique index called a spatial index to speed up database performance. Spatial indexing enables the system to retrieve data from a large collection of objects without searching the whole database. The detail explanation of the spatial indexing is described in Chapter 2. In addition, spatial database requires a special kind of data type to model the geometric structure encountered in spatial objects and to provide the object's relationship and properties with respect to its operations in a spatial environment. Thus, the spatial data types have become an important part of data model in spatial database management system.

In the beginning, spatial objects were usually assumed as static in the spatial database. With the advance in positioning technologies such as mobile communication devices, the spatial temporal database came into existence which can store attributes of objects that change with respect to time. It is now possible to track continuously moving objects. Objects whose location change in time need special handling from the database system. To handle temporal spatial data, using

the traditional approach to retrieve and identify the locations and relationship of spatial objects, high update rates must be sustained by the database system. To address this problem, we present the various continuous spatial queries are related to moving objects in this thesis.

1.1 Querying Spatial Objects

Efficient processing of spatial queries is an important part to implement the spatial database. Various spatial queries have been formulated to address end user preferences. A spatial query must fulfill the requirements of the real scenarios [42] viz: 1) be able to incorporate the network connectivity data and provide exact distances between objects, 2) returns the answer efficiently to support the query objectives, 3) be applicable to the large networks, and 4) be independent of the distribution of the data objects in road map. To accomplish these objectives, generally, we can utilize two ways such as pre-computation and on-the-fly approaches to process spatial queries. We can classify mainly two types of the spatial query according to the characteristics of objects; these are static objects or moving objects.

1.1.1 Querying Spatial Object on Static Environment

In a static environment, the data objects on the road network are stationary and location changes do not arise. The spatial query that runs on a static environment executes once and contains only the results which met the specified criteria at the time the query was created. Numerous spatial queries on static environments including NN query, RNN query, range query have been well studied in the last 30 years resulting in the development of numerous conceptual models and query processing techniques. The detail theoretical explanation of each query will be presented in the next section. Every spatial query type has a counterpart in spatial database. The fundamental concept for solving the spatial query problems in spatial database is capturing and processing the spatial objects and the underlying network

including both the Cartesian coordinate and spatial network [26]. The early research works for spatial processing of stationary objects were based on Euclidean distance metrics.

The spatial data processing techniques especially for spatial road network must take into account real-life constraints such as unidirectional road, unsymmetrical road network which the existence of one-way roads and obstructions or delays affecting only one direction of a two-way. The shortest road distance between objects (e.g., user and restaurant) is not only decided by considering the objects locations, but, also on other constraints such as the connectivity of the road network. This is explained by comparing in Euclidean and road network distance in Figure 1.1. In the figure, the black rectangle represents the hospital and red dot represents the query user, the numerical values attached on the straight dashed line are Euclidean distance and the numerical values on the road segments are road network distance. When a user searches the hospitals within $10km$ range, the result will contain A, B, and C accordingly to the road network distance. If we consider Euclidean distance, A, B, C, and D are the result. Note that, the nearest hospital is B in road network. However, the nearest hospital in Euclidean distance is D which is the furthest hospital from query object in road network distance. To perform the query processing on the road network, the road network distance calculation between two objects is essential and the distance computation on the road network is expensive. Therefore, the spatial data processing in the spatial network becomes a challenge in spatial database research community.

To achieve the spatial query processing, we can attempt using structure based, and non-structure based techniques. Structure based technique uses the index structure such as R-Tree [4], R*-Tree [8], Quad-Tree [6], and Voronoi diagram (VD) [14]. Indexing is mainly designed to speed up objects retrieval since objects are usually assumed to be constant unless explicitly updated. There is also a different approach that is based on pre-computation of the solution space or the pre-computation of distance from query object q to its closest objects of interest such

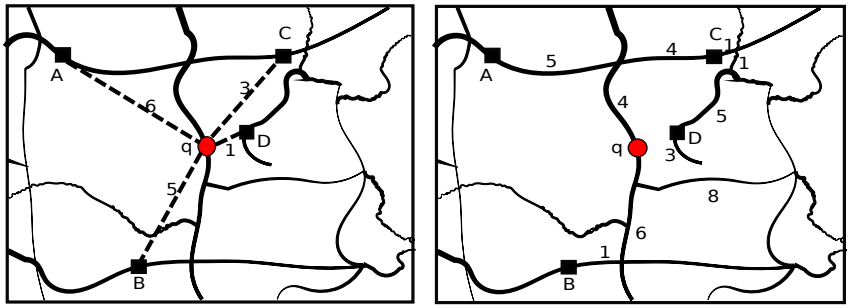


Figure 1.1: Distance Comparison of (a) Euclidean distance on a road map and (b) road network distance on a road map.

as materialized path view (MPV). However, when we use these approach, we need storage/computation-time tradeoffs algorithms because pre-computation approach suffers from the problem of requiring a huge memory space. We discuss and study such kind of tradeoffs methodology called simple materialized path view ($SMPV$) index structure in chapter 3.

1.1.2 Querying Spatial Object on Dynamic Environment

In dynamic environment, the data objects or query objects move and hence frequent update is required. Queries made by a moving object need frequent updates simultaneously to provide the latest query result. The strategies targeted for spatial queries on static objects are not efficient for dynamic environments since the results may be invalidated as soon as the objects move. To address such situation, we need a query algorithm that can monitor the moving objects and maintain the correctness of the query result continuously. As opposed to snapshot spatial queries that are evaluated only once to return a single result, continuous queries require constantly evaluate and update the results according to the position of objects. Such queries are especially relevant to spatio-temporal databases, which are inherently dynamic, and the result of any query is strongly related to the temporal context [27].

An example of a continuous spatio-temporal query is asking question such as: “ based on user current location and speed of moving object, what will be the

two nearest gas stations from user's current location for the next 5 minutes?". Continuous queries can be categorized into three groups based on the mobility of the query and the data objects [52] viz. 1) static query objects querying moving data objects, 2) moving query objects querying moving data objects, and 3) moving query objects querying static data objects. The bulk of the research for the distinct types of spatial queries on the dynamic environment have been carried out since last two decades using various techniques for the moving objects database where data or queries (or both) objects move. The related research works are presented in the chapter 2.

Generally, we described some techniques to accomplish the query processing in above section. The tree based indexing techniques are not well suited for the moving objects and incur a large latency to access data if index structure is not inefficient. Therefore, we need an innovative methodologies to process query on moving objects efficiently. In chapter 4 and 5, we propose some mechanisms for the continuous spatial queries.

1.1.3 Spatial Queries Proposed in this Thesis

This section discusses the various spatial queries such as range query, k nearest neighbor query, reverse k nearest neighbor query, and trip route query which are studied with our proposed index structure and framework in this thesis. In this thesis, we have considered spatial queries on both static querying and continuous querying the objects.

Range Query

Range query arises frequently in spatial applications such as tracking the number of nearest taxis within $1km$ distance. Range query selects the objects that lie within a specified range (e.g. $1km$). Range query can also be useful for other queries such as nearest neighbor query. It provides as preprocessing tools by retrieving the data

objects within the specified range for reducing the accessing amount of data. In the static environment, the range query can probe the data objects easily. In a dynamic environment, given a collection of moving objects and a moving query, the range query retrieves the objects that lie within the specified region defined by the query object and updates synchronously the latest result when the query object moves. As the query object moves, the position and outline of the query region also change; hence, the methodology of the range query especially for moving environment should be accomplished by considering a dynamic shaped region approach instead of employing the traditional range queries such as rectangular range (window query). The query achieves the result based on R-Tree structure and constructs the rectangular range around the query objects and returns the data objects that lie within the given range. Tree based methodology especially R-Tree is slow in update and the query efficiency declines while processing query continuously. An efficient way to process continuously range query is explained in chapter 4.

k Nearest Neighbor query

k nearest neighbor (kNN) query is an extention of nearest neighbor (NN) query. NN query finds a closest object from the query location and focuses the search of potential neighbors only. kNN query retrieves the arbitrary k nearest objects such that no other objects are nearer to the query object. As an example, kNN query initiated by a location aware devices such as car navigation or smart-phone navigation to find the 3 (k) nearest cafes from the user's current location. kNN query is more complicated compared to the range query. The kNN query can be classified according to the characteristics of objects as 1). kNN query that retrieves results at a time instant called snapshot kNN and 2). kNN query that evaluates the result continuously during a time interval.

The original and most influential kNN searching algorithms utilizes the tree based index structure and Voronoi diagram. Although these approaches are capable to retrieve kNN on static objects, applying these approaches directly for the moving

objects suffers performance related problem. To address this, we present a better approach for kNN query to process continuously in chapter 4.

Reverse k Nearest Neighbor query

Another related variant of the nearest neighbor (NN) queries is reverse nearest neighbor query. Reverse nearest neighbor (RNN) query retrieves all the data objects that consider query object (q) as one of their closest points. RNN can also be extended into reverse k nearest neighbor ($RkNN$) query that probes all the data objects for which q as their k nearest points. For example, imagine, a player looking for the players from an opponent team who are considering him as their nearest player among others. As a next example, a business owner planning to open new convenient store may ask "Where is the best location for the new store to attract many customers?". We can rephrase this question as "How many customers will consider this location as their nearest?". Then each candidate locations invokes $RkNN$ query and makes a comparison with the existing store locations. The study of RNN and $RkNN$ queries have received considerable attentions due to their usefulness in several applications involving location management system, decision support system and emergency (such as 911).

$RkNN$ queries are generally classified into two types: monochromatic $RkNN$ ($MRkNN$) and bichromatic $RkNN$ ($BRkNN$) queries. In $MRkNN$, the query and the data objects are the same data type. In the above two examples, the former one (the player example) is the example of the $MRkNN$ query. Unlike the $MRkNN$ queries, in $BRkNN$ queries, the data objects and query objects belong to two different types of objects. The latter example (the business planner example) is able to deal with $BRkNN$ queries in which the data objects (customers) and query (convenient stores) objects belong to two distinct types. Several related studies for $RkNN$ queries are described in chapter 2 and our strategies to enhance the $RkNN$ queries for static and continuous querying are described in chapter 3 and 4 respectively.

Trip Route Planning Query

One more interesting spatial query called trip route planning query ($TRPQ$) has been evolving to cater road navigation system. Due to the rapid increase in the number of car navigation system and smart phone devices, the use of route assistance and direction support application are increasing rapidly in daily activities. For example, a tourist wants to go a museum from a hotel in an unfamiliar city with a car. In addition, during trip from hotel to the museum, he/she wants to visit a restaurant and a gas station. In such situation, he/she needs the best route (i.e. shortest distance) with more than one stopovers. This type of query is called trip route planning query. TRP query, given a source s and destination d, retrieves the shortest route from s to d passing through one object of interest from each related category. In above example, the gas station and the restaurant are visited with any order i.e. the visiting order is not specified while traveling between the hotel and museum. In general, there could be many restaurants and gas stations in the neighboring of the trip route. Therefore, TRP query needs to retrieve a sequence of objects of interest from each category that gives the shortest trip route length. The detail study about TRP query is presented in chapter 5.

1.2 Objectives and Contributions of this Thesis

As we discussed in the previous sections, various spatial queries with several processing techniques have been proposed for different environments. However, existing techniques still suffer from query processing on the road network either stationarily or continuously. This thesis aims to solve these problem with twofold approaches. First, we studied snapshot $RkNN$ queries using pre-computation approach. Second, we expanded our studies for continuous querying the spatial objects with the mobile query object. For continuous queries, we studied several vicinity queries and trip route planning queries which have widespread usage in daily activities. To accomplish these spatial continuous queries, we proposed several safe-region gen-

eration methods which reduce the communication and computation cost between the query objects and server. In addition, it can address the main challenges of continuous queries which is the maintenance of the valid query answer with the mobile objects.

1.2.1 Contributions on Snapshot RkNN Queries

The detail of the algorithms will be covered in chapter 3. Here, we summarize the main contribution as follow:

- We proposed an efficient reverse k nearest neighbor ($RkNN$) queries including monochromatic $RkNN$ and bichromatic $RkNN$ that is implemented with simple materialized path view ($SMPV$) framework. This study is mainly related to the instantaneous queries.
- An extensive experimental evaluation were carried out for both $MRkNN$ and $BRkNN$ queries for real road network are presented. These benchmark test results showed that the new method improves the processing time significantly.

1.2.2 Contributions on Continuous Vicinity Queries

Continuous queries generally have client-server model. The task of the server is to compute the query requested by the query object continuously and update the query according to the current location of the object. The main challenges of the spatial queries with moving objects are the maintenance of the up to date query result when the querying object moves arbitrary and minimizing the communication cost between the server and the moving objects. To acknowledge these challenges, we proposed a new approach for moving object called safe-region generation for various vicinity queries. The details of this query processing technique are discussed in chapter 4. Here, we summarize the main contributions as follow:

- We proposed a fast safe-region generation method applicable for vicinity

queries including range queries, set-kNN, ordered-kNN and $RkNN$ queries in road networks. Our proposed safe-region technique does not compromise on query types and maintains the efficiency.

- In addition, we proposed a new method to generate safe-region utilizing materialized run-time distance view. This method generates the safe-region efficiently and has significant performance improvement.
- The experimental studies showed that the proposed safe-region generation method outperforms the existing algorithms in the computational processing time.

1.2.3 Contributions on Trip Route Planning Queries for Stationary and Moving Objects

For the trip route planning query, we examined its earlier studies and found that it is a kind of time consuming query especially when the data objects are distributed sparsely. To reduce the processing time, we introduced sparse category first approach. In addition, the previous studies are mainly for static objects and although, there is some approach for TRP query with moving objects, they are impractical to some extent. One of techniques for querying with mobile objects: safe-region for several continuous queries has been studied except the TRP query. Therefore, we proposed continuous $TRPQ$ on road network with multiple stopovers using the safe-region method. The detail explanation of our strategies for the $TRPQ$ is presented in chapter 5.

The main contribution of this thesis in TRP query are as follow:

- As a first attempt, we proposed a continuous trip route planning query for the spatial network utilizing an efficient safe-region techniques called the preceding rival addition (PRA) and the tardy rival addition (TRA) to generate the safe-region for continuous monitoring.

- We introduced an on-the-fly network distance materialization method which is amenable to efficient safe-region generation.
- According to the conducted experiments, the snapshot trip route queries usually consume longer processing time when it works with multiple data object categories whereas our methods can reduce the processing time significantly and monitor effectively the moving query objects.

1.3 Thesis Organization

The rest of this thesis is organized as follows.

1. Chapter 2 reviews some relevant background knowledge, querying techniques in static spatial database and moving object database.
2. The snapshot $RkNN$ queries using $SMPV$ structure and an analytical analysis of proposed methods are described in Chapter 3.
3. Chapter 4 presents a safe-region generation approach for continuous vicinity queries with moving query object.
4. Chapter 5 presents the new query called the Continuous Trip Route Planning Query ($CTRPQ$). We present $CRTPQ$ with the various safe-region techniques with experimental evidences.
5. Finally, the conclusion of the thesis is presented in Chapter 6.

CHAPTER 2

Literature Review

In this chapter, we discuss a brief overview of the theoretical background and the related studies previously carried out on the topic presented in this thesis.

2.1 Spatial Index Structure

Study of spatial database is one of the active research area in current database research activities. A well-known spatial database system is Geographical Information System (GIS). A spatial database system consists of a collection of objects over the multidimensional space. Spatial data is huge in quantity and complex in structure and relationship. For a road network consisting a large set of spatial data objects, pre-computing the distance for all pair of nodes and storing all the data objects becomes inefficient and sometimes infeasible. One of the methods to retrieve road network distance between two nodes creates precomputed distance tables and stores the road network distance value in a database. Efficient processing of the spatial queries mainly depends on the index structures. The spatial data index structure not only supports efficient spatial operations such as locating the closest objects and identifying the objects in a definite query region, but also optimize the

search operation on spatial data objects. The way how this is done is explained in the following sections.

2.1.1 Indexing Methods in Spatial Database

Indexing in a spatial database is different from indexing in a conventional database. In the spatial database, the database system is constructed with multidimensional objects associated with the geographical locations and features. Objects in spatial database are composed of geometric elements such as lines, points, polygons and irregular shapes to represent geographical locations. To deal with such geometric elements in the database, conventional indexing approach such as Hash table [62] or B-Tree [3] is not suitable for spatial data operation such as insertion, storage of the objects and retrieving of the relevant data for computation. Hash tables are based on the exact matching which do not support in searching the closest objects within the specified range, whereas B-Tree relies on single attributes ordering which is not applicable for retrieving the closest objects. To solve this problem, we need an effective index structure suitable for spatial database. The spatial data objects often cover areas in multi-dimension and a common operation on spatial data is to search for all objects in an area. The R-Tree [4] algorithm, proposed by Guttman in 1984, is the most used indexing in spatial query.

R-Tree

R-Tree [4] is a hierarchical, high balanced data structure in which all leaf nodes appear at the same level, designed for use in secondary storage. It is a generalization of the B-Tree for multi-dimensional spaces and one of the earliest proposed tree structures for non-zero sized spatial object indexing. The key idea of R-Tree is grouping the nearby objects and indexing them using their minimum bounding rectangle (MBR). In a R-Tree, objects are represented by tuples, and each tuple has a unique identifier for retrieving its leaf nodes. Leaf nodes contains entries of the form $(R, tuple - identifier)$ where R is a d-dimensional rectangle so called MBR

and $tuple - identifier$ is the identifier of the objects detailed description. Non-leaf nodes contain entries of form $(R, child{-}ptr)$ where $child{-}ptr$ contains the address of a lower level nodes in R-Tree and R covers all rectangles in the lower node's entries. The structure and characteristic of R-Tree is illustrated in Figure 2.1. R-Tree has several advantages to be applied in spatial database compared with the traditional indexing methods. Unlike hash based indexing methods, R-Tree supports efficiently exact matching. R-Tree provides two-dimensional ordering and it can be extended dynamically to multi-dimensional spaces. This is the main advantages comparing with the B-Tree which provides only one-dimensional ordering of a single key value. Due to the effectiveness of R-Tree, it has been adopted as one of the useful indexing method for spatial queries in LBS. Moreover, new applications and queries continue to demand improved indexes and associated algorithms.

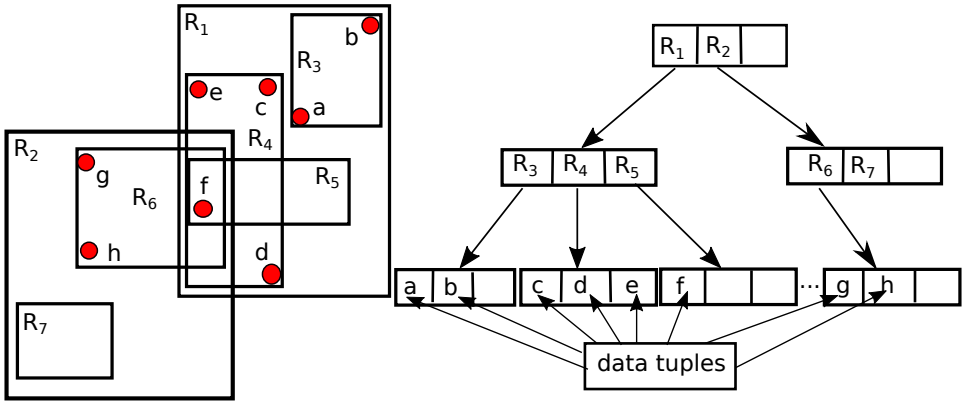


Figure 2.1: An example of R-Tree index structure.

Voronoi Diagram in Euclidean Space

Next spatial data object indexing structure is Voronoi diagram (VD). A Voronoi diagram divides a given area into the disjoint polygons. The closest objects of any points inside a polygon is called the generator of the polygon. The concept of the Voronoi diagram in both Euclidean and network spaces is presented in [14]. Suppose P: $\{p_1, p_2, ..., p_n\}$ be a set of n distinct points called generator points distributed in an Euclidean space. These generator points can be the spatial objects such as restaurant or bank. All locations in the plane assemble to their closest generators.

A set of locations allocated to each generator forms a region called Voronoi polygon of that generator. The set of Voronoi polygons associated with all the generators is called the Voronoi diagram with respect to the generators set.

In Euclidean space, Voronoi polygon is a convex polygon. The Voronoi polygons of a Voronoi diagram are collectively exhaustive because every location in the plane is associated with at least one generator. The polygons are also mutually exclusive except for their boundaries. The boundaries of the polygons, called Voronoi edges, are the set of locations that can be assigned to more than one generator. Each edge of Voronoi polygon is a segment of the perpendicular bisector of the line segment connecting p to another point of the set P. The Voronoi polygons that share the same edges are called adjacent polygons and their generators are called adjacent generators. The Voronoi polygon and Voronoi diagram can be mathematically defined as follow: Assume a set of generators $P = \{p_1, p_2, ..., p_n\}$, where $2<n<\infty$. The Voronoi polygon is given by:

$$VP(p_i) = \{p | d(p, p_i) \leq d(p, p_j), i \neq j, j \in I_n, I_n = \{1, ..., n\}\}$$

where $d(p, p_i)$ denotes the minimum Euclidean distance between p and p_i (e.g.,length of the straight line connecting p and p_i in Euclidean space), is called the Voronoi polygon associated with p_i, and the set given by: $\mathrm{VD}(P) = VP(p_1), ..., VP(p_n)$ is called the Voronoi diagram generated by P. Figure. 2.2 shows the simple Voronoi diagram with 7 generator points in Euclidean space.

Voronoi Diagram in Road Network

The network Voronoi diagram (NVD) in the road network space is a generalized version of VD created by replacing the Euclidean distance with spatial road network distance [79]. In NVD, the distance between the objects is the road network distance.

Definition 2.1. *A spatial network can be modeled as a directed weigh graphs* $G =$

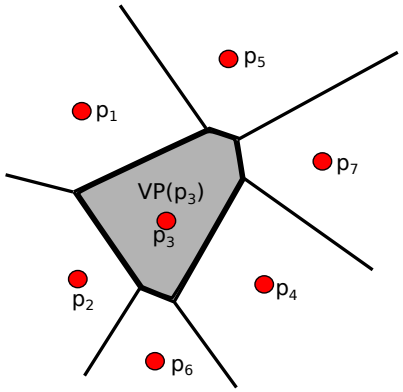


Figure 2.2: The Voronoi diagram (VD) in Euclidean space.

(N, E), *where* $N = N_i | 1 \leq i \leq n$ *is the set of nodes representing intersection and terminal points called the nodes of the graph.* $E = (E_i, E_j) | 1 \leq i, j \leq n$ *and* $i \neq j$ *is the set of edges representing the network edges each connecting two nodes* (n_i, n_j), n_i *and* n_j *are starting and ending nodes, respectively.*

Assume that, Voronoi generators are located on the network segments as the graph nodes. Each edge connecting nodes p_i, p_j stores the network distance $d_N(p_i, p_j)$. For nodes that are not directly connected, $d_N(p_i, p_j)$ is the length of the shortest path from p_i to p_j. The dominance region and the border points of the each Voronoi polygon (VP) are defined as:

Definition 2.2. *Dominance Region of* p_i *over* p_j

$Dom(p_i, p_j) = \{p | p \in \cup_{o=1}^{k} e_o, d_N(p, p_i) \leq d_N(p, p_j)\}$ *represents all points in all edges in* E *that are closer or equal distance to* p_i *than* p_j.

Definition 2.3. *Border Points between* p_i *and* p_j

$b(p_i, p_j) = \{p | p \in \cup_{o=1}^{k} e_o, d_N(p, p_i)\}$ *represent all points in all edges that are equally distanced from* p_i *and* p_j.

The elements of NVD are mutually exclusive and collectively exhaustive similar to ordinary Voronoi diagram. The network Voronoi diagram can be constructed using the parallel Dijkstra algorithm [1] with the Voronoi generators as multiple sources [79]. Specifically, the random initial node can expand shortest path trees

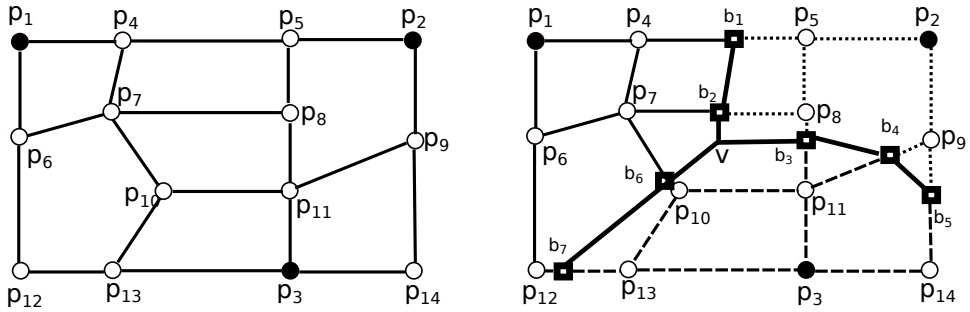
from each generator simultaneously and stop the expansions when the shortest path trees meet. The shortest path tree is formed by linking the next hop of all nodes involved in the calculation [71]. Figure 2.3 shows an example of network Voronoi diagram (NVD).

Figure 2.3(a) depicts the road network as a directed graph $G(N, E)$ in which N=$\{p_1, p_2, ..., p_{14}\}$ are nodes, p_1, p_2, p_3 are the Voronoi generators (e.g., data objects) and p_4 to p_{14} are nodes connected by a set of edges. Figure 2.3(b) illustrates the network Voronoi diagram of original road network of Figure 2.3(a) where each line style of corresponds to the shortest path tree according to each generator points. Each shortest path tree composes a network Voronoi polygon (Voronoi cell). Some edges (e.g., $e(p_4, p_5)$) can be partially contained in different Voronoi cells and $e(p_4, p_7)$ is completely inside the Voronoi cell of p_1. The border points b_1 to b_7 are the nodes where the shortest path tree meet as a result of the parallel Dijkstra's algorithm (this algorithm is described in next section).

The border points between any two generator points are equally distanced from each other. The figure also describes how adjacent border points should be connected to each other: when the two adjacent border points (e.g., b_6, b_7) are between two same generators, they can be connected with an arbitrary line that does not cross any edges. In addition, any three or more adjacent border points (e.g., b_2, b_3, b_6) can be connected to each other through an arbitrary auxiliary point (v in figure). Moreover, unlike Voronoi polygons in Euclidean space, common edges between two network Voronoi polygons contain more than two border points and thus they are not necessarily straight lines.

2.2 Road Network Distance Calculation

The fundamental and widely used query in spatial network is shortest path finding query between the two objects such as hotel and meeting venue. Such kinds of query algorithm has been studied since 1950's with various strategies and data structure.



(a) Road Network (b) Network Voronoi Diagram

Figure 2.3: (a)An example of road network. (b) An example of network Voronoi diagram of road network(a).

The performance of spatial queries in spatial network are essentially examined by its ability to search the objects of interest efficiently in terms of processing time and storage requirements. There are two main approaches to examine the network distance between two nodes in spatial networks. These are 1) compute-on-demand approach and 2) pre-computation which employs precomputed distances between data points and network vertices for improving the performance. The well-known shortest path finding algorithms in spatial networks are Dijkstra's [1] and $A*$ [2]. These are described in the following sections.

2.2.1 Description of Algorithms

In this section, we present the various algorithms of road network distance calculation with their principles and processing procedures.

Dijkstra's algorithm

Dijkstra's shortest path algorithm [1] is the fundamental algorithm in computer science and related fields for network distance calculations. Dijkstra's algorithm uses the greedy approach to solve the single source shortest problem. It works by solving the k sub problems by computing the shortest path from source to the vertices among the k closest vertices to the source.

The key idea of the Dijkstra's algorithm is keeping all the shortest distance of vertex v from the source in a priority queue. Initially, distance form source to all other vertices are set to infinity to indicate that those vertices are not yet processed. After the algorithm finishes the processing of the vertices, the shortest distance of vertex from source to every other vertex is updated. The procedure of the Dijkstra's algorithm is as follows:

1. First, the algorithm marks all nodes as unvisited and sets the initial node as a current node.

2. For the current node, it loops to all its unvisited neighboring nodes and calculates their distances. After that, it makes a comparison with all distances and chooses the one that has a minimum value. Then the selected node is marked as visited node which is never checked again.

3. If the target node has been marked as visited or the minimum distance is infinity means there is no connection between the initial node and the remaining unvisited nodes. In such case, searching will be stopped and the algorithm will terminate.

4. The algorithm iterates with the unvisited node that is marked with minimum tentative distance, and sets it as the new current node then loops back to step 2.

Although Dijkstra algorithm can determine the best route, it has some restrictions such as the graph should be directed-weight graph and the edges should be non-negative. If the edges are negative, the actual shortest path cannot be retrieved. The shortest path searching between s and d using Dijkstra's algorithm is illustrated in Figure 2.4. Although Dijkstra's finds the shortest path, Dijkstra's algorithm does a blind search and it visits the nodes in all directions like a circular wavefront and searches from a start node and then gradually expands the search space to neighboring nodes. Consequently,it consumes a lot of time. On a graph with n nodes and m edges, the complexity of the processing time will be $O(n^2)$ because

it allows for directed cycles. To overcome those defections, $A*$ algorithm has been proposed which is described in the next section.

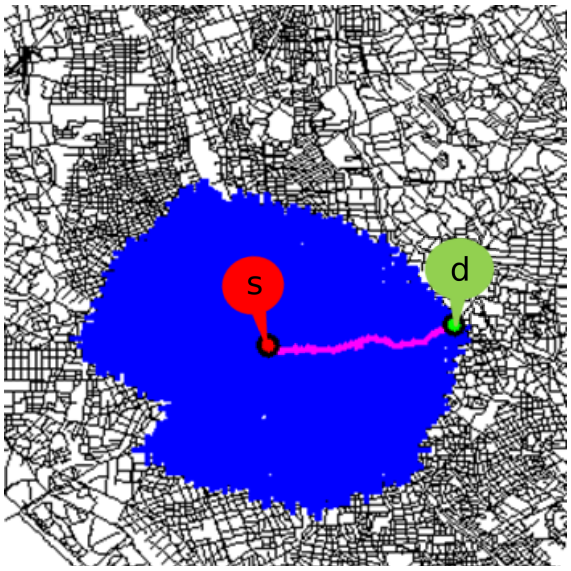


Figure 2.4: The example of shortest path searching between s and d on road map and the way of expansion of the searching area using Dijkstra's algorithm.

$A*$ algorithm

Hart et al. [2] introduced $A*$ algorithm which is a graph search algorithm that finds the shortest path for a given initial node to a given goal node using the best first search approach instead of greedy approach that is used in Dijkstra's. $A*$ visits the nodes according to the estimation of the heuristic estimation $h(x)$. However, it employs the advantages of Dijkstra's algorithm that is favoring the vertices closer to the starting point and pros of the best first search favoring vertices that are closer to the goal. Consequently, $A*$ is faster compared to Dijkstra's algorithm. In addition, the heuristic implantation improves the efficiency of search process.

In $A*$ algorithm the cost function $f(n)$ is calculated as $g(n) + h(n)$, where $g(n)$ is the distance value between two nodes represented as actual cost and $h(n)$ is the estimated heuristic cost (distance value) from node v to destination node. In $A*$ algorithm, $h(n)$ is admissible means it should not overestimates actual cost to

guide an optimal path.

$A*$ algorithm has two main input parameters: first parameter holds nodes which will be next visited nodes in the path (the "open" set) and second parameter handles nodes that have already been visited (the "closed" set). The processing of $A*$ algorithm is as follows:

1. Set a start node s.
2. Put the neighbor nodes of start node s by referring the adjacency list into " open" set as unexpected nodes.
3. If "open" set is not empty; the following steps are continued. Otherwise, the search algorithm is terminated.
4. Remove a node n with minimum f value from "open" set, and place it into a "closed" set to be used for expected nodes.
5. Expand node n, generating all its adjacent nodes with point back to previous node (node n).
6. Iterative process for all adjacent nodes n' of n. (a) Calculate f(n'). (b) If n' is not included in "open" set, add it into "open" set. Then, assign the other computed $f(n')$ to node n. (c) If n' exists in "open" set, compare the newly computed $f(n')$ with previously assigned n' node. If the new value is lower, substitute it with the old (i.e., update the cost of this node to any successors). Then, add node n' to "closed" set.
7. Loop to step 2.

An empirical study found that $A*$ algorithm explores less than 10% of the nodes expanded by the Dijkstra algorithm [1]. The result is shown in Figure 2.5. Some researchers demonstrated that $A*$ algorithm identifies the shortest path in many Euclidean graphs with an average polynomial computational complexity [5]. Owing to its performance and accuracy, $A*$ algorithm is one of the most popular

algorithm implemented in current navigation systems. However, the main drawback of $A*$ algorithm is memory requirement since it needs to save the entire openlist and it is severely space-limited in practice.

Figure 2.5: The example of shortest path searching between s and d on road map and the way of expansion of the searching area in $A*$ algorithm.

Single Source Multi Target A* ($SSMTA*$)

The single source multi target $A*$ algorithm [84] is a variation of the $A*$ algorithm. It is designed to apply in searching the shortest distance from one single source to multiple target objects. As we mentioned above, $A*$ algorithm is faster than Dijkstra's algorithm to find the shortest path from a given initial point to a given target node. However, the performance of $A*$ algorithm deteriorates when the target nodes are more than one because $A*$ algorithm executes the same node repeatedly. Consequently the searching areas on the road network overlap while searching the target object, and the total number of searching at each node are increased. To reduce the searching iteration for each road network node and searching area, node expansion can be controlled by employing the heap for each target point [69]. However, when the target nodes are multiple, several heaps are required to manage the search area and the contents in each heap must be kept the same which requires much more processing time. To cope with these drawbacks, $SSMTA*$ algorithm controls the

node expansion by using only one heap and avoid the content synchronization.

$SSMTA*$ algorithm finds the shortest path in basically same way as the $A*$ algorithm. However, it searches the shortest paths to multiple target points simultaneously which is the main difference of $A*$ algorithm. The way of searching the multiple target points and node expansion in $SSMA*$ algorithm is demonstrated in Figure 2.6. In the figure, s is source and t_1 to t_4 ($\in T$) are target points. Suppose, the shortest path from s to node n has been found. Before reaching the target points, the closest neighbor nodes of n such as n_a to n_c are obtained by referring adjacency list. To calculate network distance adopting A* algorithm, we need a heuristic distance (Euclidean distance) between each neighbor node and target points. For instance, the minimum heuristic Euclidean distance from n_a to t_2 which is the closest of t_2 is $d_E^{min}(n_a, t_2)$. The minimum cost from s to t_2 is assigned as Cost= $d_N(s, n) + d_N(n, n_a) + d_E^{min}(n_a, t_2)$. When the searching reaches each target node, the target node is removed from the target list (T). The searching operation is repeated over the remaining neighbor nodes of n. Finally, the algorithm terminates when the search paths from s reach to all the target points.

When the node expansion reaches the target point t, t is removed from the target list T. However, the entry for which the cost from the nodes to target point remains in the heap are calculated. To synchronize the history record, the function recalculates the cost in heap based only on the remaining target points. In $SSMTA*$, the function called *RenewQueue* recalculates the cost in proportion to the size of heap. Heap contains only wave front nodes and the number of wave front nodes is roughly proportioned to the distance from s. The recalculation function is only invoked when the target list is changed, and no disk access is required. Because, $SSMTA*$ reuses all records inside heap and the order of nodes in heap according to the recalculated cost value, the processing cost is lower than executing $A*$ algorithm.

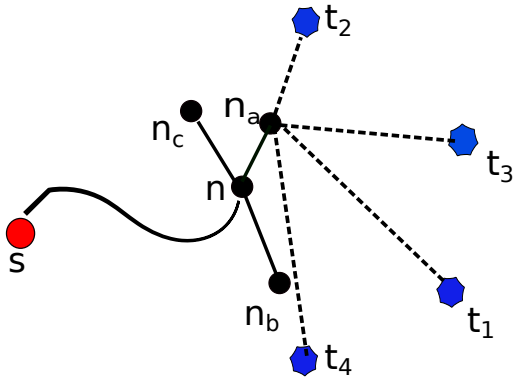


Figure 2.6: The example of searching the shortest paths to multiple target points and node expansion in $SSMTA*$ algorithm.

Shortest Path Finder using Materialized path view (MPV)

Materialized Path View (MPV) can retrieve the optimal shortest path by computing paths between all pairs of source and destination nodes. If all paths are enumerated in such a path view, a single lookup query serves to meet any path request. Although, the shortest path can be retrieved fast on MPV, such kind of structure requires an unrealistically large amount of storage. It needs $O(n^2)$ space when the number of nodes on the given graph is n. If the road network is large, this method is infeasible to apply in real situations.

A semi-materialized view approach, proposed by Agrawal et al. [7], is a compromise between space and retrieval efficiency. It encodes the path information with encoding structure which keeps with a via node. Via nodes are the first hop in the path from the source node to the destination node. In this approach, the next hop procedure can be retrieved with one lookup query. Jing et al. [11] proposed a semi-materialized method of the shortest path route to reduce the data amount. It only records the next pursued node along the shortest path, and the whole shortest path route is restored by tracking the next visiting node in the sequence. Samet et al. [54] reduced the data amount to $O(n^{1.5})$, using the shortest path quad tree leads to the reduction of the storage requirements.

Several methods based on materialized path view (MPV) have also been pro-

posed to compute the road network distance faster. When two points are located on the road network nodes, the distance can be obtained by only single access to the table. Generally, two points are not always located on nodes, which means data points exist on the network link. In such situation, distance cannot be obtained directly from the tables and needs to be gathered from the nearest nodes of each data point. After investigating the nearest nodes of data points and network vertices, the distance can be referred from each respective table, therefore, at most 4 times access is required. However, in any case, the road network distance can be determined in a constant time by referring the precomputed distance table in MPV.

Car navigation systems sometimes search the shortest paths between two points located very far away. In this situation, the most suitable search method can be considered as a hierarchical structure based on the types of roads [11]. For example, roads are divided into the highway and the usual road. First, it searches a rough shortest path on the highway network, and then searches the path between each given terminal point and the access point of the highway on the usual road network. Though this method may not give the shortest path, the result is adequate for the practical purposes.

A shortest path finder (SPF) [85] based on a lightweight local distance materialization called simple materialized path view ($SMPV$) is constructed with the partitioned subgraphs. In SPF, the road network is partitioned into the subgraphs, and the distance materialization is performed only in the subgraphs. Therefore, the amount of precomputed data is greatly reduced. The shortest path is retrieved using a best first-search approach in a priority queue. In SPF, the road network nodes are classified as inner node that belongs to only one subgraph and border node that belongs to at least two neighboring subgraphs. This method outperforms the $A*$ algorithm, as they reduce the data amount drastically compared with the conventional hierarchical distance materialization methods.

In $SMPV$, the length of road network is calculated by traveling inside the subgraph. If there is no connected path between a paired node inside the subgraph, algorithm refers the table called border-to-border distance table ($BBDT$). To retrieve the distance from the starting point as an inner node to a border node, the inner to border node distance table ($IBDT$) is referred to obtain the distance. the node-to-node distance table ($NNDT$), that lists the distances of all combinations of the nodes in each subgraph, is used to know the distance between two arbitrarily specified nodes. There are three SPF approaches based on the variations of the distance materialization level on each subgraph such as full materialization ($SPFFM$), medium materialization ($SPFMM$) and light materialization ($SPFLM$). The main difference of these three approaches are as follow:

- $SPFFM$ determines the shortest path by referring the $NNDT$ which has all combinations of the distances between any two inner-nodes.
- $SPFMM$ obtains the distance by referring $IBDT$ and $BBDT$ tables when the nodes are not inside the subgraph. Otherwise, the distance is obtained by $A*$ algorithm.
- $SPFLM$ determines the shortest path by $A*$ algorithm by referring to usual adjacency list.

Figure 2.7 is illustrated the shortest path searching using the $SPFMM$ approach on the $SMPV$ structure.

2.3 Categories of the Spatial Queries

Spatial databases have been studied extensively in the last two decades resulting in rapid development of numerous conceptual models, multi-dimensional indexes and query processing techniques. The main challenge of the spatial queries for LBS applications is handling distinct types of objects for instance query object and data objects are either static or moving, both are static, and both of query

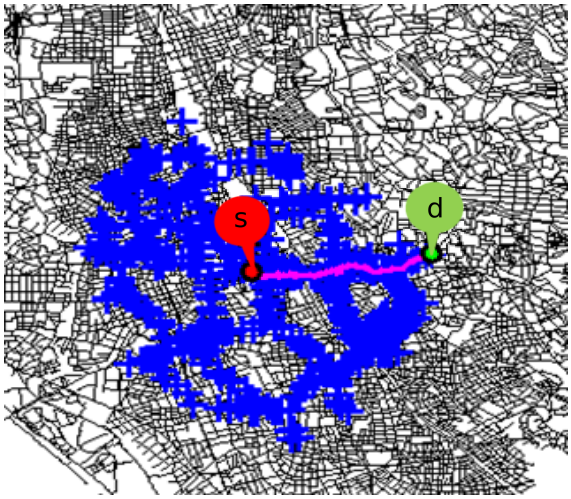


Figure 2.7: The example of shortest path searching between s and d on road map and the way of expansion of the searching area in the $SPFMM$ approach.

and data objects are in a mobile environment. Similarly, we can outline specifically the spatial data processing on types of spaces such as data processing considering Cartesian (Euclidean) distance or network distance.

The related research works of our studies on the spatial queries will be discussed with the two main groups called static and continuous queries.

2.3.1 Static (Snapshot) Spatial Queries

The snapshot query mainly targets the static data set including all the spatial objects of interest and the query objects (e.g. restaurant, convenience store). This section describes all related works of the static queries by classifying with the query types as follow:

Range Query

Range Query is one of the common queries in spatial database. There are several studies to retrieve all objects whose location lie within the user defined region. Based on the calculation methodology, the range query is named as rectangular range query when the query returns the objects that lie within a rectangular region,

circular range query when the search space is a circular region and distance range query retrieves the data objects within the specified distance range. The traditional range query using an index structure such as R-Tree to finds all bounding rectangles intersecting a given range is called the rectangular range query. The search process descends all subtrees that intersect or fully contain the range specification.

Papadias et al. [26] proposed two methods to achieve the range query on road network called range Euclidean restriction (RER) using the Euclidean lower bound distance between data point and query object, and range network expansion (RNE) which retrieves the data set within network range e from q. Although, their Euclidean based range is feasible for the real-world scenario, the network distance range search has time-consuming and storage space issues. Xuan et al. [68] introduced the range query with Network Voronoi diagram. However, their approach was not directly applicable for network distance. So, as an extended version, Xuan et al. [75] proposed range query on road network as constraint range queries using the Voronoi diagram. Their approaches are restricted with region (specified location/place) and k nearest objects. For instance, retrieves maximum 5 parking lots within $3km$ in city area, if the number of parking lots is less than 5 within $3km$, their approach try to examine to enlarge range radius to satisfy 5 nearest parking lots.

kNN Query

The next common spatial query is the nearest neighbor NN and kNN with arbitrary k value. In the beginning, NN and kNN queries were processed as a snapshot (i.e., one-time single output) query over static objects, assuming that the distance function is Minkowski metric (such as Euclidean). It indexes the data with a spatial access method (e.g., an R-Tree [4]) and utilizes the distance bounds between the index nodes and the query point to restrict the search space [13]. Several methods were proposed to efficiently process the nearest neighbor queries for stationary points. Some of the methods rely on index structures built specifically for the near-

est neighbor queries [10], [12]. Branch-and-bound methods work on index structures originally designed for range queries.

To be compatible with spatial networks, spatial queries in spatial networks has been studied with various indexing schemes to manage spatial data and processing strategies. Papadias et al. [26] proposed algorithms to compute static range queries and nearest neighbor queries in spatial network. They proposed two approaches 1) incremental Euclidean restriction (IER) searches k-nearest neighbor points of interest (POIs) based on the Euclidean distance before verifying the distance on the road network using Dijkstra's algorithm. and 2) incremental network expansion (INE) approach that efficiently supports the exact kNN queries on spatial network databases. However, these approaches suffer from deficient performance when the objects (e.g., restaurants) are distributed sparsely in the network.

As an optimization approach of IER, Deng et al. [69] proposed new approach called lower bound constraint (LBC) that calculates the lower bound distance of the object. Lower bound distance is used for pruning hence, the workload of the network distance calculation is reduced. Kolahdouzan et al. [32] studied k nearest neighbor search using the network Voronoi diagram (NVD) for spatial network on static objects. This approach is based on partitioning a large network to small Voronoi regions, and then pre-computing distances both within and across the regions.

RkNN Query

Another type of spatial query called RNN query was first proposed by Korn et al. [15]. Their algorithm requires pre-computed data, in which the distance from each point to its NN. Given this data, a set of point and their distances to the NN are registered in an R-Tree, and the circle centered at a data point with the radius equal to the distance to the NN called its vicinity circle. This approach is not suitable for RNN query with k values and road network. Because of the R-Tree construction using vicinity circles with predefined k^{th} NN distances. Stanoi et al. [16] and Tao

et al. [31] proposed $RkNN$ queries without using the pre-computation approach called SAA and TPL respectively. Although, these approaches were applicable to general $RkNN$ queries, they were not feasible for spatial road network because they only focused for Euclidean plane.

In the foremost, Yui et al. [47] proposed $RkNN$ query algorithm relevant to the road networks. In $RkNN$ algorithm, the area in which the reverse kNN objects exist is searched gradually enlarging the search area using the Dijkstra's approach. They proposed two algorithms called Eager and Lazy algorithms that differ in their respective pruning methods. In eager method [47], kNNs are searched at every visited road network node and prune nodes that cannot be reverse nearest neighbor of query point q proactively. In essence, the Eager algorithm is efficient only when the search area and k value are small. In lazy approach, it exploits the verification phase of the algorithm to prune nodes for their future searches. Lazy traverses a large part of the road network.

Safar et al. [63] proposed a solution for snapshot RNN queries in spatial networks based on manipulating network Voronoi diagram. In addition, they introduced a progressive incremental network expansion ($PINE$) approach to find the Voronoi polygon on the network distance. They also extended the studies for proximity $RkNN$ using the same approach of Tran et al. [64]. Similarly, $RkNN$ query algorithm using the $SMPV$ structure and adopting IER strategy was proposed by Hlaing et al. [89]. Their approach solved the performance deficiencies of the Eager algorithm [47].

Trip Route Planning Query

Another spatial query gaining attention in location-aware application is trip route planning query. As an earlier study, Li et al. [38] introduced trip route planning query called trip planning query (TPQ). TPQ mainly targeted for the static objects on spatial network. Given a source s and a destination d, the TPQ retrieves the

shortest route from s to d passing through at least one object of interest from each category, however, there are no order constraints in category. This paper proposed several approximation algorithms to provide the near optimal solution of the TPQ problems in metric spaces for road network distance, whose approximation ratios either depends on the number of categories, or the maximum number of categories of each type, or both. When finding the optimal route with TPQ, the candidate space extremely grows, thus, it requires enormous processing time due to the lack of any restriction on the visiting order of the data category. TPQ requires the processing time in proportion to M ($\prod_{i=1}^{M} N(C_i)$) where M is the number of data points set to be visited during the trip and N(C_i) is the number of data points in a category C_i. Therefore, TPQ is only feasible when M is small.

One of the related study of TPQ query called the optimal sequence route (OSR) query, addressed to solve the TPQ problem, was proposed by Sharifzadeh et al. [39] in both vector based on the Euclidean distance and metric spaces based on the road network distance in which visiting order was explicitly considered. [39] introduced light threshold based iterative algorithm called $LORD$ for Euclidean space OSR and progressive neighbor expansion (PNE) algorithm for network space OSR query. There are some spatial queries that could be posed to a spatial database community called aggregate nearest neighbor queries [45], [94] which return the objects with minimize aggregate distance with respect to a set of query points, skyline query [18], [19] and top$k$ query which retrieve the most interesting and preferred items based on all the preferences of all users.

2.3.2 Continuous Spatial Queries

Mobile devices and widespread wireless networks have brought a propagation of location-aware applications such as traffic monitoring, enhanced emergency service and mixed-reality games. Such applications involve either data object or query object moves unpredictably and both data and query objects are mobile, therefore, their locations have to be updated frequently. In such highly dynamic environment,

the results of traditional spatial queries are no longer valid due to the location of either query or data objects are constantly changing. A very recent trend in spatio-temporal database research is to continuously monitor such query results. Although there are numerous spatial queries, we emphasize only the spatial queries that are related with our studies in this thesis.

Continuous Range Query

Prabhakar et al. [23] proposed velocity constrained indexing and query indexing for continuous evaluation of static queries over moving objects. Distributed approaches for continuous range query was introduced in [34]. Gedik et al. [34] introduced a technique called MobiEyes, which reduces the computation load on the server and communication costs between the clients and the server by delegating some computation load to the client objects. To enhance the performance and system utilization of MobiEyes, [48] introduced a set of optimization techniques, such as Lazy Query Propagation, Query Grouping, and Safe Periods, to constrict the amount of computations handled by the moving objects. But, there approaches did not target to apply for road network.

As a different approach, Hu et al. [43] studied a generic framework to monitor continuous range queries and kNN queries over moving objects. They define the safe zones for each data object such that the query results remain unchanged if the object does not leave the region. However, they assumed that queries objects are registered in server and kept in main memory. The previous continuous studies are applied for the both of query and data objects are moving but there is no clear descriptions for the spatial road network.

Earlier studies for continuous range query on road network is proposed using the tree index structure and the tradition way include the filter step and refinement steps in [52]. Similarly, [77] used the grid index and distance pre-computation approach. As an different approach for study on continuous query, [73] proposed

distance based range query that continuously changes the locations in a Euclidean space and returns every objects (o) that lies within distance (r) of the moving query location q. Their proposed range query used the concept of safe zone (in other literature called as safe-region) in which the query result does not change. Kefeng et al. [75] extended their previous studies in static queries [68] using their network Voronoi diagram to achieve the continuous range query for road network.

Continuous kNN Query

There are several studies on the kNN queries over moving objects using traditional indexing techniques [28] such as Voronoi diagram and safe-region [57], [65]. The concept of the safe-region provides an efficient way to achieve the continuous queries with accurate answer without sampling the moving objects. Nutanong et al. [57] studied moving kNN queries using the incremental safe-region based technique called $V*$-Diagram. The $V*$-Diagram computes safe-regions based on not only the data objects, but also the query location.

Similarly, Hasan et al. [65] studied moving kNN queries using safe-region. The algorithm in [65] for moving kNN queries assumes known trajectory path and needs to set in advance the sufficient resources according to the dataset. However, these studies are mainly functionable for Euclidean space. Recently, the attention of spatial database researches has shifted to the continuous queries especially in spatial network [41], [49], [88] which is essential for the mobile data. The first one used tree based index approach. But their approach need large memory because the network information, data objects and queries are stored in memory. The continuous queries run for the long-time periods and demands frequent update which makes the query process more complicated than process of static queries.

Continuous RkNN Query

Benetis et al. [22] presented the first continuous RNN monitoring algorithm using TPR-tree (Time Parameterized R-tree [17]). Their approach is only for singly RNN

and Euclidean plane. As a different approach for continuous RNN and $RkNN$ queries, [51], [56] adopts a two phases computation technique. In the filtering phase, objects are pruned by using the existing pruning paradigms and the remaining objects are considered as the candidate objects. In the verification phase, every candidate object for which the query returns its closest point as the RNN. In this approach, whenever the query or a candidate result changes the location, the expensive pruning phase is needed to be re-invoked. To overcome such problem, Cheema et al. [66] introduced the lazy updates for continuous $RkNN$ queries by reducing the number of processing time to re-invoke the pruning phase by assigning a rectangular region for each moving object.

The continuous queries on Euclidean space cannot be applied directly for the real road map. Therefore, we need to seek alternative methodologies. For road network, Sun et al. [58] studied the RNN queries for moving objects in spatial network. They created the multi-way tree for each query that employs in identifying a monitoring region. They consider updating the query only in the region which affects the results. In addition, their studies mainly focus on the case where k=1. Cheema et al. [80] presented a continuous $RkNN$ monitoring algorithm for moving objects and queries in spatial networks which is extended version of [66]. They studied continuous $RkNN$ queries using safe-region. In [80], although they avoid frequent calls to the pruning phase, unfortunately their algorithm needs to verify the location of client whenever it changes its position. 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. Therefore, we need a methodology to address expensive verification issue.

Continuous Trip Route Planning Query

We continue with explaining the next common spatial query which has the great interest in navigation systems called trip route query processing. We mentioned in previous section that this query is mainly focused on static route [38], [39] op-

timization for snapshot query type. Nowadays, finding the shortest (optimal) trip route query applicable with the dynamic environment is frequently used. [60], [74] studied the dynamic route with the near optimal solution. Utilizing the Voronoi diagram, Chen et al. [67] presented OSR query which can apply for the continuous queries. Nirmesh et al. [74] used the combination of pre-computation and on-the-fly route calculation method to achieve approximate result.

Aljubayrin et al. [90] studied new path finding problem which extended studies of the path computation called the safest path via safe zone. In their scenarios, the buildings and street blocks are assumed as the regions of interest (safe zones). Their methods finds the optimal path to minimize the traveled distance through a set of discrete safe zones. Natanong et al. [81] studied the continuous detour queries (CDQ) in spatial networks which targeted to find the shortest route between two locations with a stopover. Although their research goal is to find the optimal trip route with minimum trip distance for moving objects on spatial network, their algorithm is not considering for multiple stopovers (not visiting multiple categories). Ohsawa et al. [96] proposed the continuous trip route planning query visiting multiple data points (multiple categories) taking the advantage of safe-region.

2.4 Synopsis of Our Proposal

We presented several processing mechanisms to achieve the static and continuous spatial queries above. To summarize the related studies of spatial queries, we have seen that most of them, either static or continuous queries, are developed using integration of index structure (R-Tree, Grid), and filter/refinement strategies, utilizing Voronoi diagram and safe-region derived from the Voronoi diagram theory. Among them, safe-region is mainly used for continuous queries. In this thesis, to process static spatial queries, we used the pre-computation approach because query object and objects of interest are static. Therefore, our goal in this study is to overcome the processing time problem of existing approach by achieving pre-computation

query processing. Specifically, we used a strategy of the integration of indexing (R-Tree) and materialized path view which is similar idea of the Voronoi diagram. The theoretical explanation of our studies for static query is presented in chapter 3. Next, we studied various continuous spatial queries. Pre-computation becomes problematic when the data objects or query objects are needed to be updated if they are mobile. To process continuous spatial queries with moving query object, the computation and query result with guaranteed accuracy are the major concern. To achieve these goals, we accomplished by applying on-the-fly query processing approach. More specifically, we proposed several safe-region generation techniques for continuous spatial queries and they are presented in chapter 4 and 5.

2.5 Summary

In this chapter, we mainly discussed about three topics that are very typical for the spatial database management system ($SDBMS$).

1. In first section, we briefly discussed about the various spatial indexing techniques for spatial objects.

2. Next, we presented the existing road network distance calculation techniques for spatial queries and introduced our proposed strategy for road network distance calculation. Road network distance calculation is very typical task for querying spatial objects especially on the road network.

3. Finally, we provided a brief overview of the related studies for each types of spatial queries we worked with in this thesis.

CHAPTER 3

Reverse k Nearest Neighbor (RkNN) Queries on Road Network

The goal of reverse k nearest neighbor ($RkNN$) query is to identify the influence of a query object on the whole date set. Although the $RkNN$ problem is the complement of the k nearest neighbor (kNN) problem, $RkNN$ and kNN problem are not symmetric. The naive approach for $RkNN$ problem on spatial network requires $O(n^2)$ time thus, more efficient approach is necessary. In this chapter, we present a new and an efficient $RkNN$ approach for road networks.

3.1 k Nearest Neighbor (kNN) Queries and Reverse k Nearest Neighbor (RkNN) Queries

In this section, the theoretical explanation and the general definitions of the kNN and $RkNN$ queries are described. When a set of objects of interest P distributed geographically around a spatial road network is given, a common type of spatial query in a location-based application finds the k nearest neighbors with an arbitrary $k \geq 1$ around a given query object q. This type of query is called k nearest neighbor

(kNN) query and mathematically we define the query as follow:

$$kNN(q, k, P) = \{r \in P | \forall p \in P : d(q, r) \leq d(q, p)\}$$

where r and p are the objects of interest (data points) from P set and $d(a, b)$ is the distance between a and b.

The above definition of kNN query is illustrated with Figure 3.1. In Figure 3.1(a), there are five points in a two-dimensional (2D) plane and the dotted lines indicates the distance between two points. The first and second nearest neighbor neighbors are represented as $1NN$ and $2NN$ respectively in Figure 3.1(b).

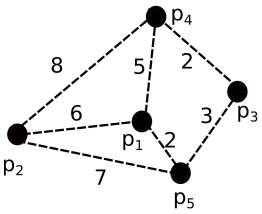
Next, we focus on the inverse relationship among the set of objects of interest (P). When a set of objects of interest P and a query point q ($\in P$) are given, if q is included in the kNN of p ($\in P$), q is called an $RkNN$ of p. For example in Figure 3.1(a), the RNN query for point p_1 returns point p_2 and p_5. p_3 and p_4 are not returned as results because they have each other as their nearest neighbors. Note that even though p_2 is not a nearest neighbor of p_1 because p_1 is the point closest to p_2. The result of $R1NN$ and $R2NN$ of each data point are listed in Figure 3.1(c). Commonly, we can define the reverse k nearest neighbor ($RkNN(q)$) as follow:

$$RkNN(q, k, P) = \{p \in P | q \in kNN(p, k, P)\}$$

where $kNN(q, k, P)$ is the k^{th} NN of p.

Generally, $RkNN$ query can be classified into two cases: monochromatic $RkNN$ ($MRkNN$) and bichromatic $RkNN$ ($BRkNN$) queries. In $MRkNN$, all objects of interest and query objects are the same data types. In $BRkNN$, the data objects and query objects belong to two different types of objects. A number of methods are proposed to process $RkNN$ with different index structures are proposed. The aim of these studies is to maximize the algorithm efficiency for different environments such as static or dynamic.

We studied $MRkNN$ and $BRkNN$ queries on road network with the assumption that traffic environment is static in [89], [91]. Our approaches for $RkNN$ queries employs $SMPV$ framework and IER strategy. The detail explanation of our strategies to solve the $RkNN$ queries are presented in the next sections.



(a)

	1NN	2NN
p_1	p_5	p_5, p_4
p_2	p_1	p_1, p_5
p_3	p_4	p_4, p_5
p_4	p_3	p_3, p_1
p_5	p_1	p_1, p_3

(b)

	R1NN	R2NN
p_1	$\{p_5,p_2\}$	$\{p_5, p_2, p_4\}$
p_2	ϕ	ϕ
p_3	$\{p_4\}$	$\{p_4, p_5\}$
p_4	$\{p_3\}$	$\{p_3, p_1\}$
p_5	$\{p_1\}$	$\{p_1, p_2, p_3\}$

(c)

Figure 3.1: The example of the kNN and $RkNN$ on the road network.

3.2 Monochromatic RkNN (MRkNN) Queries

In this section, we explain the basic concept of monochromatic $RkNN$ and our new strategy to achieve the $RkNN$ problem efficiently.

3.2.1 Basic Method for MRkNN Query

First, to understand the basic concept for $RkNN$ search in road network, we mainly consider for only one data type called Monochromatic $RkNN$. We define the $MRkNN$ query as follow:

$$MRkNN(q, k, P) = \{p \in P | d(p, p') \leq d(p, q)\}$$

where $d(p, q)$ is the distance between p and q. For query arising at q, the object p will be the RNN of q if q is the nearest point to p than p' ($\in P$).

A simple method for retrieving the RNN set initializes from q and traverses $p \in P$. The query registers q as a nearest neighbor if the distance between p and q is not greater than the distance between p and its NN. There is no fixed data size in the set $RNN(q)$ which may contain points that are not the closest point of q.

Thus, generally, RNN query needs to visit all data points. Yiu et al. [47] proposed the following lemma to minimize the road network traversal.

Lemma 3.1. *Let q be a query point, n be a road network node, and p be a data point that satisfies $d_N(q,n) > d_N(p,n)$. For any data point $p'(\neq p)$ whose shortest path to q passes through n, $d_N(q,p')>d_N(p,p')$. This means that p' is not an RNN of q.*

Let's explain the above lemma 3.1 with a diagram shown in Figure 3.2. In figure, when the algorithm probes the nearest data point from node n_7, the NN data point is p_3 and NN data point of p_3 is p_2; hence, the NN data point of q is p_3. However, according to the lemma 3.1, we observe that $d(q,n_7)>(n_7,p_3)$ and $d(q,p_2)>d(p_3,p_2)$. Therefore, if we search pass through from n_7, we cannot find the RNN of q.

Using the lemma 3.1, they proposed the Eager algorithm that visits the road network nodes from q to surrounding nodes in a same manner as the Dijkstra's algorithm. Specifically, q retrieves the k nearest data points within the distance between q and its nearest node n, $d(q,n)$. This function is called the **rangeNN**(q,n,d) as a first stage of the algorithm. In second stage, the nearest data point p of the node n is investigated to check whether q is the kNN of p by applying a **rangeNN** function. This function is called **verify** and the function is executed in algorithm as **verify**(p,k,q). Eager algorithm uses Dijkstra's shortest path finder to calculate the road network distance between data points and network nodes. The Eager algorithm can perform well when the density of data point is high as the search area remains small. In contrast, when the density of data point is low and the k value is small, the processing time increases rapidly because the search area is large.

To address this problem, we propose a new and fast $RkNN$ query by adopting IER strategy proposed by Papadias et al. [26] to retrieve kNN objects. In addition, we run our algorithm on $SMPV$ framework to construct an individual partitioned subgraph and to reduce the amount of data requirements.

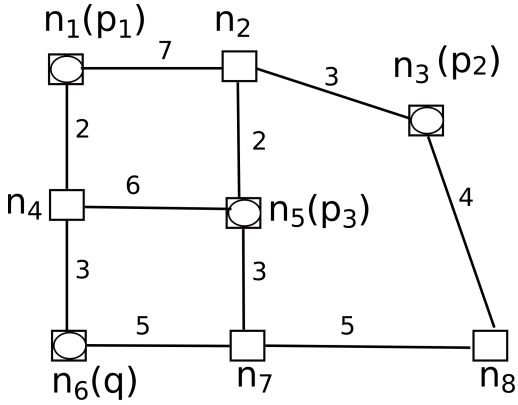


Figure 3.2: The basic example of searching $MRkNN$ on the road network applying lemma 3.1.

3.2.2 Simple Materialized Path View (SMPV) Structure

Before explaining our methodology for $RkNN$ query, we discuss briefly about $SMPV$ structure [85] that is applied as the underlying framework to run our algorithm. We briefly introduced about the $SMPV$ structure in previous chapter. In this section, we explain the characteristics of $SMPV$ structure using some figures and tables. In $SMPV$ structure, the road network is partitioned into multiple subgraphs. A road network is modeled as a directed graph $G(V, E, W)$, where V is the set of vertices (nodes), E is the set of edges (road segments) and W is the set of weights(we assume here is road network distance). A subgraph $SG_i(V_i, E_i, W_i)$ is partitioned graph of G with $V_i \in V, E_i \in E$, and $W_i \in W$. Figure 3.3(a) describes the simple graph G of a road network and (b) is the multiple subgraphs SG_i of G.

Partitioning of a road network into the subgraphs are performed by the following procedures: (1) source nodes on the given road network are selected with the specified number of divisions, (2) multiple sources Dijkstra's algorithm is used to categorize each node into a subgraph that has the same source node as nearest neighbor. In each subgraph, there are two types of nodes namely border node and inner node which are represented as black rectangle and white rectangle respectively. The definition of these nodes are described below. Two subgraphs are

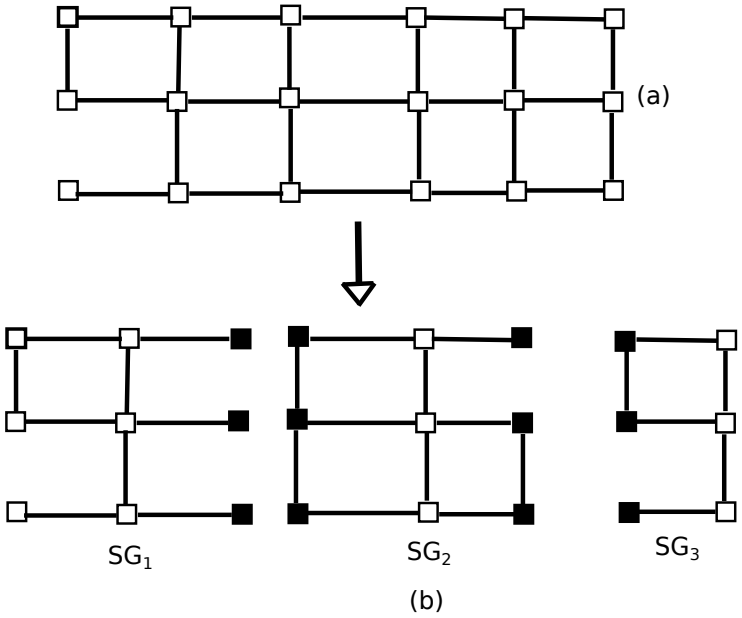


Figure 3.3: (a)The Simple graph which represents the road network. (b) The partition subgraphs of (a).

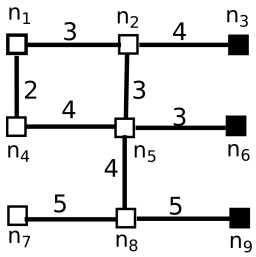
defined as being adjacent if they have at least one common border node. The set of border nodes of SG_i is denoted by BV_i.

Definition 3.1. *Border Node: The node belongs to plural subgraphs.*

Definition 3.2. *Inner Node: The node belongs to only one subgraph.*

After partitioning into subgraphs, distances within the respective subgraphs are materialized, and two distance tables called the border-to-border distance table ($BBDT$) and inner-to-border distance table ($IBDT$) are constructed. The distance calculation between two points on the road network is rendered by the best first search approach. Figure 3.4 describes $BBDT$ and $IBDT$ tables for the subgraph SG_1 of Figure 3.3(b). Figure 3.4(b) shows the $BBDT$ distance table in which the shortest path length between every pair of border nodes of the subgraph SG (in example SG_1) are listed. This table is applicable to retrieve the distances between border nodes among different subgraphs. If there is no connected path between a pair of nodes inside the subgraph, an infinity value (∞) is assigned. Figure 3.4(c) denotes $IBDT$ distance table in which the distance between inner node to border

bodes are listed. To reduce the precomputed distance data, the table for the inner-to-inner nodes is not considered and an adjacency list is used in our approach to get the road network node information.



(a) SG_1

	n_3	n_6	n_9
n_3	0	10	16
n_6	10	0	12
n_9	16	12	0

(b) BBDT of SG_1

	n_3	n_6	n_9
n_1	7	9	15
n_2	4	6	12
n_4	11	7	13
n_5	7	3	9
n_7	16	12	10
n_8	12	7	5

(c)IBDT of SG_1

Figure 3.4: (a)Subgraph SG_1 of Figure 3.3(b). (b) Border node to border node distance table of SG_1. (c) Inner node to border node distance table of SG_1.

3.2.3 MRkNN Query on SMPV Structure

The performance deficiency step of Eager algorithm is executing the **rangeNN** function at every expanded node. In our new method, **rangeNN** is invoked only on the border nodes of the subgraph which improves the performance effectively. In addition, in the Eager algorithm, verification function searches the kNN of each p ($\in P$). If q is included in the kNN set, p is determined to be an $RkNN$ of q. This check requires a wide searching area. This query can also be efficiently performed with IER using $SMPV$.

In our approach for $RkNN$ queries, the shortest paths on road network calculations are done by applying the shortest path finder [85], especially $SPFMM$ algorithm run on $SMPV$ structure. For simplicity, we assume that the object of interest is on the road network nodes. In practice, the object of interest has possibility of existing on the network edges, therefore our method can be easily extended for network edges. The procedures of monochromatic $RkNN$ query on $SMPV$ structure can be summarized as follow:

(i) When the query object q and set of objects of interest P are given, initially, the subgraph where q belongs is determined.

(ii) The objects of interest belonging to the subgraph are searched.

(iii) Next, **verify** function is invoked to check whether each object of interest is exact $RkNN$ object of q.

(iv) If the result of verify is true, the object of interest is added to the result set.

(v) The algorithm enlarges the searching area to include the neighboring subgraph and **rangeNN** is invoked at the border node and repeats the step ii, iii and iv.

(vi) If the size of candidates from **rangeNN** function is smaller than k, other $RkNN$ could exist on the path through the current node $v.n$. Therefore, the node expansion and searching will continue from the current node.

(vii) Otherwise, no more $RkNN$ exists on the path through $v.n$, the searching will be terminated.

The detail theoretical explanation and performance studies of the $MRkNN$ query on $SMPV$ structure have already described in [89] , [93]. Our method expands the search area in concentric circles, similar to the Eager algorithm and invokes the **rangeNN** only on border nodes that drastically reduce the overall processing time. In addition, IER adoption for **rangeNN** and verification function supports to enhance the performance especially when the distribution of the data points is sparse, and arbitrary k value is large.

3.3 Bichromatic RkNN Queries

$BRkNN$ query is an extended type of $RkNN$ queries to deal with the two different data types. When a set of rival objects S and a set of objects of interest P from different data types are given, a random query point q ($\in S$) is set, $BRkNN$ query

retrieves object of interest which considers query object is the nearest than other rival objects. The specific definition of $BRkNN$ is described as:

$$BRkNN(q, k, P) = \{p \in P | q \in kNN(p, k, S)\}$$

where $kNN(p, k, S)$ is a set of kNN objects in S for p.

For instance, let P be a set of supermarkets and S be a set of residence. To construct a new residence q ($\in S$), $BRkNN$ query can find all the supermarkets which are the nearest to the new residence location among other existing residence. Depending on the query result, the best location to build the new residence can be decided.

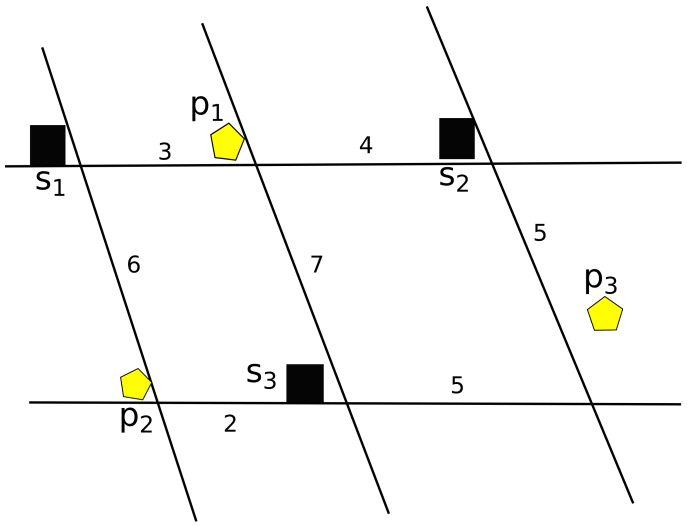


Figure 3.5: An example of the bichromatic $RkNN$ query.

Figure 3.5 shows an example of the $BRkNN$ query. Assume the rival objects are black rectangles and the objects of interest are shown by the pentagons. The $BRNN$ query finds the objects of interest (either p_1, p_2 or p_3) that the query object as their nearest neighbor among the rival objects. In this example, although the distance between the p_1 and s_2 is shorter than the distance between s_2 and p_3, the object of interest p_1 does not consider s_2 as its nearest neighbor, because p_1 is closer to s_1. Therefore, the answer of the $BRNN$ of s_2 will be p_3 which considers the s_2

as the nearest object among the other rival objects.

3.3.1 The Concept of BRkNN Query with Voronoi Diagram

Voronoi diagram (VD) [14] has been used to solve the various spatial query problems in spatial database. In general, a VD of a data point set P is a collection of regions that divide the Euclidean plane. These regions are called Voronoi polygons (VPs). Each region (VP) is associated with a data point and all the points in one region are closer to the corresponding points of a region than any other points. The Voronoi diagram can be applied to process the $BRNN$ objects. There are some previous research works [51], [61], [70] for $BRkNN$ for various environments using Voronoi diagram. When a set of rival objects S and a set of objects of interest P are given, $BR1NN$ of a query object ($\in S$) can be obtained by processing the following two steps:

1. Generate the Voronoi regions of the rival objects S.
2. Search the object of interest p ($\in P$) which are included in the generated Voronoi region of the respective query point.

Figure 3.6 illustrates the Voronoi diagram for S set in Euclidean plane. As a simple example, when s_1 is specified as the query object, the $BR1NN$ of s_1 retrieves p_1 because p_1 exists in the Voronoi region of s_1. To find the $BR1NN$ in the road network, the network Voronoi diagram (NVD) [79] for the set of rival objects S can be generated with the network distance.

The $BRNN$ results with k>1 is more complicated than the $k = 1$. For k>1, the Voronoi diagram in which a query point q is included in the set of kNN among S must be generated. Specifically, in Figure 3.6, when q is s_1 and k is 2, the searching steps for the $BR2NN$ are as follow: 1) the region in which s_1 include in the $2NN$ set is generated, and then 2) the objects of interest p ($\in P$) include in the region are reported as the result.

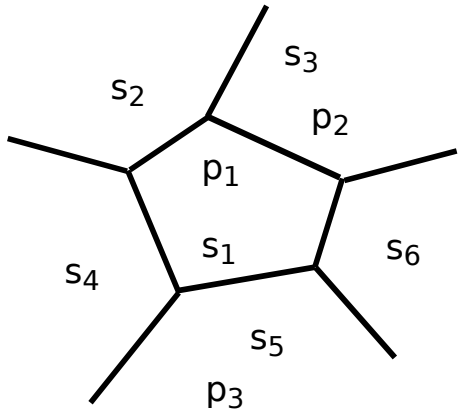


Figure 3.6: $BR2NN$ searching using Voronoi diagram

3.3.2 Our Approaches for BRkNN Query on Road Network

In this section, we mainly discuss our proposed strategy to accomplish the $BRkNN$ query on spatial road network.

BRkNN query using Eager algorithm

The Eager algorithm [47] mainly proposed for the monochromatic $RkNN$ query for the large road network, is also applicable to the $BRkNN$ query on road network. The original Eager algorithm processes $RkNN$ searches in two stages named **rangeNN**(q, n, d) and **verify**(p, k, q) as described in section 3.2.1. We adopt the original Eager approaches for searching the $BRkNN$. Algorithm 3.1 is used to find the $BRkNN$ query result.

First, the algorithm initializes a priority queue (PQ) by inserting the road network node n existing the query point q. Using the lemma 3.1 described in section 3.2.1, Eager algorithm determines the expansion process. Next, it retrieves the k nearest rival objects set performing the **rangeNN**$(n, k, d(q, n), S)$ function. If the rival objects are less than k, the algorithm expands to the neighbor nodes. Concurrently, the algorithm examines the current node n whether it is object of interest because we assumed the objects of interest are on the network node for simplicity. If the current node is object of interest p, the algorithm invokes the

Algorithm 3.1 Eager algorithm for $BRkNN$ query

```
1: function BRkNN(q)
2:     PQ ← ∅, RS ← ∅
3:     Let rec be the record of network nodes which exist q.
4:     PQ.enQueue(rec)
5:     while PQ not empty do
6:         v ← PQ.deQueue()
7:         CS.add(v)
8:         KNN ← rangeNN(v.n, k, d_N(v.n, q), PQ)
9:         if then v.c ∈ P
10:            p ← v.c
11:            if verify(p, k, q) then
12:                RS ← RS ∪ p
13:            end if
14:        end if
15:        if |KNN| < k then
16:            PQ.enQueue(ExpandNeighborNode(v.c))
17:        end if
18:    end while
19:    return RS                                   ▷ BRkNN of q
20: end function
```

verify(p, k, q, S). If $q \in kNN(p)$, p is added to the result set. The verified p is marked as verified to avoid the duplicate verification.

The main deficiency of the Eager algorithm is the huge processing time requirement when the rival objects are distributed sparsely on the road network or the k value is large. Therefore, we studied the $BRkNN$ query to overcome the above difficulties employing the following procedures:

1. Apply the IER strategy when we decide to expend the region.
2. Employ the $SMPV$ data to suppress the times of checking a road network node to determine whether it is included in the $BRkNN$ region or not.

BRkNN query on SMPV structure

When a set of rival objects S, a set of objects of interest P are given and a query object q ($\in S$) is specified, the $BRkNN$ query on $SMPV$ structure initializes to find the range R in which q is included as the kNN. Then, the objects of interest which lie in the range are retrieved. We can prune the unnecessary nodes that are not guaranteed to find the $BRNN$ of the query object q by utilizing the lemma 3.1. Although lemma 3.1 is proposed for the $MRkNN$ query on the road network, it also solves for the $BRkNN$ query in the road network.

Figure 3.7 explains the lemma 3.1 with an example using the border nodes $SMPV$ structure described in chapter 2. In this figure, the black rectangles are border nodes belonging to more than one subgraph in $SMPV$ structure. When the distance between the border node b_i and the object s_1 : $d(s_1, b_i)$ is greater than the distance between b_i and $s_2 : d(b_i, s_2)$, $BRNN$ of s_1 passing through the b_i cannot be found. Therefore, all paths passing through the border node b_i can be pruned safely. This constraint can also be applied to reduce the unnecessary node expansion. While retrieving kNN ($\in S$) of a border node (b_i), if q is not included in the kNNset of b_i, the searching for all paths which pass through b_i can be pruned.

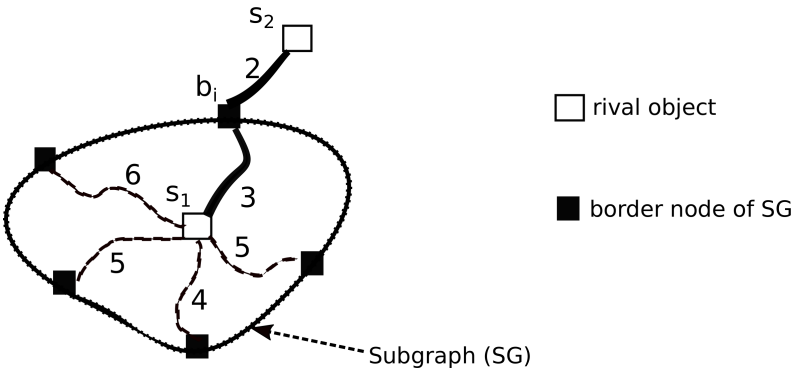


Figure 3.7: An example illustrates lemma 3.1.

We continue by explaining the detail processing of the $BRkNN$ in $SMPV$ structure. Suppose that q be a given query object and SG_q be a subgraph in which q exists. Initially, the algorithm determines the SG_q where the searching initializes.

To obtain the fast access in retrieving each subgraph information where elements of rival set S belong, the position of the rival objects s ($\in S$) is indexed by R-Tree [4]. The data points p ($\in P$) and subgraphs related table are constructed according to the subgraph in which object of interest belongs. Referring this table can reduce the processing time and node expansion dramatically. Because, we can omit accessing the subgraph in which there are no data point. Figure 3.8 shows the simple subgraph where q belongs.

In the figure, p is the data point from P set. The q object retrieves the data points belonging in SG_q then the kNN set of each retrieved data point (p) is searched. If q is included in kNN set of P, p is added to the result set. The record of each border nodes of SG_q are inserted into PQ with the following format:

$$<d, n, p, c_{id}>$$

where d is the road network distance from q object to current node n (a border node (b_i), n is the current node, p is the previous visited node (in this case, q object) and c_{id} is the subgraphID in which current node locates (SG_q in this case). The first record enqueued into PQ is composed as follow:

$$<d(q, b_i), b_i, q, SG_q>$$

Algorithm 3.2 describes the above procedure as a pseudo code.

After examining the initial step, the algorithm searches the most kNN of the current node (border node) $v.n$ that is dequeued from PQ and then kept in kNN. This function is called **rangeNN** described in Line 7 of Algorithm 3.3. The set of objects of interest C belonging to the subgraph of current of node are retrieved using **findPinSubgraph**($v.n$) instead of examining the each current node as in the original Eager approach. These objects are called candidate objects C. Each object in C is checked whether its kNN contains q or not. If $q \in kNN(p)$, $\{p \in C\}$, the

Algorithm 3.2 StartSubgrpah

```
1: procedure StartSubgrpah(q, PQ, R)
2:     sg ← determineSG(q)
3:     P ← findPOIinSG(q)
4:     for all p ∈ P do
5:         if verify(p, k, q) then R ∪ p                ▷ add p to result set
6:         end if
7:     end for
8:     for all b ∈ BN do
9:         PQ.enQueue(< d_N(q, b), b, q, sg >)
10:    end for
11: end procedure
```

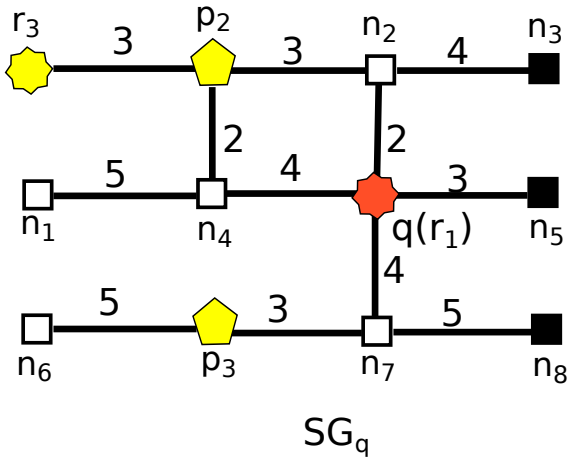


Figure 3.8: An example of $BRkNN$ query processing on SG_q

candidate object is $BRNN$ of q object. This verification is invoked in line 10. To avoid duplicate investigation in subgraph, the investigated subgraph is marked.

If the number of data points from KNN is less than k, other $BRkNN$ on the path through the current node might exist. The algorithm continues the search by invoking the **ExpandSubGraph** function from the node $v.n$ as shown in line 15 of algorithm 3.3. The detail explanation of this function is described in algorithm 3.4. In algorithm 3.4, line 2-5 collect the neighbor subgraphs and then the information of the border nodes of the subgraph are enqueued into PQ. The $BRkNN$ searching is terminated when no more $BRkNN$ exists on the path through $v.n$.

Algorithm 3.3 $BRkNN$ query algorithm with $SMPV$ structure

```
1: function BRkNN(q)
2:     PQ ← ∅, RS ← ∅
3:     StartSubgrpah(q, PQ, RS)
4:     while PQ not empty do
5:         v ← PQ.deQueue()
6:         CS.add(v)
7:         KNN ← rangeNN(v.n, k, d_N(v.n, q), PQ)
8:         C ← findPinSubgraph(v.n)
9:         for all p ∈ C do
10:            if verifyPSet(p, k, q) then
11:                RS ← RS ∪ p
12:            end if
13:        end for
14:        if |KNN| < k then
15:            ExpandSubGraph(v.n, PQ)
16:        end if
17:    end while
18:    return RS                                   ▷ BRkNN of q
19: end function
```

Algorithm 3.4 ExpandSubGraph

```
1: procedure ExpandSubGraph(q, PQ, R)
2:     SG_s ← determineNeighborSG(v.n)
3:     for all sg ∈ SG_s do
4:         for all b ∈ BN do
5:             PQ.enQueue(< d_N(q, b), b, p, v.cid >)
6:         end for
7:     end for
8: end procedure
```

3.3.3 Performance Study

Experimental Environment and Settings

Extensive experiments were conducted to investigate the performance of our proposed algorithms and existing algorithms utilizing a PC Intel Core i7-4770 CPU

($3.4GHz$) and $16GB$ memory. In addition, all the algorithms described in thesis are implemented using Java programming language using real road map data of Saitama city (small map) and Saitama prefecture (large map), Japan as described detail in Table 3.1. Table 3.2 shows the detail information of the road network nodes and subgraphs of each map for the $SMPV$ structure that is used in our studies. The objects of interest on the road network map are generated by the pseudorandom sequences. These set of data points are generated with varying the distribution (density). For example, the density of data points set D=0.002 indicates that two data objects exist on every 1000 links. Table 3.3 describes the relationship between the data points density and the number of data points in each set of data points in small map used in the all experiment of the $RkNN$ queries. Among the values, the bold values represent the default values.

Table 3.1: Road network maps used in all experiments for performance evaluation

Map Name	# of Nodes	# of Edges	Area Size	Adj. List	BBDT	IBDT
Small Map	16,284	24,914	168 km^2	1.5 MB	1.1 MB	4.1 MB
Medium Map	81,233	109,373	284 km^2	6.8 MB	4.5 MB	17.4 MB
Large Map	465,245	638,282	3,797 km^2	39.7 MB	26.1 MB	100.8 MB

Table 3.2: Data size (MB) of $SMPV$ structure for each road map

Map Name	Border Nodes	Inner Nodes	Subgraph Numbers
Small Map	1,780	14,504	100
Medium Map	7,496	73,737	350
Large Map	43,418	421,827	3000

Table 3.3: List of the density of distribution of objects utilized for performance evaluation for $RkNN$ queries

Type of object set	Setting
Number of data objects in P set (density)	24(0.001), **48(0.002)**, 124(0.005), 249(0.01),498(0.02),1245(0.05)
Number of rival objects in S set (density)	24(0.001), **48(0.002)**, 124(0.005), 249(0.01), 498(0.02),1245(0.05)

Experimental Results

In this section, we focus on the performance study of the $BRkNN$ query. Figure 3.9 compares the performance of proposed method and the Eager approach for

$BRkNN$ query when the density of rival objects set S and objects of interest set P were set to 0.002 and the number of k values was varied from 1 to 10. Figure 3.9 shows the results for small, medium and large map respectively. In these figures, the horizontal axis represents the varying k values and the vertical axis represents the processing time measured in second (s). The Eager approach for all maps increased the processing time when the k value was increased because the search areas were enlarged gradually. In contrast, although the proposed approach linearly increased with k values, it took only about one second when the k value was set to 10. According to the observation, the tendency of both approaches for $BRkNN$ query were almost the same for all road network maps.

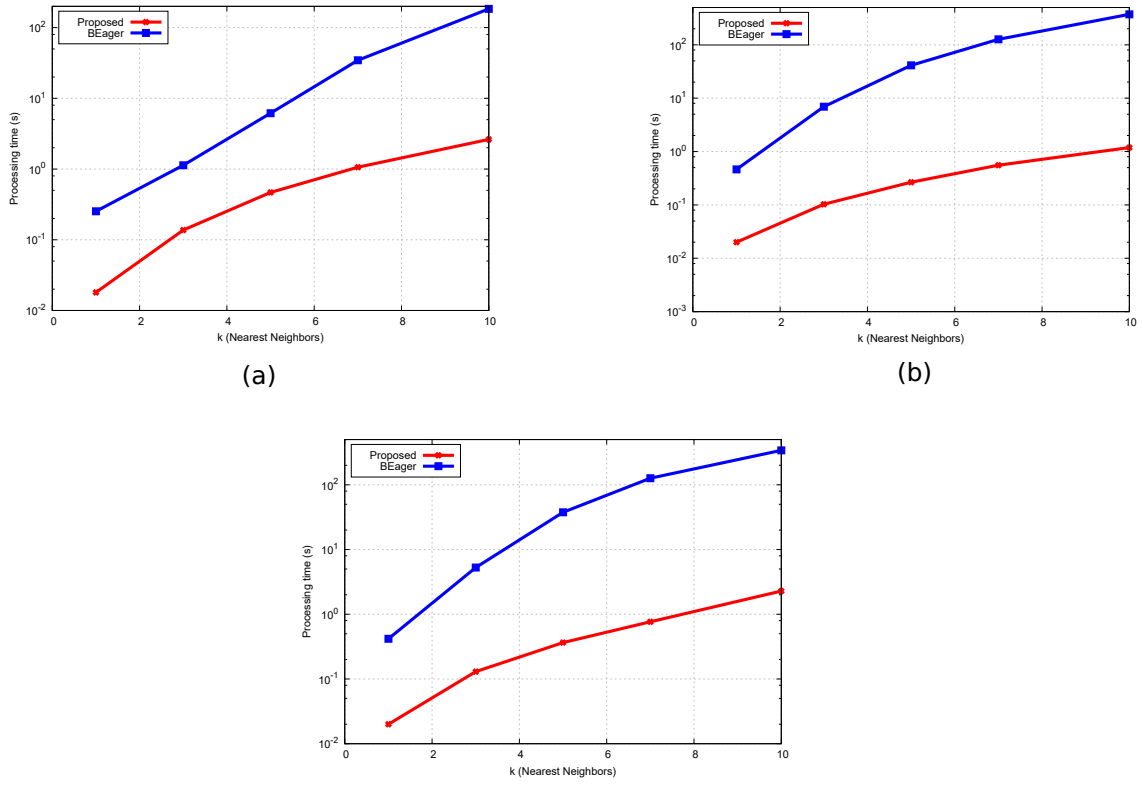


Figure 3.9: Processing time comparison for $BRkNN$ query with varying k values (a) conducted on small map, (b) conducted on the medium map, (c) conducted on the large map.

Next, we changed the experimental setting with varying objects of interest

set P density, the rival objects set S was fixed to 0.002 and k value to 3. The performance results are shown in Figure 3.10. Clearly, the $SMPV$ approach still outperformed with the lower processing time while the Eager approach was increasing sharply. We observed one thing from this experiment was the trend of both approaches were going with stable conditions. However, the processing time for $SMPV$ approach rose slightly when the density was distributed densely specifically, D= 0.05. This was due to the reason that when the candidates of $BRkNN$ were found from P, the candidate of p was required to verify whether each candidate object was truly $BRkNN$ of q or not. If the density of P was high, the existence of objects in P within the candidate subgraph might be increased and several invoking times of verification function caused increase in the processing time. Similar results were found in all maps.

We also conducted the experiments with varying the rival objects set S. We set the objects of interest set P and k value as 0.002 and 3 respectively. In this experiment, 200 query objects are randomly selected and set as a query point and the average processing time is measured as shown in Figure 3.11. We evaluated the performance for all maps. According to the experimental results, when the density of S was low, searching had to be done broad range, so the Eager algorithm needed long processing time. Conversely, when the rival objects set density was high, the searching area became smaller and the processing time decreased. The proposed approach showed the stable result means it was independent of the rival objects set density.

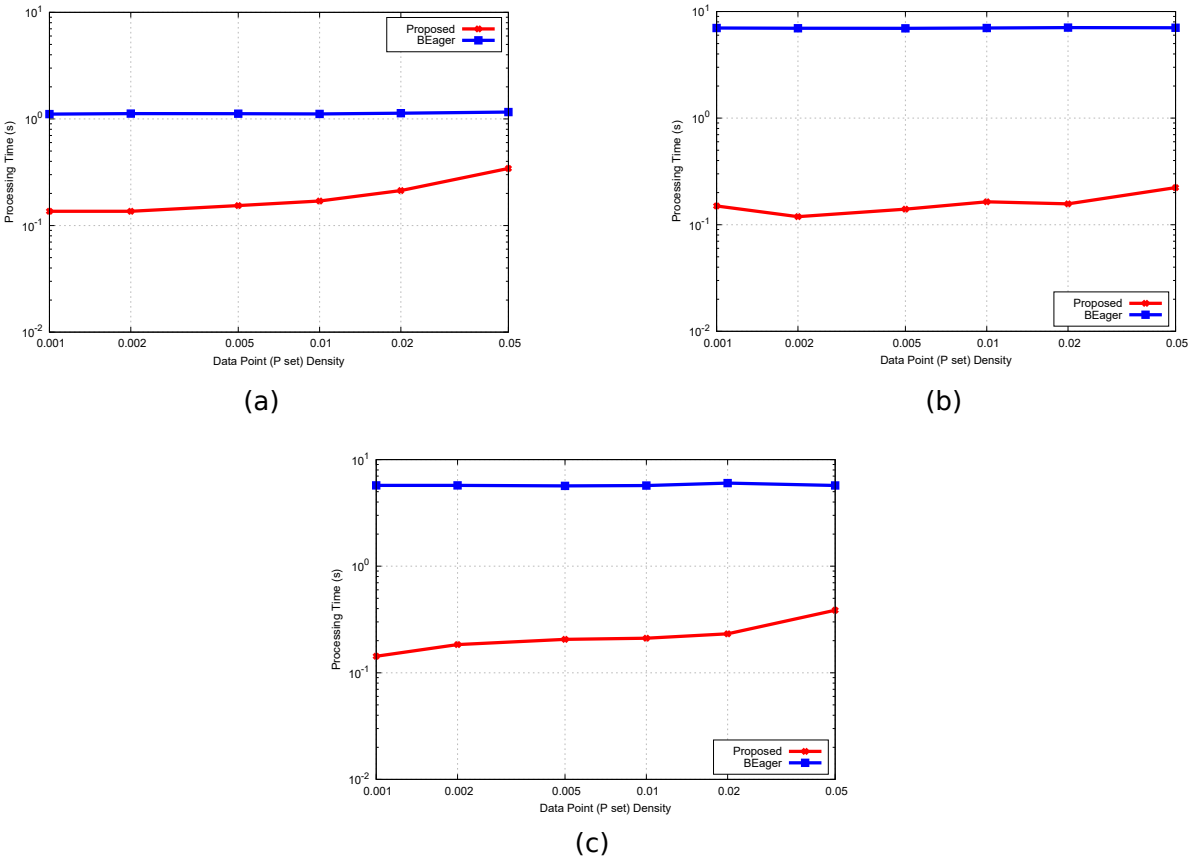


Figure 3.10: Processing time evaluation on varying the density of data point set (a) evaluated on the small map, (b) evaluated on the medium map, (c) evaluated on the large map.

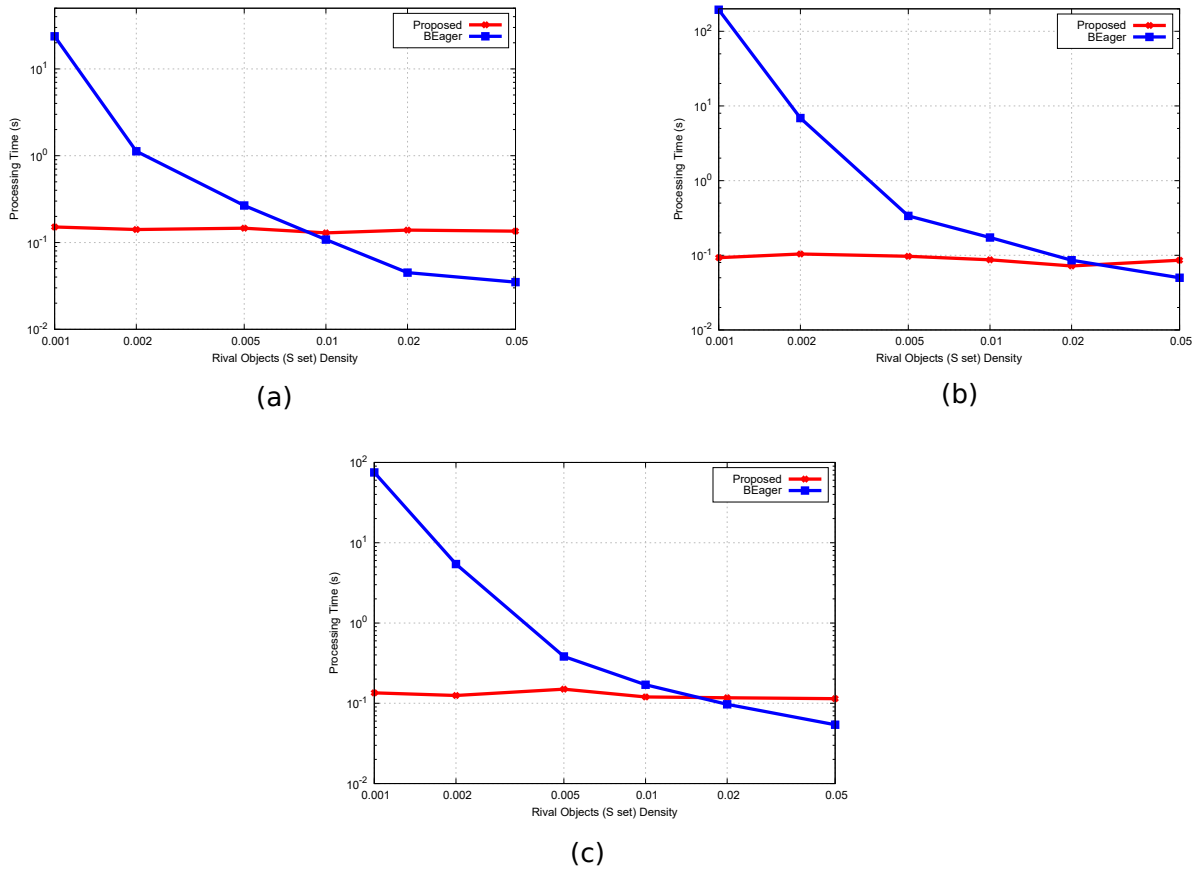


Figure 3.11: Performance evaluation with processing time on varying the density of rival objects set (a) assessed on the small map, (b) assessed on the medium map, (c) assessed on the large map.

3.4 Summary

In this chapter, we studied the algorithms for answering the snapshot type reverse k nearest neighbor ($RkNN$) queries on road network using distance pre-computation approach. Our studies covered for two types of $RkNN$ queries: monochromatic $RkNN$ and bichromatic $RkNN$ queries. Although $RkNN$ query is strongly related to the kNN query, unlike kNN query, $RkNN$ finds the set of objects of interest that have the query as their nearest neighbors. There is no fixed data size in the set $RNN(q)$ which may contain points that are not the closest point of q. Thus, generally, RNN query needs to visit all data points. Due to this reason, RNN queries on the road network distance require long processing time and cause a large number of node expansions especially when the data points are distributed sparsely. To address this problem, we presented an effective and efficient techniques by adopting the $SMPV$ structure and IER strategy for fast kNN queries.

The $SMPV$ structure is constructed with small road network subgraphs and materialized distance tables that performed on each subgraph. The usage data amount of $SMPV$ structure is lesser than the conventional hierarchical network distance materialization. Our proposed method expands the searching area in concentric circles. In our method, performing **rangeNN** function only on border nodes of subgraph and adapting IER to process **rangeNN** and **verify** reduce the overall processing time. We conducted extensive experiments to show the performance of our approaches. The experimental results indicated that our proposed approach is 10 to 100 times faster in processing time when the number of k is large and data points are distributed sparsely on the road network. For $BRkNN$ query, although, the processing time of our proposed algorithm rose slightly when the density of data points (P set) distributed densely, our algorithm was stable in both sparse and dense distribution of rival objects (S set) on road networks. Next, we extended our knowledge on spatial queries with snapshot query type to continuous query type.

CHAPTER 4

Safe-Region Generation Method for Versatile Continuous Vicinity Queries

In chapter 3, our studies mainly focused on the snapshot spatial queries with static query objects. With the upsurge in availabilities and popularities of the mobile communications and real-time location information management, scalable computational capabilities of mobile objects are becoming essential in location-aware applications. Location monitoring is an essential and complex function to process the moving queries generated by moving objects (MO). In spatial database with moving objects, a query requires to monitor and evaluate continuously according to the current location of query object. Hence, in this chapter, we broaden our studies on the spatial queries over moving objects and present our techniques on solving the continuous queries efficiently in road networks.

4.1 The Approaches for Continuous Queries

There are several approaches studied on continuous queries for moving objects in $SNDB$ since 2000s. Generally, we can classify the studies for continuous queries

based on three main categories: 1) sort of queries, 2) types of distances (Euclidean and road network), and 3) mobility of queries and data objects.

In the literature, varieties of continuous queries have been proposed (we mentioned various continuous queries for both Euclidean and road network in the chapter 2). These queries vary according to the mobility of the query objects and data objects. When we study about continuous queries, we must consider the client-server perception in term of the communication cost between server and moving object. In continuous query, mobile objects report their position to the server whenever their position changes, and the server re-evaluates and updates the query result. When the frequency of communication between the server and mobile object becomes high, the performance in monitoring declines thus, requires a more sophisticated methods to optimize the processing time.

One of the most effective optimization techniques to manage location information of the moving objects is by employing an efficient spatial indexing structures. Several processing techniques have been extensively studied utilizing spatial index structures [17], [27], [30], [86]. However, the index structure for querying the moving objects are not effective for frequent location updates. A structure should handle frequent location updates and also process the query fast enough according to the movement of the object.

To overcome this limitation, a threshold based algorithm is presented in [44] which intends to minimize the communication cost. A threshold value is transmitted to each MO and when the moving distance exceeds the threshold value, the MO issues an update. Cheng et al. [53] studied a time-based location update technique to improve the temporal data consistency for the objects by only maintaining the correctness of data that are relevant to the execution of continuous queries. However, in his method, an object sends the location updates repeatedly to the server when it is enclosed by a query region. As a result, that approach deteriorates the query performance.

The next approach for continuous queries is Q-index [23] in which queries are indexed by an R-Tree and avoids the periodical updates from moving objects by introducing the concept called safe-region. They generated the safe-region for each moving object. The moving object needs to monitor all queries whose regions intersect with its assigned domain and report to the server only when the query moves out from the region. In [29], a region based spatial queries called nearest neighbor and window queries for dynamic environment is presented. They proposed construction of the safe-region using time parameterized (TP) queries [24] uses R-Tree index structure. There are assorted studies for varieties of spatial continuous queries specifically range queries [43] [46] [73], kNN queries [57] [65], and reverse kNN queries [66] [78] [80] [92] using the safe-region approach.

Among several studies for continuous queries described previously, the methodology which guarantees not missing any changes in the result and produces the up-to-date query result is safe-region approach. In addition, safe-region does not need any assumption and the server updates require only when the query exits the valid region; this reduces the excessive overhead on the server side. The previous research works with the safe-region are mainly targeted only for the specific type of spatial query. Specifically, their approaches are covered only for range query and kNN queries. To overcome this limitation, Ohsawa et al. [95] proposed the versatile safe-region generation method for continuous queries including the $set-kNN$, $ordered-kNN$, $RkNN$ and distance range queries.

As an extended version of [95], [98] is proposed with the performance improvement. In extended version, we enhanced the methodology of determination of border point of safe-region to generate the compact and efficient safe-region. In addition, we conducted several extensive experiments with the simulation of the mobile query object. Our studies for the continuous queries are mainly targeted for queries in which query objects are moved freely while data objects are static condition which is frequently encountered in most of the location-aware applications. In the subsequent section, we discuss the basic principles of the safe-region

and proposed methodologies that are applicable for various vicinity spatial queries.

4.2 The Basic Principles of Safe-Region Generation

In general, a safe-region is a region where the query point can move without changing the query answer. A typical example of the safe-region is Voronoi diagram (VD) [14]. The VD is a well-known space decomposition determined by the distance to a given set of points. In a VD, each object of the dataset lies within a cell called Voronoi cell (Voronoi region). The Voronoi region of an object has a property that any point lies in it is always closer to that object than any other object in the dataset.

For a kNN query, Voronoi diagram can be generalized to the k^{th} order Voronoi diagram (kVD) and k order Voronoi regions (VR) can be treated as the safe-region (SR). Each cell region is associated with the set of k nearest neighbors. To illustrate with the simple example, Figure 4.1(a) shows a set of points P= $\{p_1, p_2, p_3, p_4, p_5, p_6\}$ in a 2-dimensional ($2D$) space and Figure 4.1(b) is the 1($k = 1$) order Voronoi diagram which is the simplest SR. In the above example, q falls in VR_1 and p_1 is the nearest neighbor of q. The query point remains valid as long as it stays in VR_1. This VR_1 (gray region) is called the safe-region. When the query point moves across the boundary of VR_1 to VR_5, the NN of q will change to p_5.

However, the VD has the following major limitations: 1) expensive precomputation: the kVD requires to precompute all VD cells and access to all the data points. Both computation and storage cost are high. 2) The VD cannot deal efficiently the update operations. The original concept of VD is not feasible to generate the dynamic safe-region efficiently. Thus, novel methodologies for safe-region generation that are adaptable for the multiple types of dynamic continuous queries such that it can avoid unnecessary location probes to the objects in the vicinity (i.e., the objects which overlap with the current query region) are essential. In this chapter, we propose the safe-region which can be used for various vicinity queries.

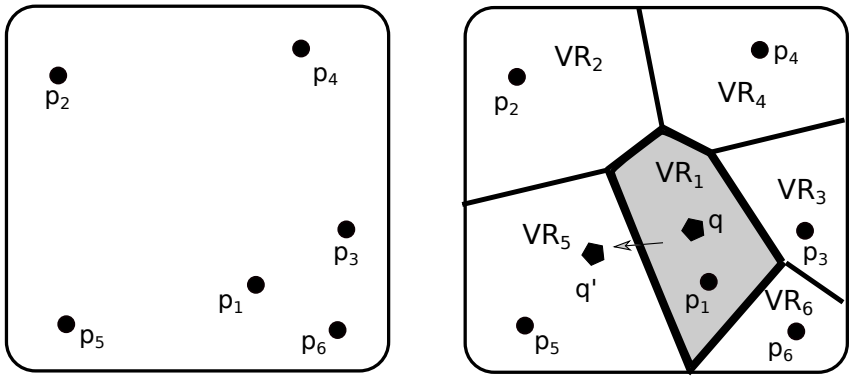


Figure 4.1: (a) A set of data points P. (b) A simple safe-region with the Voronoi diagram.

4.3 Safe-Region for Vicinity Queries

Before we discuss about the safe-region for vicinity queries, we will explain about what are the vicinity objects and queries. Generally, the objects which lie within the current region specified by the query object are called the vicinity objects. The queries which probe these vicinity objects are called the vicinity queries. Remarkably, the results of all the vicinity queries have the mutual results between them. The example of the vicinity queries is explained with Figure 4.2.

Figure 4.2 demonstrates four types of vicinity queries namely distance range query, $set-kNN$ query, $ordered-kNN$ query and $RkNN$ query. In Figure 4.2(a), the data point p_1 to p_4 belongs to a data object set P, and a to d represent the query objects. We deal with the 2 nearest neighbors ($2NN$) of each query point. For query point a, the $2NN$ are p_1 and p_4, and for b are p_2 and p_1. When the order of the query result is taken into account which is called the $ordered-kNN$ query, the result of query a and b are different. Otherwise, the order of the query result is ignored which is called the $set-kNN$ query, the query results of a and b are the same with any order of the result. This is the main difference between the $set-kNN$ and $ordered-kNN$ queries. Furthermore, the result of $2NN$ of the

query object c is p_4 and p_1; consequently, p_1 is a common data object for all $2NN$ of the query a, b and c.

When the p_1 is set as a query object and invokes $2NN$ query from p_1, a, b, and c will be included in the result as shown by thick circles in Figure 4.2(a). This type of query is called the reverse kNN query. In $set-kNN$ and $ordered-kNN$ queries, the space is partitioned into non-overlapping regions. Contrarily, the space is divided into mutually overlapping regions in $RkNN$ query as shown in Figure 4.2(a). The next vicinity query is called the distance range query as shown in Figure 4.2(b). In the Figure 4.2(b) x, y, and z are query points and p_5 to p_7 are data objects. The radius of dotted area which centers of each data objects is r. The area in bold line shows the common region in which the range query result is p_5, p_6 and p_7. Therefore, if a object lies inside this region and invokes distance range query from these objects, each query will give the same query result.

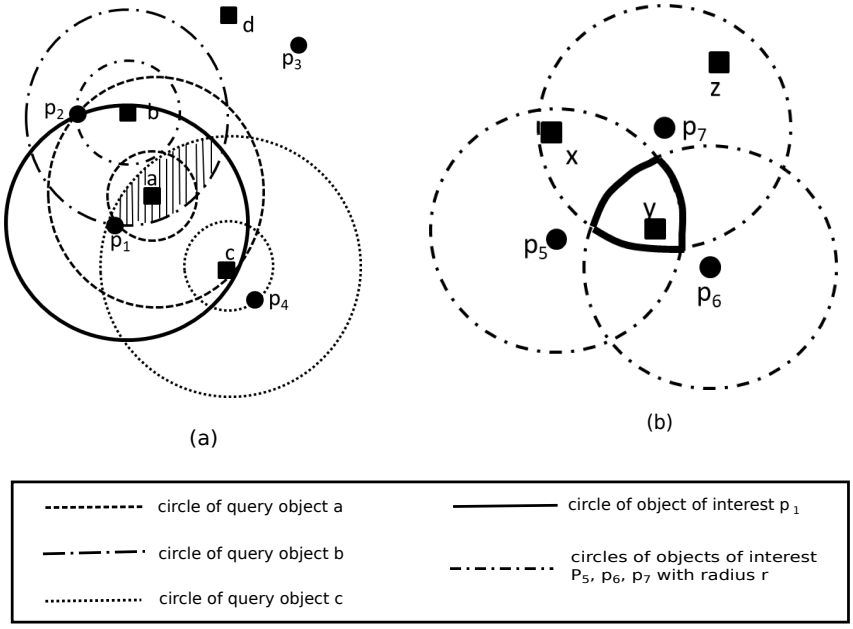


Figure 4.2: (a) Examples of the vicinity queries including $set-2NN$, $ordered-2NN$ and $R2NN$ queries. (b) Example of the distance range query.

Our objective is to generate a safe-region for these vicinity queries to apply in a dynamic environment. Here, we outline some definitions for the safe-regions of

each vicinity query.

Definition 4.1. *Safe-region (SR) for vicinity queries. SR is a region around the current location of moving object (MO) in which the query result is same as long as MO lies inside. A query on a road network probes the objects that lie inside the query's current safe-region whose query result is always same is called a vicinity query. This type of queries includes set − kNN, ordered − kNN, RkNN and distance rang query.*

Definition 4.2. *Safe-region for set − kNN query (SR_s). SR_s is a region in which kNN query is invoked anywhere inside the region returns the same set of data points with any distance order.*

Definition 4.3. *Safe-region for ordered − kNN query (SR_o). SR_o is a region in which kNN query is invoked anywhere inside the region returns the same result with the same distance order.*

Definition 4.4. *Safe-region for RkNN query (SR_r). SR_r is a region in which kNN query is invoked anywhere in the region always contain a specified given data point.*

Definition 4.5. *Safe-region for distance range query (SR_d). SR_d is a region where distance range query with the distance (r) and a query point is invoked anywhere in SR_d contains the same set of data points.*

According to the above definitions 4.2 to 4.4, we can define the vicinity queries relationship based on the kNN queries (except the distance range query). The $set - kNN$, $ordered - kNN$ and $RkNN$ queries satisfies the following condition:

$$SR_o \subseteq SR_s \subseteq SR_r$$

In addition, according to the above condition, we can estimate the size of the safe-region of each vicinity query; for instance SR of the set and ordered kNN is the subset of SR of the reverse kNN means all objects in set and ordered kNN queries are contained in the set of $RkNN$. The size of the SR of $RkNN$ will be

the largest among three kNN queries. The SR size of $ordered - kNN$ query will be the smallest. The proof of the SR size of each query is presented in the next sections.

4.4 The Strategy and Architecture of Our Proposed Safe-Region Generation for Vicinity Queries

In this section, initially, we present the processing flow of each vicinity queries, then we discuss our strategies for safe-region generation applicable for the vicinity queries and some requisites of our approach.

4.4.1 Distance Range Query

Range queries arise naturally and frequently in location-aware applications such as a query that applies to keep track the number of people who entered a specific region (e.g. building). Several processing techniques for range query processed with both static and dynamic objects have been proposed (described in previous chapters). There are several variations of the range queries. Given a set of data object P and a query object q, a query search space around the query object (such as taxi, building) is called the circular range query. Another variation of the range query which returns the objects that lie within the rectangle around the query location is called the rectangular range query.

In our strategy, we consider a set of data object P, a mobile query object q and a positive range value r (network distance). Specifically, our strategy on range query is mainly based on the network distance between data object and q that lies within the range r i.e., every objects that $dist(p, q) \leq r$. Thus, such query is called distance range query. R-Tree [4] is utilized to index the data objects in our approach.

Algorithm 4.1 explains the processing procedure of the distance range query

using a pseudo code. This algorithm retrieves the nearest data objects within the specified range r. In algorithm, we need to define two types of objects called internal objects and external objects. The purpose of defining these objects is to define the border point of the safe-region. The detail explanation of the border point and safe-region construction are presented in section 4.4.3 and 4.4.5. The objects which lie inside the range r are called the internal objects and only internal objects will contribute as the actual results of the distance range query. The objects which lie outside the range r are called the external objects. However, when the number of external objects will be arbitrary large. For such a case, in order to restrict the space, only the objects within the distance $2r$ of the query are defined as the external objects.

Line 2 and 3 initialize the auxiliary list to store the internal and external objects. Each internal and external objects are stored as a $A*$ record including the data point p and the minimum shortest distance between q and p. Line 4 retrieves the nearest neighbor of query point and finds the minimum network distance between NN and q using $A*$ algorithm (Algorithm 4.3) stated in line 7. We state the original $A*$ approach in Algorithm 4.3. Later, we describe how we can enhance $A*$ for performance issue. The NN object that lies inside the range r is stored in InObj set. In line 8 to 20, the algorithm retrieves incrementally the other possible data points inside the range r. If there are no other data points in range r, the incremental data point searching probes inside the range $2r$ and stores the objects as the external objects. After accumulating all internal and external objects, the internal objects are returned as the result data objects.

4.4.2 Continuous kNN query on Road Network

The common tasks for the vicinity queries regarding spatial network are finding the k nearest neighbor from the query object and finding the shortest path between any two points. The existing algorithms for kNN queries were described in Chapter 2. To the best of our knowledge, Papadias et al. [26], for the first time, introduced

Algorithm 4.1 Distance Range Query Algorithm

```
1: function DistanceRange(q, r)
2:     InObj ← List < AStar > ()
3:     ExObj ← List < AStar > ()
4:     p = Euclidean − NN(q)
5:     CAStar cas=new CAStar(p,q)
6:     H_a ← cas.PriorityQueue
7:     Dist=cas.ShortestPath(q,H_a)                    ▷ Algorithm 4.3
8:     while Dist<r do
9:         InObj.add(cas)
10:        p = Euclidea − NN(q)
11:        cas=new CAStar(p,q)
12:        H_a ← cas.PriorityQueue
13:        Dist = cas.Shortestpath(q, H_a)             ▷ Algorithm 4.3
14:    end while
15:    while Dist ≤ 2r do
16:        ExObj.add(cas)
17:        cas=new CAStar(p,q)
18:        H_a ← cas.PriorityQueue
19:        Dist = cas.ShortestPath(q, H_a)             ▷ Algorithm 4.3
20:    end while
21: end function
```

some frameworks for nearest neighbor queries in network space by defining an architecture that integrates the Euclidean and spatial network information. One of the framework called the Incremental Euclidean Restriction (IER) can achieve high efficiency to probe the k candidate data points.

In our approach, we adopt the IER strategy to search kNN data points for a moving query. Further, we assume there is a spatial index: R-Tree for the data objects which is useful for all query types. Initially, the algorithm retrieves the number of k candidates' data point in Euclidean distance referring the R-Tree index. And then, the road network distance between each candidate data point and the query point is obtained by $A*$ algorithm. Each data point with the minimum network distance are sorted in ascending order according to their distance and then data points are copied into top_k data point list. The auxiliary vicinity object

set ($AVOS$) with vicinity object of each k candidates' data point are created for efficient processing of the mobile query object.

Next, the query algorithm confirms the top_k list by performing the following iteration. Let the k^{th} nearest data point from the top_k list be p_k and the distance between p_k and query object be $Dist_k$. The algorithm probes the next possible nearest data points in R-Tree until $d_E(q, p_i) > Dist_k$ is satisfied. We can summarize the repetition process as follow:

1. Search i^{th} ($i>k$) nearest data point p_i in Euclidean distance.
2. Create the $A*$ objects of i^{th} nearest data point and store in the auxiliary vicinity object set.
3. Calculate the road network distance between the query object and p_i.
4. If the network distance of the i^{th} data point is shorter than the $Dist_k$, the top_k will be updated with the data point having the minimum network distance.

When the next nearest data point does not satisfy the condition, the final top_k set is confirmed as an optimal kNN result of the query object and the result is returned. The processing procedures described above are stated as pseudo code in Algorithm 4.2

Next, we continue the discussion of the calculation of the road network distance using the $A*$ algorithm in continuous kNN queries. Algorithm 4.3 outlines the pseudo code of the $A*$ algorithm to find the network distance between q and data point p. We assume that two data point q and p on a road network are given and the algorithm returns the minimum network distance $d_N(q, p)$. The $A*$ algorithm employs a heap (H_a) to extract the shortest path exploration. The heap H_a manages the record with the following format: $< h, n, d >$ where n is the current notice node, h is the heuristic cost calculated by $d_N(q, n) + d_E(n, p)$, and d is the road network distance from q to n. Records in the heap are ordered by the heuristic value. Initially, in line 2, the closed set CS_a is initialized. Next, the records from H_a are

de-heaped. Line 4 calls the **deleteMin()** function which returns the record with the minimum h value.

In Algorithm 4.3, we set the closed set CS_a which keeps the expanded record once as describe in line 9. To expand the neighbor node, **expandNode** function is executed in line 10. The expanded nodes comprise a new record with the heap format and en-heap in H_a outlined from line 11 to 13. The algorithm returns the distance when the expanded node reaches the given data object p and then the algorithm is terminated.

Algorithm 4.2 $CkNN$ Algorithm

```
 1: function CkNN(q, k)
 2:     *q is the query object & k is the number of NN objects
 3:     AVOS ← ∅, top_k ← ∅
 4:     NN_set p_1, .., p_k = Eculidean − NN(q, k)
 5:     p_next, d_E(p_next, q) = next − Euclidean − NN(q)
 6:     d_Emax = d_E(p_next, q)
 7:     for all p ∈ NN_set do
 8:         AStar as=new AStar(p,q)
 9:         H_a ← as.PriorityQueue()
10:         AVOS.add(as)
11:         d_N(p, q) = as.ShortestPath(q)                    ▷ Algorithm 4.3
12:         top_k.add(p.d_N(p, q))
13:     end for
14:     repeat
15:         p_next, d_E(p_next, q) = next − Euclidean − NN(q)
16:         d_Emax = d_E(p_next, q)
17:         AStar as=new AStar(p_next, q)        ▷ initialize object for A* Algorithm
18:         H_a ← as.PriorityQueue
19:         AVOS.Add(as)
20:         d_N(p_next, q) = as.ShortestPath(q, H_a)          ▷ Algorithm 4.3
21:         if d_N(p_next, q) < d_N(q, p_k) then   ▷ next NN is closer to q than k^th NN
22:             Replace k^th NN (p_k) with p_next
23:         end if
24:     until top_k[k].dist > d_Emax
25:     return top_k
26: end function
```

Algorithm 4.3 $A*$ algorithm

```
1: function ShortestPath(q, H_a)
2:     CS_a ← ∅
3:     while H_a is not empty do
4:         r ← H_a.deleteMin()
5:         if CS_a.contains(r.n) then continue
6:         end if
7:         if r.n == p then break
8:         end if
9:         CS_a.add(r)
10:        A ← expandNode(r.n)
11:        for all a ∈ A do
12:            Dist_a ← r.h + |l(r.n, a)|
13:            H_a ← H_a ∪ < Dist_a + d_E(a, p), a, Dist_a >
14:        end for
15:    end while
16:    return r.d
17: end function
```

4.4.3 Typical Principle of Safe-Region Generation

In order to compute the safe-region, firstly, we assume a large road network as an undirected graph modeled by $G(V, E, W)$. V is the set of nodes (intersection) ($v \in V$), E is the set of edges (road segments) ($e \in E$), and W is the set of edge weights ($w \in W$). The edge weight is assumed as the length of the road segment in our studies. Although, we assumed the road network as the undirected graph where $(v_1, v_2) \in E \Leftrightarrow (v_2, v_1) \in E$ for the simplicity of the explanation, we can apply for the directed road networks where edges have a specific direction as well.

As mentioned, the process in generating a safe-region is the same as the generation of a high order Voronoi region on the road network map. As a simple situation, a moving object continuously monitors two nearest neighbor ($2NN$) data objects while it is moving on a road network. This simple situation is illustrated in Figure 4.3. In the figure, a moving object q is represented as a white rectangle, the white circles denote the data objects from p_1 to p_6. The query object probes

the $2NN$ data points and generates the safe-region which is bounded within the area represented by symbol x. In this example, the $2NN$ results of q are p_1, p_2 and the safe-region is the part of the road segments shown by thick lines. All possible places inside SR guarantee the $2NN$ result of q remains the same. Consequently, we can formulate the query result in SR as follow:

Definition 4.6. *Query Result (QR). QR is a collection of data points that satisfy the query condition, which remains the same inside the safe-region.*

Consider Figure 4.3, p_1 and p_2 are the members of QR. The objects p_3, p_4, p_5 and p_6 are the objects that lie nearer to the boundary of the safe-region than others are called the rival objects. We define the rival objects as follow:

Definition 4.7. *Rival Objects (RO). RO are the objects that determine the size of the safe-region, in other words, they determine the border points of SR.*

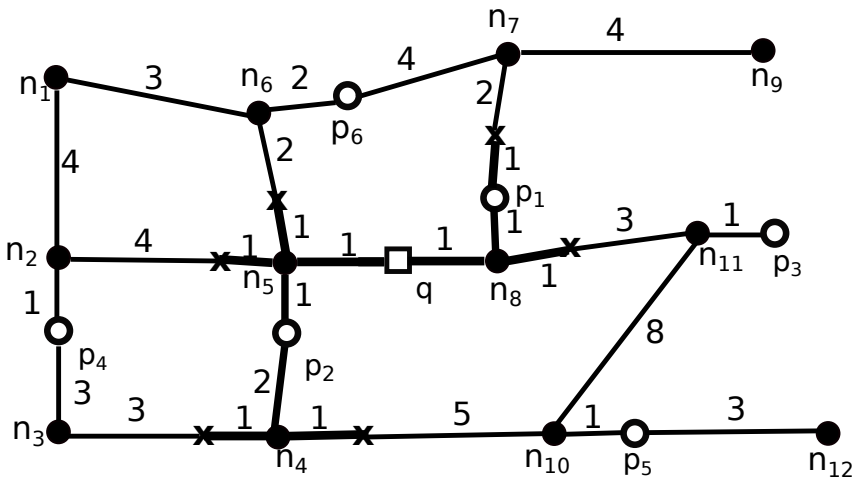


Figure 4.3: An example of a safe-region for $set - 2NN$ query.

We continue with explaining the generation of the SR. SR generation starts from the current position q, and gradually expands the region on the road network until the QR result is same when query searching at the node v ($\in V$). The region is expanded from the query point to the adjacent nodes similar to the Dijkstra's shortest path finding algorithm. The records of expanded nodes are inserted into

the heap H with the following format:

$$<\delta, n, p, l(p, n)>$$

where δ is the distance from q to n, n is the current notice node, p is the previous visited data point and $l(p, n)$ is the road segment between p and n. Heap is ordered ascendingly with the minimum distance δ value.

In Figure 4.3, the $2NN$ of q is searched on road network, and we obtain the result p_1, p_2 in QR by using the $CkNN$ algorithm described in previous section. The purpose of the rest of the operation is to find the SR in which the $2NN$ query result is the same with QR and is defined as a SR of q. To construct the SR, the road network nodes and road segments have to be expanded from the query object gradually. The expanded nodes and segments return the same query result with QR when $2NN$ is invoked from these nodes and segments are considered for SR. In the example, the two records corresponding to two end nodes of the link on which q exists are inserted into H. They are

$$<1, n_8, q, l(q, n_8)>, <1, n_5, q, l(q, n_5)>$$

Then, the record with the minimum cost is de-heaped from H. Thus, initially $<1, n_8, q, l(n_8, q)>$ is obtained from H. The cost of the two records in H are the same in this example. However, we assume $<1, n_8, q, l(n_8, q)>$ is de-heaped here. And then, the $2NN$ at n_8 is searched and checked whether the $2NN$ at n_8 is the same with QR or not. Here, the $2NN$ queried at n_8 gives the same result with QR of q, therefore n_8 also lies in $SR(q)$.

Similarly, the algorithm continues to search range expansion at the adjacent nodes of n_8, then n_7 and n_{11} in Figure 4.3, and inserted into H.

$$<5, n_7, n_8, l(n_8, n_7)>, <5, n_{11}, n_8, l(n_8, n_{11})>$$

Here, we highlight the first value in the en-heaped records. Those values are the δ values is the road network distance between q and n. In this stage, δ is the sum of the values of the previous de-heaped record and current record. For instance, for node n_7, δ is 5 means the sum of the distance between q and n_8 is 1 and the distance between n_8 and n_7 is 4. Similarly, $<1, n_5, q, l(q, n_5)>$ is de-heaped from H and the similar distance calculation is applied at the node n_5. For the node n_5, the following records are inserted into H:

$$<4, n_4, n_5, l(n_5, n_4)>, <4, n_6, n_5, l(n_5, n_6)>, <6, n_2, n_5, l(n_5, n_2)>$$

When the algorithm de-heaps the next record $<4, n_4, n_5, l(n_5, n_4)>$, the $2NN$ are searched at node n_4. Since the $2NN$ result at node n_4 is the same as the QR, the whole segment of $l(n_5, n_4)$ is occupied in SR. Subsequently, beyond the node n_4, the next two records $<8, n_3, n_4, l(n_4, n_3)>$ and $<10, n_{10}, n_4, l(n_4, n_{10})>$ are inserted into H. The next de-heap record will be the $<5, n_6, n_5, l(n_5, n_6)>$. For the node n_6, the $2NN$ result does not return the same region as the QR. Therefore, we state the whole segment $l(n_5, n_6)$ is not included in SR and the border point of the SR will exist on the segment $l(n_5, n_6)$. To define the border point on a segment, we state the following definition.

Definition 4.8. *Border point of SR (BSR). BSR is a point x on a segment. Specifically, the place on a segment that satisfies the equation $d_N(x, QR_f) = d_N(x, RO_n)$.*

Where $d_N(x, y)$ shows the road network distance between x and y. QR_f is the furthest data point from x in query results (QR), RO_n is the nearest data point to x in rival objects (RO). In this example of Figure 4.3, QR_f is p_2 and RO_n is p_3. By repeating the same process, all border points shown by x are determined.

4.4.4 Algorithms for Generating Safe-Region

The overall principle of the safe-region generation described in previous section is outlined as a pseudocode in Algorithm 4.4. The heap H in line 4 controls the region

expansion and the initial records at q are inserted into H. In line 5, a closed set (CS) is prepared to keep the road segment to avoid the duplicate check and RS is the result set of the road segment included in the safe-region. In line 6, **INITIALSET** function is executed to retrieve the kNN at q and the result is updated in QR. This function mainly uses the $CkNN$ algorithm to retrieve the NN result. In the case of set-$2NN$ query, the $2NN$ at q is set to QR (in this case, p_1 and p_2). The procedure of the **INITIALSET** function for kNN based vicinity query is same and only range query is differently implemented. The detail of this function for general query types is explained in section 4.4.5.

Next, the algorithm iterates the H until the size of H becomes empty to generate the safe-region edges on the road segments from line 7 to 22. Initially, a record with the minimum cost is de-heaped from H, and the road segment of the record, $r.l$ is checked whether it is already registered in CS. If $r.l$ is in the CS, it means that the segment has already been processed, and the rest of the processing steps are skipped. Otherwise, $r.l$ is added into the CS as stated in line 11. The node from the current record should validate with the QR so as to define whether the road segment occupy in the SR or not. In line 12, the current node $r.n$ is checked whether the query result at the node satisfy the query condition using the function called **VERIFY**.

The query condition is that the result of kNN from the current node ($r.n$) should be the same as the values in QR. However, **VERIFY** function will be varied and implemented according to the vicinity query type. For example, for ordered-kNN, if the query results from every expanded node ($r.n$) are identical, the node $r.n$ can expand to the next neighbor node for SR generation and the expanded segments are included in SR. The adjacent road segments of the current node are searched by referring to the adjacency list (see the line 13). The records for all adjacent road segments are composed and en-heaped into H as outlined in line 14 and 16. When the **VERIFY** function returns false, another function **ADDWITHCHECK** calculates the part of the edge where the verify condition

is satisfied and the satisfied parts is added into RS. The **ADDWITHCHECK** function is presented in detail in the next section.

Algorithm 4.4 Safe-region Generation Algorithm

```
1: function SRG(q, k)
2:     Input: q(current position of query object)
3:     Result: safe-region
4:     H ← ∅
5:     CS ← ∅, RS ← ∅
6:     CS ← INITIALSET(q)
7:     H.enHeap(0, q, null, 0)
8:     while H ≠ ∅ do
9:         r ← H.deleteMin()
10:        if CS contains r then continue
11:        end if
12:        CS ← CS ∪ r.l
13:        if VERIFY(r.n,QR) then
14:            nodes ← AdjacentNode(r.n)
15:            for all n ∈ nodes do
16:                H ← H∪ < r.d + |r.l|, n, r.n, n.l >
17:            end for
18:            RS ← RS ∪ ADD(r.l)
19:        else
20:            RS ← RS ∪ ADDWITHCHECK(r, k)
21:        end if
22:    end while
23:    return RS
24: end function
```

4.4.5 Variation for Vicinity Queries

The principle and algorithm for the safe-region generation applicable for all vicinity queries are presented in previous sections. We described some functions that are executed while constructing the SR such as **INITIALSET**, **VERIFY** and **ADDWITHCHECK**. These functions have some variation for each vicinity query. In this section, these variation are discussed.

In $set-kNN$ and $ordered-kNN$ query, **INITIALSET**(q) function returns

the kNN objects of q. This function is simply achieved by executing the $CkNN$ algorithm 4.2 described above. In this function, we retrieve the k nearest objects of the query object without considering the set or ordered or reverse value. After collecting the kNN objects from the **INITIALSET** function, the **VERIFY** function executes to verify vicinity query result separately. The function **VERIFY**(n, QR) checks whether the kNN query result at node n accords to the result at q (the objects in QR). To satisfy with the QR, the verify function requires to implement differently according to the principle of each vicinity query.

Next, we discuss about the distance range query. The range query algorithm initializes using the **INITIALSET**(q) function. In this function, a specified distance range D is given for instance $1km$. This function returns the data objects lying in a distance D from q. For the range query, the VERIFY(n, QR) function requires the distance parameter in function as **VERIFY**(n, QR, D). This function searches the data objects located in the distance range D from q, and returns true if the result set exactly corresponds to QR. If other objects appear in the range from a node or some objects disappear from the range, **VERIFY** returns false. The **INITIALSET** and **VERIFY** functions in Algorithm 4.4 can be relevant to generate the safe-region for any vicinity queries with minor modifications such as adding the necessary parameters.

4.4.6 Determination of the Border of Safe-Region

In previous section, we stated that the safe-region is a collection of the road network segments on which the query result remains the same with the value in QR of mobile query object. Based on the discussion of the previous section regarding the verification phase at the road network nodes, if the query condition is satisfied at the node, we assumed the whole road segment ended at the node lies inside the safe-region. On the contrary, when the query result at the expanded node does not return the same result with QR of q, we described that the border of the safe-region exists on the edge means the safe-region area will be completed at the border point

without further expansion beyond the border.

In section 4.4.4, we defined the border point of a safe-region for all vicinity queries. Algorithm 4.5 summarizes the calculation for the border position invoked by the Algorithm 4.4. Initially, we need to collect the internal and external objects from the node that do not satisfy the query condition. The searching of the internal and external objects in Algorithm 4.5 is applied only for kNN based vicinity queries. For the distance range query, the internal and external objects are searched in initial phase as explained in section 4.4.1.

For kNN based vicinity queries, the internal objects are searched via the previous node and the external objects are retrieved from the current node as stated in line 2 and 3. Alternatively, the internal objects are the objects from QR because they are probed from the previous node (node that satisfied query condition) and the external objects are the rival objects (RO). The minimum distance from external objects D_{ExMin} and the maximum distance from Internal objects D_{InMax} are examined in line 4 and 8. Line 10 examines the border position which satisfy the definition 4.8. For distance range, **ADDWITHCHECK** function is needed to implement with different parameter as outlined in Algorithm 4.6.

For a distance based range query, the algorithm checks the maximum network distance between the node that satisfies the query condition for each internal object as described in line 7. Then, line 9 calculates the InSplit position on the segment (pl) which exists within the distance D minus the maximum distance of the internal object to get the lower bound value. Similarly, to define the OutSplit position, the algorithm examines the minimum network distance between the node and each external object as described in line 11. In line 13, the OutSplit value is calculated by extracting the range distance from the summation of the segment length and minimum network distance. Finally, the actual split position (border position) is defined with the minimum value of the InSplit and OutSplit in line 14. In line 15 and 16. **ADDWITHCHECK** function returns the valid segment to SRG Algorithm

4.4.

Algorithm 4.5 Find the valid segment (kNN based Vicinity queries)

1: **function** ADDWITHCHECK (r, k)
2: Input:the record of the node that do not satisfy the query condition
3: Output:the valid segment
4: Polyline pl ← readPolyline(r.n) ▷ retrieve the segment that the node exists
5: $previous \leftarrow r.getPrevious()$
6: $current \leftarrow r.getCurrent()$
7: $InObjp_1, ..., p_k \leftarrow CkNN(previous, k)$
8: $ExObjp_1, ..., p_k \leftarrow CkNN(current, k)$
9: $D_{InMax} \leftarrow GetMaxDistance(InObj)$ ▷ $QR_f(x)$
10: $D_{ExMin} \leftarrow GetMaxDistance(ExObj)$ ▷ $RO_n(x)$
11: $borderPosition = ((D_{InMax} + D_{outMin} + pl)/2) - D_{InMax}$
12: $Segment = pl.split(borderPosition)$
13: **return** $Segment$
14: **end function**

4.4.7 Improvement on Query Processing Time

In this section, we present an additional enhancement technique to improve (reduce) the query processing time. The objects of interest on road network are typically restricted by an underlying road network. Considering that when the data point distribution is extremely sparse on the road network, it can suffer from long processing time especially when we calculate the road network distance between objects. In our study, the road network distance is calculated using $A*$ algorithm (Algorithm 4.3). As a drawback of $A*$, when data points distribution is only one side of the query object, the search areas overlap with each other. As we observed that a node is visited several times while computing the distance when we search the kNN of query object. Consequently, the efficiency of A* algorithm deteriorates and it effects the overall query processing. To improve the performance issue, we applied the idea of the single source multi-targets $A*$ ($SSMTA*$) [84]. The original $SSMTA*$ algorithm concurrently finds the shortest distance from a source(q) to multiple target objects T. Basically, the processing flow of $SSMTA*$ algorithm is

Algorithm 4.6 Find the valid segment (distance range query)

```
1: function ADDWITHCHECK (r, D)
2:     Input:the record of the node that do not satisfy the query condition
3:     Output:the valid segment
4:     Segment pl ← getSegment(r.n) ▷ retrieve the segment that the node exists
5:     current ← r.getCurrent()
6:     for all p ∈ InObj do                          ▷ Get InObj from Algorithm 4.1
7:         D_InMax ← GetMaxDistance(InObj)                              ▷ QR_f(x)
8:     end for
9:     InSplit = D + pl − D_InMax
10:    for all p ∈ ExObj do                          ▷ Get ExObj from Algorithm 4.1
11:        D_ExMin ← GetMaxDistance(ExObj)                              ▷ RO_n(x)
12:    end for
13:    OutSplit = D_ExMin + pl − D
14:    borderPosition = GetMin(InSplit, OutSplit);
15:    Segment = pl.split(borderPosition)
16:    return Segment
17: end function
```

the same as the $A*$ algorithm (chapter 2).

In Algorithm 4.4, $CkNN$ query is invoked a large number of times. Firstly, the query result (QR) with kNN objects set is initialized in the function called the **INITIALSET**. In the following steps, $CkNN$ algorithm is called by **VERIFY** function to construct the verified region as a safe-region. In our approach, we utilized the IER framework exploits kNN query in road network by integrating the $A*$ algorithm. Although, $A*$ algorithm can perform efficiently to search the shortest path, when it is repeatedly invoked a large number of times, it causes a drastic increase in processing time. One crucial concern is applying a runtime distance materialized path view to upgrade the performance especially in the verification phase.

In the **VERIFY** function, the road network distance between each candidate data point and a noticed node is calculated. The number of candidate data points is not less than arbitrary k value and the number of invoking time of the **VERIFY** function is same as the addition of the number of the nodes in the created SR and

the nodes environs with the SR. This invoking process can be found when a source position of the candidate data points set is fixed and the target nodes are changing in a similar way of the $SSMTA*$.

The $A*$ algorithm repeatedly explores the search area in the road network for each candidate data point. There is likelihood of checking the same node from the different candidate. Therefore, we keep the expanded network nodes in the closed set (CS) to reuse the processed records in $A*$ algorithm. Once the records from H are dequeued, these records are registered into CS. Since the distance of a dequeued record has already been determined, CS has the shortest path distance. It means that CS holds the materialized run-time distance view from an object to each node. We can take this advantage to reduce the processing time in our technique.

Figure 4.4(a) shows the contents of H and CS after the road network distance between a data point (p) and a node on the network have been searched by A*. In the figure, the black rectangles express the contents of H, and white circles show the records in CS. When the target node (n) moves to its neighbor node (n'), generally, we need to apply $A*$ algorithm to get the road network distance between p and n' similar to the Figure 4.1(a). In our technique, we obtain the network distance using one of the following options:

(option 1) If $r.n$ has already been in CS, the distance between q and n can be obtained by referring to d value in the record, that is $r.d$.

(option 2) Otherwise, recalculate the heuristic cost of all records in H_a for a new target node n' and then resume the search by $A*$ algorithm.

After re-utilizing the content of H and CS with changing the target nodes n_1 to n_4, the number of records are plotted in Figure 4.4(b). The number of records (shown by white circles) are reduced noticeably than the number of visited nodes by repeating the pairwise $A*$ algorithm four times.

We now discuss the method to apply the materialized run-time distance value

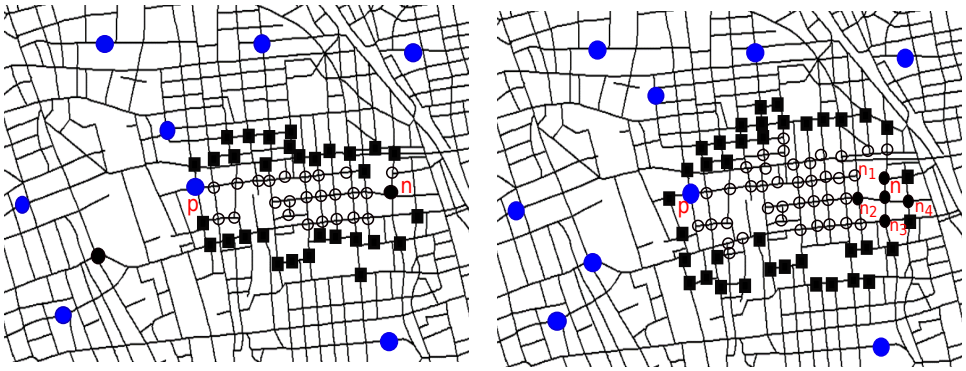


Figure 4.4: (a) The content of the H and CS once invoking $A*$ algorithm from node n to p.(b) Instance of invoking $A*$ algorithm for five time from n to n_4 and recycling the contents of CS.

to calculate the network distance. Let us recall the vicinity object described in section 4.4.1 and 4.4.2. When a vicinity object is found, it is registered to the candidate data point set (refer Algorithm 4.4) as an instance of the Vicinity Object class. The pseudo code of this class is shown in Algorithm 4.7. This class has a heap H, a closed set CS, and a data point *origin*. When a new instance is created with the candidate data point, the function **INITIALIZE** is invoked to initialize the H, CS, and *origin*.

To get the road distance for verification, **DISTANCE**(n) function calculates the network distance between o and node n. First, CS is checked (line 9 and 10) whether n is already processed or not. If n is found in CS, the function simply returns the distance of the record ($r.d$). When the **DISTANCE** function is called for the first time, H is empty. Line 11 checks H, and if it is empty, an initial record for this data point is inserted. Otherwise, H value of all records in H are modified according to the new target point n. This operation is necessary because heuristic values of records in H are calculated from the preceding target node. Line 15 to 22 in Algorithm 4.7 are the same as the corresponding lines in Algorithm 4.3.

Function **MODIFYWAVEFRONT** alters the values of all the records in H. Line 24 prepares new H_{temp} and initializes it by empty set. Lines 25 to 28 are iterated the through H till H is empty. The iteration is done by getting a

record from H, recalculating the heuristic cost and adding into H_{temp} and then finally transferring H_{temp} values to H. This function is called every time the target point is changed. However, this function is executed in the main memory, without referring to the adjacency list which is in the secondary drive. Therefore, this function can be executed faster.

Algorithm 4.7 Class Vicinity Object

```
1: Heap H
2: ClosedSet CS
3: Point origin;
4: function INITIALIZE(o)
5:     Input: Data object o;
6:     CS ← ∅
7:     H ← ∅
8:     origin ← o
9: end function
10: function DISTANCE(n)
11:     r ← CS.search(n)
12:     if r ≠ null then
13:         return r.d
14:     end if
15:     if H == ∅ then
16:         H ← H∪ < d_E(origin, n), origin, n >
17:     else
18:         ModifyWaveFront(n)
19:     end if
20:     while H is not empty do
21:         r ← H.deleteMin()
22:         if CS.contains(r.n) then continue
23:         end if
24:         if r.n == p then break
25:         end if
26:         CS.add(r)
27:         A ← expandNode(r.n)
28:         for all a ∈ A do
29:             Dist_a ← r.h + |l(r.n, a)|
30:             H ← H∪ < Dist_a + d_E(a, p), a, Dist_a >
31:         end for
32:     end while
33:     return r.d
34: end function
35: function MODIFYWAVEFRONT(n)
36:     H_temp ← ∅
37:     while H ≠ ∅ do
38:         r ← H.deleteMin()
39:         r.h ← r.d + d_E(r.n, origin)          ▷ modify the heuristic cost
40:         H_temp ← H_temp ∪ r
41:     end while
42:     H_temp
43: end function
```

4.5 Performance Study

In this section, we cover the extensive experimental studies conducted on both our proposed and basic approaches for various continuous vicinity queries. First, we describe the experimental settings. Then, we report on a comparison with our proposed approach and existing approach.

4.5.1 Experimental Environment and Settings

All the experiments were run on PC with Intel Core i7-4770 CPU ($3.4GHz$) and $16GB$ memory. All of the spatial query algorithms were developed using Java language. We evaluated our proposed approaches on two maps viz. a small map consisting of the center part of the Saitama city, and a medium map that includes the center of the city and rural area. The detail information of each road map is described in Table 3.1. In our experiments, we assumed query object was moving and objects of interest were static. The objects of interest were generated by pseudo-random sequence on the road network links with various distribution of objects of interest on road network. For instance, the density of 0.001 means an object of interest exists once 1,000 road edges. We evaluated our experiments over 500 query objects created by pseudo random sequence and averaged the data. Table 4.1 summarized the parameter values used in the all experiments.

4.5.2 Experimental Results

In all figures, "$Prop(S)$" represents our proposed method and "$Basic(S)$" denotes the basic algorithm. In addition, "S" and "M" in the parentheses indicates the small and medium maps, respectively. The basic algorithm for range query probes the objects of interest within range "r" as internal object and "$2r$" as external objects using similar way of IER strategy without considering the performance efficiency described in section 4.4.7. While generating safe-region, the basic algorithm

Table 4.1: Parameters used in performance evaluation of continuous vicinity queries

Parameter	Setting
Number of objects in each density of objects of interest (in small map)	24(0.001), 48(0.002), 124(0.005), 249(0.01),498(0.02),1245(0.05)
Number of objects in each density of objects of interest (in medium map)	109(0.001), 218(0.002), 546(0.005), 1093(0.01),2187(0.02),5468(0.05)
Range value (in km)	1, 2, 2.5, 3, 4, 5
Number of nearest neighbors (k)	1, 2, 3, 5, 7, 10
Default data object density	0.005
Default arbitrary k value	5
Default range value (in km)	3
Randomly generated query objects	500

needs to invoke $A*$ algorithm several times in **VERIFY** function to calculate road network distance. Due to this reason, the basic approach needs large processing time. Similarly, the basic algorithm for kNN based vicinity queries searches kNN by applying pairwise $A*$ algorithm repeatedly. All of the experiment results are average value evaluated with 500 queries.

Processing time comparison

Figure 4.5 shows the processing time comparison for safe-region generation of $set-kNN$ query. In Figure 4.5(a), the horizontal axis represents arbitrary "k" value-the number of queried nearest neighbors from the query object. For this experiment, the distribution of objects of interest on the road network was set to default value of 0.005. As shown in the figure, the processing time of our proposed approach is about two orders of magnitude less than that of the basic method in both small and medium map. We then compared the processing time by varying the density of objects of interest as shown in Figure 4.5(b). In this experiment, arbitrary "k" value was default value. The results demonstrate that our approach still outperforms the basic approach. In this experiment, we observed that the processing time decreased

gradually when the data point density was dense. When the distribution of objects of interest is dense, the road network distance calculation between two nodes and two objects become shorter. Consequently, the average processing time decreases when the density of objects of interest is dense.

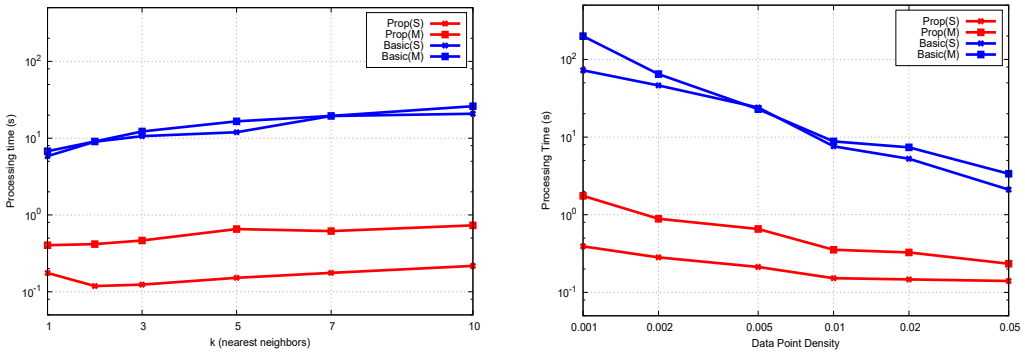


Figure 4.5: Processing time comparison to generate safe region for $set-kNN$ query (a) conducted by varying k value, (b) conducted by varying the density of objects of interest.

Next, we analyzed the processing time for $ordered-kNN$ query as shown in Figure 4.6. As expected, the size of safe-region for $ordered-kNN$ query becomes smaller than $set-kNN$ query, because the order of the distance is also considered in $ordered-kNN$ query. Accordingly, the processing time of proposed algorithm in both maps becomes faster compared to the $set-kNN$ query. For both maps, our approach is more efficient than basic algorithm in all cases.

The processing time comparison for reverse k nearest neighbor ($RkNN$) query was conducted and the outcomes are shown in Figure 4.7. In these experiments, the nearest data point (p) of the current moving object position was searched, and then the region where p was included in the kNN set was retrieved. The size of the safe-region of $RkNN$ query became the biggest among three types of the nearest neighbor queries. Consequently, the processing time also became longer than $set-kNN$ and $ordered-kNN$ queries. Nevertheless, our proposed algorithm takes less processing time compared with basic algorithm.

According to the experimental outcomes, the processing time of all kNN queries became short while the density of the objects of interest increased. The main reason is when the density of objects of interest was high, the road network distance between a query object and the kNN objects became short. Therefore, the search area in the road network also became small. The most processing time taking part to generate a safe-region was the calculation of the road network distance between the query object and each vicinity object. Therefore, the processing time decreased in accordance with the increase of the density of the objects of interest.

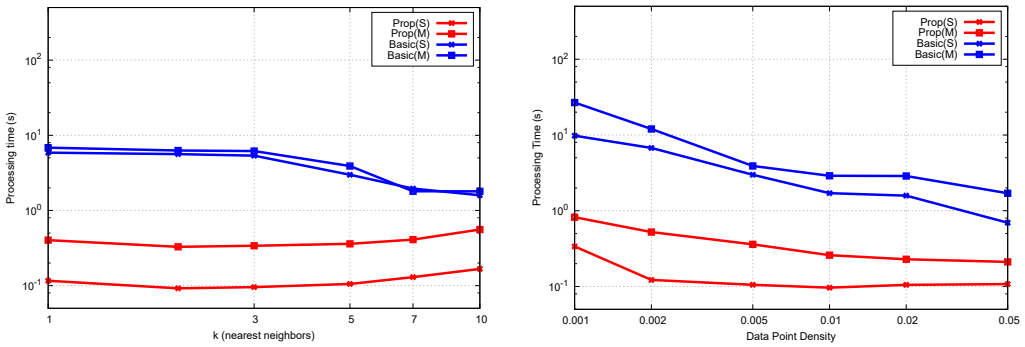


Figure 4.6: Processing time comparison for generating safe region of the $ordered - kNN$ query (a) influence of various k value, (b) influence of various density of objects of interest.

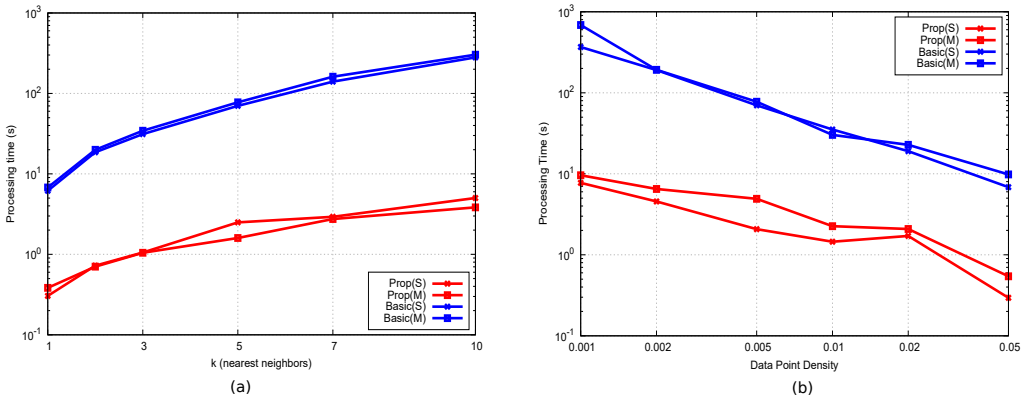


Figure 4.7: Processing time comparison for generating safe region of the $RkNN$ query (a) effectiveness of changing k value, (b) effectiveness of changing density of objects of interest.

The processing time to generate the safe-region for range query is shown in

Figure 4.8. Though, the processing time depends on the size of the range (r). For these experiments, we defined the range r is road network distance.

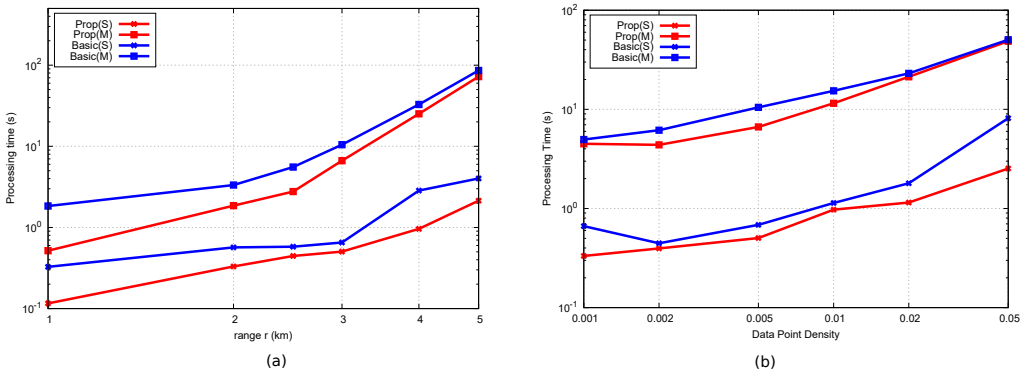


Figure 4.8: Processing time comparison for generating safe-region of the distance range query (a) effectiveness of varying the density of objects of interest (r was set $3km$), (b) effectiveness of the varying the road network distance range (r) value.

Effectiveness of the enhancements on network node access

In Figure 4.9, we evaluate the effectiveness of the performance enhancement. In this experiment, arbitrary "k" value was default value and we examined with varying the density of the objects of interest. More specifically, "B" is the basic algorithm which does not apply the enhancement for processing time described in section 4.4.7. In this section, we mentioned that the main defect of $A*$ algorithm. To address this problem, we adopted the idea of $SSMTA*$. In addition, when the algorithm invoked recently $A*$ algorithm to calculate distance while executing kNN query, it caused a drastic increase in processing time. For this reason, we applied a runtime distance materialized path view in our proposed approach to save network distance calculation time and several nodes access.

Figure 4.9 shows that our algorithm is several order of magnitude better than the basic algorithm in both road maps. The number of nodes access of basic algorithm is quite high. it is 10 to 20 times higher than the I/O cost of our proposed method. Our algorithm reduces not only the number of accessed nodes but also the processing time described in previous experimental results. Except distance range

query, when the density of objects of interest was gradually dense, the number of accessed nodes deceased because the searching area was gradually small. In the distance range query, when the density was dense, the number of invoking the distance calculation increases with the increase in the number of data objects in the specified "r".

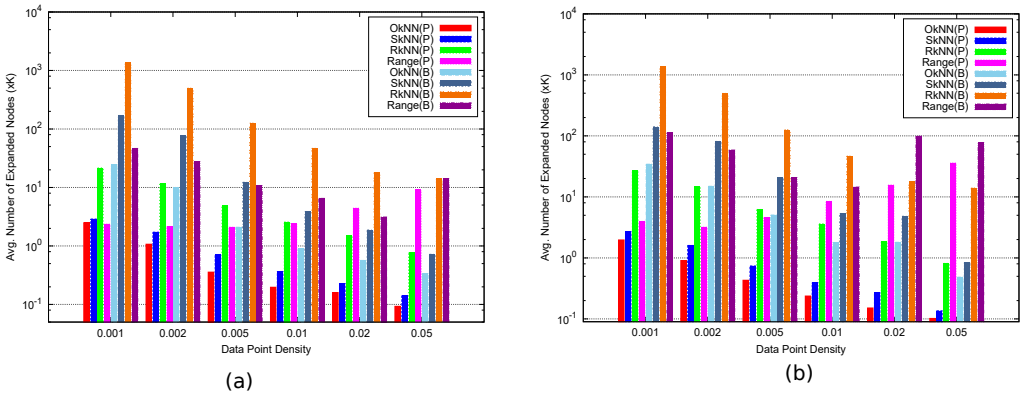


Figure 4.9: Comparison of the number of accessed nodes between proposed and basic approach of each continuous vicinity query in (a) small map, (b) medium map.

Verification of the theoretical analysis on the safe-region size

Figure 4.10 shows the average number of candidate objects to generate safe-region for three types of kNN queries. For this experiment, we use default value for k=5. The number of candidate objects increased according to the density of objects of interest. Among three kNN based vicinity query, the number of candidates of $RkNN$ is the largest. We can define the size of the safe-region according to the number of candidates in SR. In theoretical section 4.3, we mentioned that the size of the SR of $ordered - kNN$ is the smallest and the size of $RkNN$ query is the largest. The size of SR of each vicinity query is demonstrated in Figure 4.10. In addition, the size of the safe-region depends on the requested query type and the distribution of the objects of interest on the road network. When the objects of interest are sparsely distributed around the vicinity of the query point, the size of the safe-region becomes large. Contrarily, when the objects of interest are densely

distributed, the size becomes small.

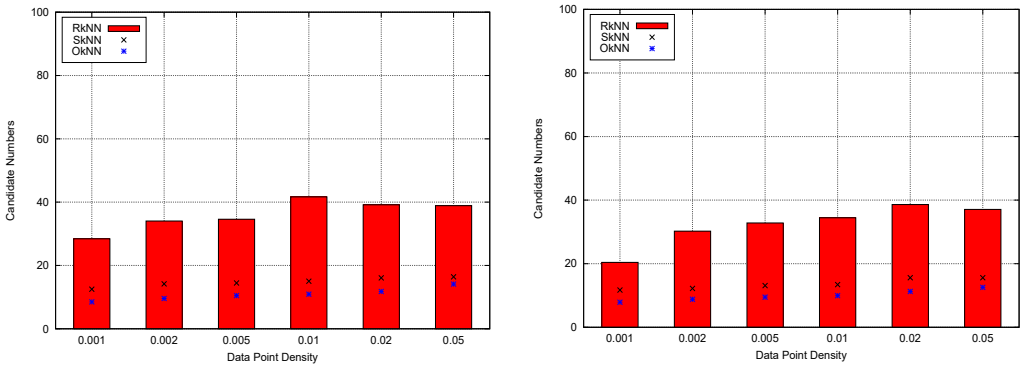


Figure 4.10: The number of candidates comparison of various vicinity query (a) evaluated on small map, (b) evaluated on medium map.

Effectiveness of the query request to server

When the query object moves fast, the moving query object needs to request a new safe-region frequently to the server. On the contrary, the query object moves slowly or stays at a position for a while, the frequency to request a new safe-region to the server becomes low.

To evaluate the frequency of query request to the server, we conducted experiments to investigate the average distance from neighboring queries to the server. For the moving path of the moving objects, we used both real paths and randomly generated paths for the moving objects in the experiments. To generate a path on the road network, we randomly created a start point "s" and destination point "d", and a moving object started to move from "s" to "d" via the shortest route. When a moving object arrived at "d", a new destination point "d'" was set randomly and the moving object continuously moved to new "d'" from the current location. By repeating this process, travel paths of moving objects were generated. Besides, we prepared 100 real paths.

Figure 4.11 shows the average trip distance between two neighboring queries requested to the server to analyze how frequently mobile query object contacts to

server by varying the density of the objects of interest. Two neighboring query means for instance, initially, a moving object moves from randomly defined "s_1" to "d_1". When it reaches "d_1", the new destination "d_2" is set and it travels from "d_1" (as "s_2") to "d_2". The queries that are invoked between two starts and destinations are called neighbor queries. In these figures, the vertical axes represent the traveling distance of a moving object within a safe-region and the horizontal axes denote the density of objects of interest. The k value is set to default value. The experimental results show that the average traveling distance decreases as density of objects of interest increases. When the size of the safe-region decrease, the frequency to request a new safe-region to the server increases. Consequently, the communication cost becomes higher in such situation. As we explained previously, the size of the safe-region of the $RkNN$ query becomes the biggest comparing to the other kNN queries.

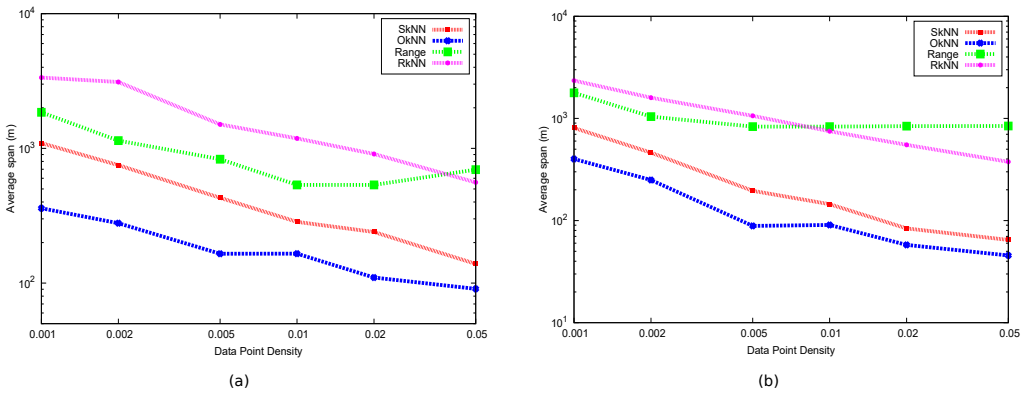


Figure 4.11: Average travel distance of moving object between two neighbor queries on (a) small map, (b) medium map.

Effectiveness of mobility rate of query object

When a moving object moves with various velocity, the effect of velocity on the performance of our approach and basic approach is examined in this section. We proceed to examine the traveling distance between neighboring queries by varying the velocity of the mobile query object in Figure 4.12. As we expected, the distance between two neighbor queries will be decreased when a object moves rapidly to new

position.

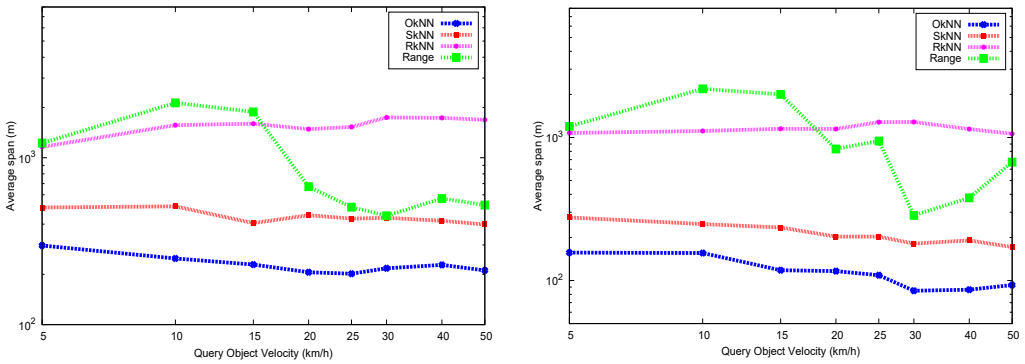


Figure 4.12: Average travel distance of moving object between two neighbor queries on (a) small map (b) medium map

While a mobile query object is moving along the randomly generated path, we analyzed the query processing time of each vicinity query and the results of both small and medium map are presented in Figure 4.13. We examined the performance where the velocity of query object varied from 5 km/h to 50 km/h. We generated the 10 travel paths of moving query object randomly with various velocity (5-50 km/h) and query processing was conducted with the distribution of object of interest set to 0.005 and arbitrary k value set to 5. We assumed the query object moves along the path with the same velocity. We measured the the processing time of each trip. When a moving object moves from one position to another rapidly, it can cover the long distance. In addition, the processing time increased with the increase in the velocity of query object because the mobile query object left their respective safe-region more often and new safe-region with updated query result required to be recomputed more frequently. The more the new query requests to server, the more the processing time consumes. However, our approach reduces the query processing time more than 10 times compared with the basic algorithm in any map.

Figure 4.14 studied how our approach and existing approach behaves on the network node expansion according to the varying velocity. Excluding the varying velocity, the number of expanded node of the existing approach is larger than our

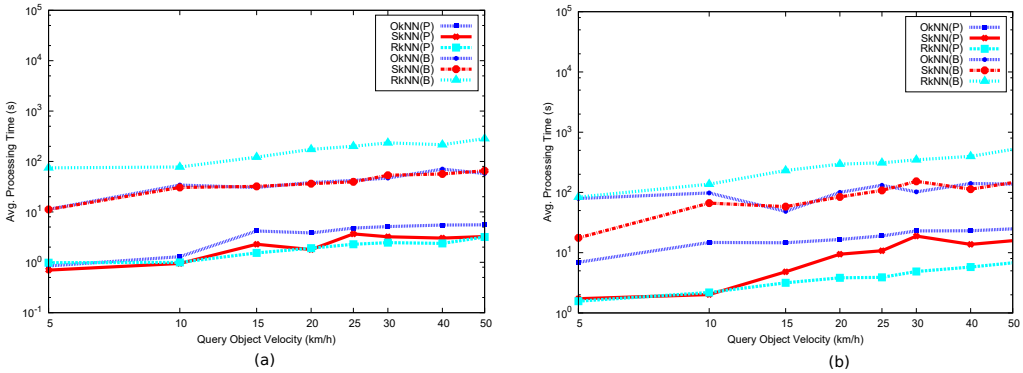


Figure 4.13: Average query processing time for each vicinity query while executing continuously in (a) small map (b) medium map

approach. Due to the performance enhancement described in section 4.4.7, our approach saved a few hundreds of node expansion by reusing the content of heap and closed set and adopting the $SSMTA*$ approach. As we noticed that, among three vicinity queries, $RkNN$ query expanded several network nodes more than other approaches.

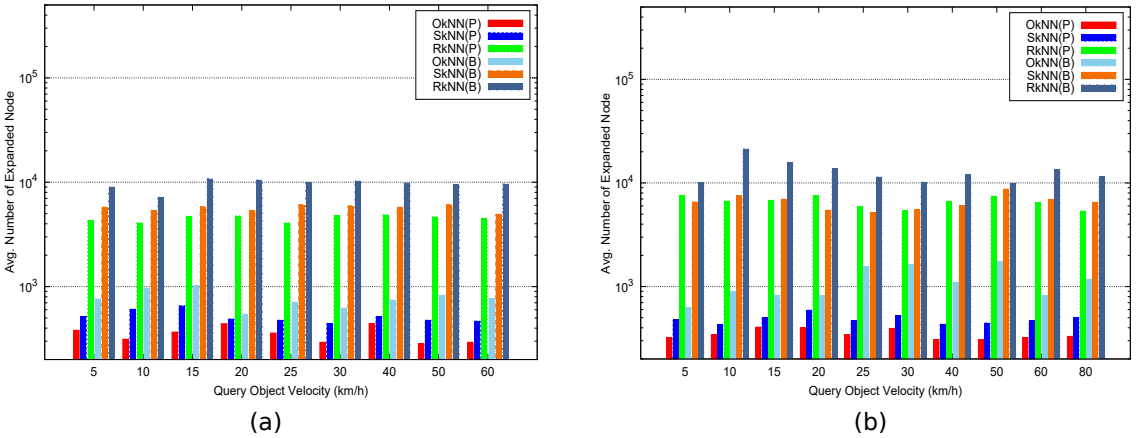


Figure 4.14: Average expanded nodes for each vicinity query while executing continuously in (a) small map (b) medium map

We next examine the communication count between server and mobile query object upon the various velocity of query object. As we described the nature of each vicinity query especially $ordered-kNN$, $set-kNN$ and reverse kNN query. The $ordered-kNN$ query considers the distance order of each query result. Therefore, the probability of the query result alteration is high compared with other vicinity

query. When the velocity of mobile query object increases, the chance of query result changes is high because the query object moves fast and thus the mobile query object needs to contact the server frequently to get new query result with new safe-region. However, the server contact number of $set - kNN$ and reverse kNN are relatively lesser than $ordered - kNN$ query because $set - kNN$ do not consider the distance order in query result and Reverse kNN query result is valid as long as a specified given object of interest lies in the safe-region. Figure 4.15 is illustrated how frequently each vicinity query requests new query result to the server. As we expected as the mobile query object moves fast, the number of server and mobile object communication increases as shown in Figure 4.15.

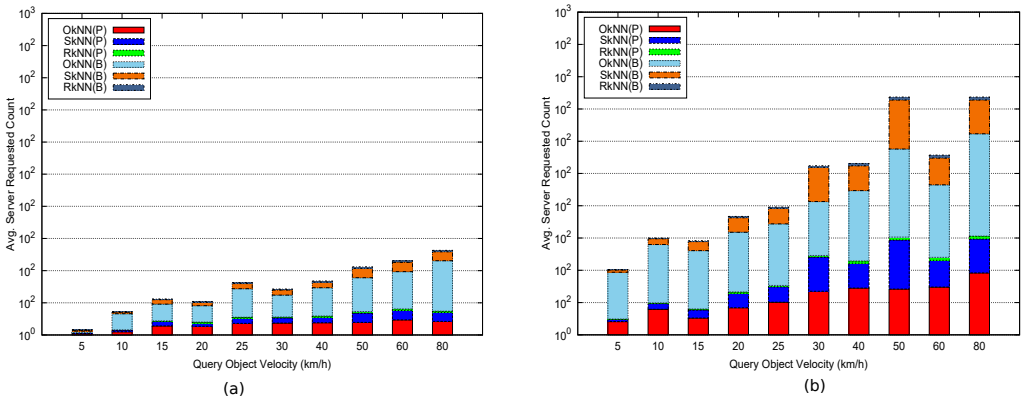


Figure 4.15: Average new query with new SR requested count to server of each vicinity query while executing continuously in (a) small map (b) medium map

4.6 Summary

In this chapter, we mainly studied the various vicinity query processing techniques for dynamic environment. The query processing in a dynamic environment needs to monitor and maintain the correctness of the query results until the query processing is terminated or the loss of interest in the query result occurs. Therefore, the frequent location updates from the moving objects incur in the server. In the simplest case, whenever an object moves, it sends its new location to the server. Obviously, this can be very wasteful, for instance if the moving object is within an

area where it does not affect any query results. Making the informed decision when to communicate update the query result becomes a key design issue to avoid such problem.

Among the several options to accomplish with the optimal decision, the safe-region is bounded by the road segments around the moving clients and needs to re-compute when the certain events take place such as a new query is invoked, or the moving object crosses its safe-region boundary. This strategy can reduce the frequent location update issues and improve the scalability of the query processing continuously. However, there is processing time deficiencies in some existing works of the safe-region generation especially in spatial network. Therefore, we proposed the fast safe-region generation for continuous vicinity queries.

The approach to generate the safe-region for vicinity queries including $ordered-kNN$, $set-kNN$, $RkNN$ and distance range queries were presented. While generating the safe-region around the mobile query object, the verification function is invoked to confirm the query condition at each network node. This function consumes large processing time. To shorten the processing time, we utilized IER and the idea of $SSMTA*$. We investigated a method that generate the run-time materialization path view to achieve fast safe-region generation. Through the experimental studies we verified that, compared to the existing approach, our method has an increased efficiency in terms of the processing time especially when data objects are distributed sparsely. We observed that when the data point were distributed biasedly on road network, the size of the safe-region became large and long processing time was necessary. In particular, the upper limit of the size of the safe-region should be considered to avoid some exceptional cases.

CHAPTER 5

Continuous Trip Route Planning Queries (CTRPQ)

A spatial query that finds an optimal trip route from a current position q to destination d, visits exactly one data point from specified data point set in a trip is called a trip route planning query. The optimal trip route can be measured by several criteria such as the total distance of a trip route, the total traveling time or the total toll fees. In this chapter, we will discuss the trip route planning queries for both static and dynamic environments using our proposed strategies to improve the average processing time.

Definition 5.1. *Trip Route Planning Query (TRPQ). Given N categories of data point sets $C_i(1 \leq i \leq N)$, a current position q and a destination d, TRPQ retrieves the minimum cost route while visiting each data point p_i selected one each from $C_i(p_i \in C_i)$ during the trip from q to d.*

The trip routes are denoted by $R_{1...N}(q, d)$. The subscript $[1, ..., N]$ shows each data point visits in predefined order from category one to category N. The trip route visits from the first category is denoted by $R_N(q)$ for the simplicity. Table 5.1 summarizes the notations which frequently appear in this chapter.

Several variation of TRP queries have been proposed in [38], [39], [59], [50]. In the first type [38], this query specifies only the visiting categories called a trip planning query (TPQ). In second sort [39], both visiting categories and orders are specified. It is called an optimal sequence route (OSR) query. The spatial condition when the start and destination locations are same point, is called a multi type nearest neighbor ($MTNN$) query [50]. The $MTNN$ query needs to select a sequence of data points that gives the shortest trip route length. The next sort of query covers the studies of [38] and [39] and combines with an additional idea to retrieves the trip routes involving several destinations based on some traveling rules, namely a multi-rule partial sequence route ($MRPSR$) query. For example, a user plans a short trip around the town including ATM, gas station and supermarket. $MRPSR$ query [59] applies traveling constraints that a user needs to visit ATM first to withdraw money before going to the next places to find the route with the relatively shortest traveling distance.

5.1 Snapshot Trip Route Planning Queries

We explained the characteristics of the general snapshot queries on spatial network in the chapter 1 and 2. In this section, we will mainly discuss the trip route planning query for static environment. We discuss some existing approach and our new approach for snapshot $TRPQ$ in this section.

5.1.1 TRPQ Using Progressive Neighbor Exploration (PNE) Approach

Suppose C_i be a category of the data points to be visited, S be a sequence of C_i to specify the visiting order $S = C_1, C_2, ..., C_n$ where n is the number of category. A query retrieving an optimal sequence routes from the start to destination visiting each category according to the given sequence S is called the optimal sequenced route (OSR) query and was proposed by Sharifzadeh et al. [39]. PNE is based

Table 5.1: Frequently used symbols in chapter 5

Notation	Meaning
N	The number of categories to be visited
C_i	The visiting data point category ($1 \leq i \leq N$)
C_{from}	The category which starts from
p_1^1	$p_1(\in C_1)$ from the category 1
p_2^1	$p_1(\in C_2)$ from the category 2
q	The current position of the moving object
d	The destination of the moving object
$d_E(x, y)$	The Euclidean distance between x and y
$d_N(x, y)$	The road network distance between x and y
$R_N(q)$	The optimal sequenced route starting at q, visiting N kinds of data, then terminated at the destination d
$L_N(q)$	The total trip route length of $R_N(q)$
$R_N^E(q)$	The optimal sequenced route searched in Euclidean distance
$L_N^E(q)$	The total trip route length of $R_N^E(q)$
$R_{2,\ldots,N}(p, d)$	The partial OSR starts from p and ends at d, visiting each data point from C_2 to C_N
$L_{2,\ldots,N}(p, d)$	The total length of $R_{2,\ldots,N}(p, d)$

on finding the distinct categories of nearest neighbors progressively to construct the optimal route from start point to destination. Originally, PNE algorithm is proposed to apply for OSR query on road network with a specified sequence category. PNE utilizes INE [26] to search the nearest neighbor objects from visiting category efficiently. Generally, PNE retrieves a nearest data point from

the first visiting category ($p_k^1 \in C_k$). Then, the search continues with the next nearest point of p_k^1 from the next category C_{k+1}. In parallel, the next nearest point $p_k^2 (\in C_k)$ of the first category C_k is also searched from p_k^1. Repeating these procedures, the search is terminated when N types of categories are found and the destination d is reached. In PNE, nearest neighbor searching starts from every visited data point of each category causing a massive node expansion. In addition, PNE expands the search area using the way similar to the Dijkstra's algorithm, consequently, PNE requires a long processing time and blemishes the performance. To improve the performance in finding the optimal route, a different approach for $TRPQ$ is presented in the next section.

5.1.2 TRPQ with A* Algorithm

To solve the PNE problem, Htoo et al. [83] proposed a new approach for OSR query based on the $A*$ algorithm called $OSRA$. The $A*$ algorithm is one of the renowned algorithm to find the shortest route given start and destination points. $OSRA$ algorithm utilizes the advantages of $A*$ algorithm to address the OSR query problem. In $OSRA$ algorithm, a search path starts at q then finds the p_1^1 from the first category C_1. In parallel, the searching from q finds another data point from C_1 in the same way as the PNE. Figure 5.1 illustrates the optimal route searching of $OSRA$ approach. In the figure, while the search path starts at q and visits the data points p_1^1 and p_1^2 , then the searching route reaches a node n_a. This route is called the partial route and denoted as $PR_{1,..,2}(q, n_a)$. A partial route (PR) starting from a point s visiting the i types of data points and reaching a node n is denoted as $PR_{1,...,i}(s, n)$. The length of the PR is denoted as $LPR_{1,...,i}(s, n)$.

The node expansion from the network node n_a makes a reference with the adjacency list to get all adjacent nodes n and the following record is composed for each adjacent node. These composed records are inserted into a heap H as below:

$$<Cost, C_i, L, n, n_a, P_{prev}>$$

where C_i is the next visiting category, L is the partial route length, $LPR_{1,\ldots,i}(s, n_a)$ is the length of PR, and $Cost$ is $L + d_E(n, d)$. P_{prev} is the last visited data point that belongs to C_{i-1}. The heap is ordered by $Cost$ value. The value of n_a from the previous expanded node keeps restoring the trip route path by backtracking form n to q.

Generally, the shortest path search algorithms such as Dijkstra's and $A*$ algorithm register the de-heaped records in a closed set (CS). To avoid the duplicate node expansion, before storing the de-heaped record into CS, de-heaped record from H is checked whether it has already been included in CS. In practice, CS is implemented by a hash table or balanced binary tree. Each record in CS is assigned a key with a combination of the current node (n) and the previous visited data point P_{prev} because a node on the road network needs to be accessed multiple times when the previous visited data point is different. For example, the search path targeting to C_1 is starting from q finds the data point p_1^1 and p_2^1 belonging to C_1. After that, the next search targeting category C_2 starts from both of them.

The searching from two data points are performed independently. Therefore, a node that has been expanded by another search get expanded again which causes a rapid increase in processing time. By repeating the **DeleteMin** which de-heaps the record with minimum value, operation on H and the node expansion, the search area is gradually enlarged. The search is terminated when the extracted record from H reaches the destination d. Although $OSRA$ algorithm avoids circular expansion for all directions from start point by adopting the $A*$ approach, due to the nature of OSR query, the same node gets expanded more than once. In addition, $OSRA$ algorithm runs fast when the distribution of all data points from all categories is similar. In contrast, when the data point distribution of each category varies, there is a great effect on the processing time which increases the processing time.

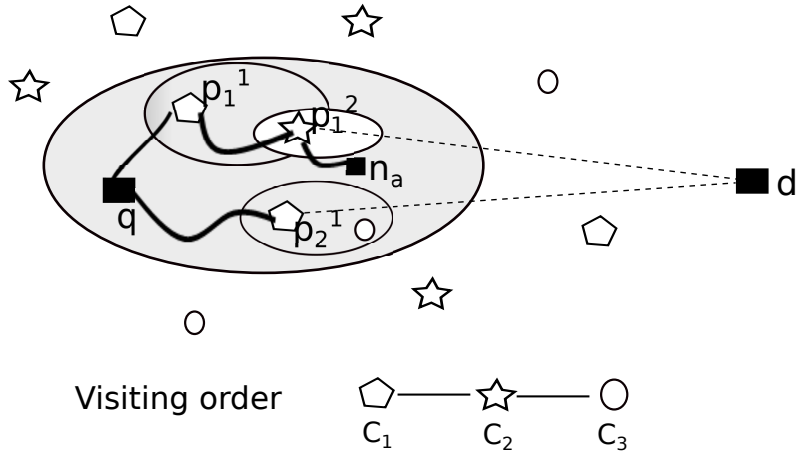


Figure 5.1: An example of optimal sequence route query using $A*$ algorithm.

5.1.3 Sparse Category First Algorithm for TRPQ

Htoo et al. [83] indicated the problem of $OSRA$ algorithm described in previous section that when the data point distribution differs largely, the sparsest data point category must be determined first to shorten the processing time. In [83], Htoo et al. assumed the densities of the data points sets were known in advance. However, this assumption does not satisfy for every real scenario. Therefore, the investigation methodology of the density of the data point set distribution on road network is presented in this section. The next section discusses how to solve the snapshot $TRPQ$ problem using the sparse category first (SCF) algorithm.

We summarize the processing procedures of the SCF algorithm for trip route query in the following steps:

Step 1 First, the algorithm searches the shortest path connecting start position q and destination d utilizing the $A*$ algorithm as shown in Figure 5.2(a). After the algorithm defines the shortest path without any visited data points, the contents of the heap (H) and the closed set (CS) are kept for the next steps.

Step 2 In next step, the algorithm identifies the sparse data point set among the categories N. The algorithm probes the data points in each category by

expanding the node n between q and d bounded by an ellipse region whose focal points are q and d. Every time a data point is found in this procedure, the category of the found data point is marked as a visited data point. The above searching continues until at least one data point from all categories are noticed. Then, it determines the last marked category C_l. C_l considers the most sparsely distributed category around the shortest path connecting between q and d. In Figure 5.2(b), the sparse category C_l is denoted with stars, and the first found data point is v_1.

Step 3 The problem to find the trip route can be solved by dividing into two sub-problems; one is finding the partial route from q to v_1 through $l-1$ data points selected each in $C_i(1 \leq i \leq l-1)$, and the other one is finding the route from v_1 to d through $N-l$ data points selected each in $C_j(l+1 \leq j \leq N)$. Then, it merges the results of two partial trip routes. Finally, the route $q \rightarrow x \rightarrow v \rightarrow y \rightarrow d$ is found as illustrated in Figure 5.2(c). Suppose, a trip route be TR and the total length of the route TR be L. In this step, the trip route TR does not guarantee to be an optimal shortest route. So, the algorithm examines this in the next step.

Step 4 The node expansion as described in step 2 continues until it satisfies the condition: $d_N(q,n) + d_E(n,d) \leq L$. The found data points from sparse category C_l are added into a candidate data point set C_{and} as the l^{th} visiting data point.

Step 5 When we find the new candidate from the C_l, it searches the trip route via the l^{th} visiting data point using the same way of step 3. Then, the shortest trip route is returned as a result. In Figure 5.2(d), the data point v_2 from C_l is found and inserted into the C_{and}. The partial trip routes from q to v_2 and v_2 to d are retrieved and the total trip route length is compared with TR to find the shortest trip route.

For our optimal trip route, we present the following lemma that validates the

optimal shortest trip route.

Lemma 5.1. *The shortest trip route visits one of the data points in C_{and} as the l^{th} visiting data point.*

Proof. Suppose, the shortest trip route visits a data point p ($\notin C_{and}$) as the l^{th} data point. The total length of the partial route from q to p and the next partial route from p to d must be shorter than the first found trip route in Step 3, therefore the following equation stands. □

$$L_{1,\ldots,l-1}(q,p) + L_{l+1,\ldots,N}(p,d) < L \tag{5.1}$$

When $L_{1,\ldots,l-1}(q,p) \geq d_N(q,p)$ and $L_{l+1,\ldots,N}(p,d) \geq d_N(q,p)$ stands, the following equation is satisfied.

$$d_N(q,p) + d_N(p,d) > d_N(q,p) + d_E(p,d) \tag{5.2}$$

However, we assumed that p is not included in C_{and}, therefore, $d_N(q,p) + d_E(p,d) > L$ and then

$$d_N(q,p) + d_N(p,d) > L \tag{5.3}$$

The proof of this lemma is by contradiction.

5.1.4 Euclidean Based TRPQ

TRP query adopting the Euclidean restriction strategy is considerably fast in query processing. This approach is proposed by Ohsawa et al. [82]; in this approach the candidates for OSR in the Euclidean space is gathered, and then verified those candidates in the road network distance using a R-Tree spatial index. Although the length of the trip route in Euclidean distance gives the lower bound of it in the road

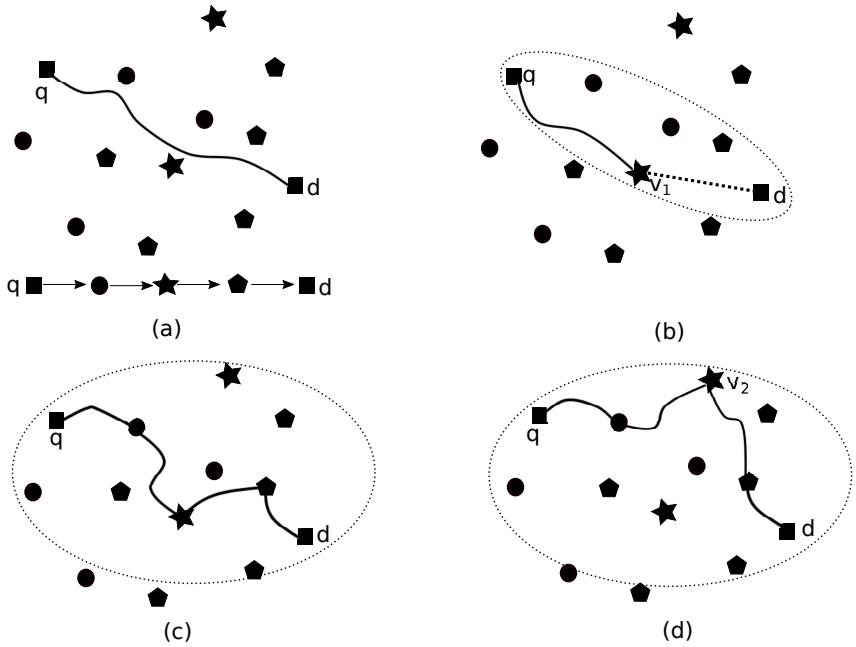


Figure 5.2: The processing flow of Sparse Category First algorithm when $TRPQ$ visits four data categories: (a) search the shortest route between q and d, (b) retrieves the object from sparse category among four data categories (c) retrieves the optimal route via object from sparse category, and (d) next possible optimal route via object from sparse category.

network distance, the shortest OSR in Euclidean distance is not always the shortest in road network distance. Let the shortest route in Euclidean space be Sr and the verified route in road network is $LN(Sr)$. The shortest routes less than $LN(Sr)$ have the potential to be the shortest route in the spatial network. Therefore, all of the OSR routes in Euclidean space less than $LN(Sr)$ must be retrieved and any OSR candidates in Euclidean space greater than $LN(Sr)$ can be safely pruned. All OSRs whose length are less than $LN(Sr)$ can be determined using an incremental search strategy. The searching process is controlled by a heap, and the optimal route is found when the heap becomes empty. This algorithm can find the trip route in two or three orders of magnitude faster in road network distance. Therefore, the trip route searched in Euclidean space can be used for pruning the search space in the road network.

5.2 Continuous Trip Route Planning Query

We discussed about the $TRPQ$ in a static environment in the preceding section. In this section, we discuss about the continuous TRP query with an effective monitoring technique to provide up to date query result which is a crucial function for any continuous query invoked by a moving objects.

In contrast to the regular queries on static environment that are evaluated only once, a continuous query has to be monitored and evaluated to preserve the freshness and update the query result continuously according to the current location of moving object (MO). The nature of continuous queries can be designated as client-server model. In this model, the clients (moving objects) issue queries to the server for the computation of the queries. Imagine a simple $TRPQ$ example, a user driving a car (MO) in an unfamiliar city wants to know the nearest gas station from the current location. The client (car) sends the query to the LBS server and obtains the query result with the nearest gas station. However, when the MO ignores the result and keeps driving to the destination, the query result becomes invalid, and MO needs to request again the same query to the server. To avoid this problem, the result can automatically update with the distance or time interval, however, when the distance or time interval between the repeated query is short, the MO will get the same result as the previous query. On the other hand, MO may overlook the optimal result. One of the big challenges for the continuous queries is to update the valid data continuously.

The traditional approach of using the index on the moving object locations suffers from the constant update of the index and re-evaluation of all query whenever the objects moves. The approaches for the continuous queries in the road network continuously monitors the distance between a query and a data object or queries are repeated periodically have been proposed. However, these approaches are not efficient because when the frequency of request and update become high, the workload on the server becomes high. To overcome the server workload problem, the

safe-region method for several types of location-dependent queries has been proposed in [23], [65], [73], [80], and [81]. When a MO issues a spatial query to the server, the server generates a safe-region which is an area such that the reported result is valid as long as the MO remains within the safe-region. By the time the MO leaves the safe-region, a new query result and safe-region are needed to request to the server. However, these safe-region generation method cannot be directly applied for continuous trip route query because their methodologies cannot achieve the $CTRPQ$ with multiple stopovers efficiently. To the best of our knowledge, our strategies to address $CTRPQ$ problem in road network with safe-region is the first attempt.

As discussed, TRP query consumes a lot of processing time even while it is invoked with the static objects compared with other spatial queries. Nevertheless, we studied some $TRPQ$ algorithms in section 5.1.2 and 5.1.3. These algorithms can perform proficiently to retrieve the optimal TR. These algorithms are the foundation of our work for the continuous $TRPQ$ in searching the optimal TR. In addition, we need a novel approach to process TRP query with the moving objects. It was observed that the safe-region approach accomplishes efficiently in several continuous spatial queries because it reduces the overall computation time as the query needs to re-evaluate only when MO leaves the safe-region. Figure 5.3 shows an example of simple $CTRPQ$ with generated safe-region. In this example, before reaching to the destination d, three data points c_1, c_2 and c_3 selected from three distinct categories, are specified to visit. The color line shows the optimal trip route and the marked region is defined with the plus signs as a safe-region. Although the MO veers from the optimal route, the route is still optimal until the current position is in the safe-region. Therefore, the MO only needs to target to the first visiting point (c_1) on the route if the user is still in the safe region. Safe-region based solution for $CTRPQ$ reduces the number of queries even MO veers the query route while MO is in the safe-region. In the following sections, we present an efficient and effective techniques to monitor MO to process TRP queries

continuously using the concept of safe-region.

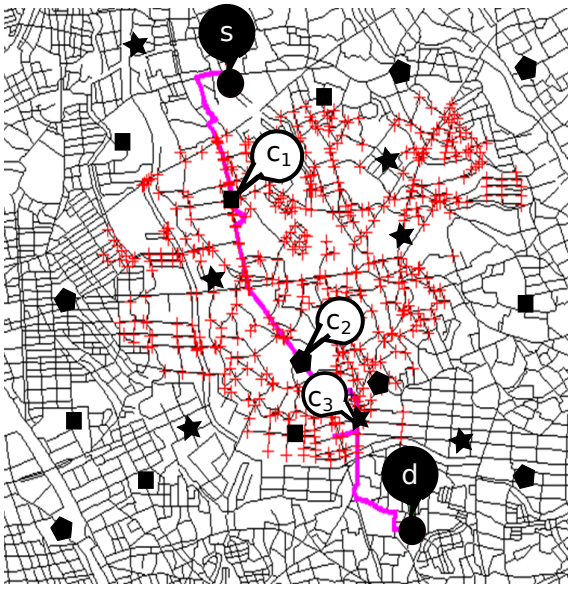


Figure 5.3: An example of safe-region which is marked with plus symbols (the color thick line is the optimal trip route between “ s ” and “ d ” , bold circles are denoted as start and destination, black rectangles are data objects of the visiting category 1, the black polygons are data objects of the visiting category 2 and stars are data objects of the visiting category 3)

5.3 The Strategies to Generate Safe-Region for Continuous TRPQ

On the continuous trip route planning queries, the current position of a moving object (MO) changes continually. When a MO at q issues a query to a server, the server searches the optimal TR with generated safe-region (SR) and sends the result to the MO as illustrated in Figure 5.4. In this example, the optimal TR is visiting the two data points p_1^1 and p_1^2 from category C_1 and C_2 ($N = 2$) respectively with the uniquely specified visiting order. The MO always checks whether it is inside the SR. When MO leaves SR, it requests a new optimal TR with the corresponding SR. On the other hand, when the MO follows the route and reaches p_1 which belongs to the first visiting category on the optimal route, the MO issues a new

query with relating safe-region from p_1^1 and the visiting category number is reduced to $N-1(C_2)$. This procedure is repeated until the MO reaches the last visiting category(C_N). After the MO passes through the data point from the final category, the SR generation is not necessary and the problem is converted into the shortest path search between the current position and the destination. For the sake of simplicity, we define the first visiting data point as p_1^1 from category C_1.

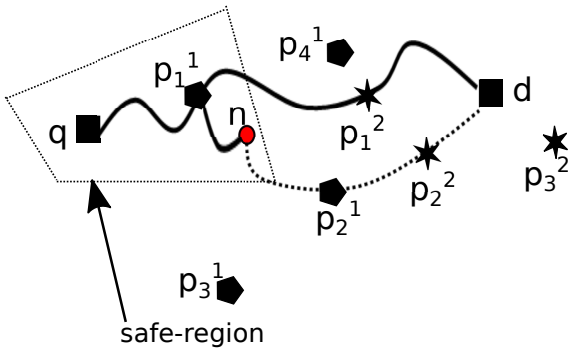


Figure 5.4: Safe region and rival objects in $CTRPQ$.

A formal definition and properties regarding the trip route and safe-region are given below:

Definition 5.2. *Safe-region (SR). A SR is a collection of road segments on which TRP queries issued at any point in the region gives the same result. In other words, in the safe-region, $R_N(q) = R_N(q')$, where q is the initial position of MO and q' is any position in SR.*

While the MO abide in the SR, no new query is necessary even MO veers the first queried trip route. The SR of TR satisfies the following properties.

Property 5.1. *Let the first visiting data point on the TR searched from a position q be $p_1(\in C_1)$. The first visiting data point searched from any other position (q') in the SR is identical with p_1.*

Proof. Property 5.1 is proved by contradiction. If the first visiting point of the query from q' is p_1' ($\neq p_1$), the TR queried from q' becomes $R_N(q')$ ($\neq R_N(q)$). This result contradicts the definition of SR. Therefore, this property holds. □

Property 5.2. *When a TR is given, the rest of the route after visiting the data point in the first category (C_1) is uniquely determined except the case for plural TR giving the same length.*

Proof. The queried TR is the optimal (the shortest route). Therefore, if the first visiting data point (p_1) is given, the rest of the TR is uniquely determined. □

According to the property 5.2, it is enough to search the area to find the safe-region on the road network where the first visiting data point for the TRs is same. By property 5.1, the first visiting data points are different on the TRs queried in the SR. On the contrary, the first visiting data points that are different on the TRs are queried by two end points (network nodes) of a network link across the SR border. In this case, the shortest TR queried from a node included in the SR goes through p_1 ($\in C_1$), and the shortest TR queried from the other node of the link goes through the other data point p' ($\in C_1$). Hereafter, p' is called a rival object (RO). For example, in Figure 5.4, p_3^1, p_4^1 and p_3^2 are rival objects against p_1^1.

Definition 5.3. *Minimal Rival Objects Set. Minimal rival objects set is the set of necessary and sufficient rival objects to form an SR.*

5.3.1 Typical Approaches for Safe-Region Generation

This section presents the typical approach to achieve the safe-region generation for the continuous TRP query.

Based on the property 5.2, the trip route queried by q always pass through a data point p_1^1 from C_1 as the first visiting category in the safe-region (refer Figure 5.4). The SR to be generated is a region where the first visiting point on TR is $p_1(p_1^1)$. The SR can be generated by expanding the region from p_1. While generating the SR, each expanded node must check and verify whether the first visiting point on the queried TR is p_1 or not. The node expansion is done using the same way of the Dijkstra's shortest path algorithm. Each records for the node expansion can

be controlled by heap (H) with the following format of record $<cost, n, l>$. Here, n is a current noticed node, cost is a road network distance between the expanded data point (p_1) and n: $d_N(p_1, n)$, and l is a road segment where one edge is n and the other is adjacent visited node. The records in H is stored in ascending order of the cost value. During the node expansion, each dequeued record from H is added to the closed set (CS) to avoid the duplicate node expansion.

Suppose, the de-heaped record from H is r, the TR starting from $r.n$ ($R_N(r.n)$) is searched by employing the algorithm described in section 5.1.2 or 5.1.3. First, if the searched TR is with the first visiting data point p_1, the link $r.l$ is added into the SR. Second, the adjacent links of $r.n$ are obtained by referring to the adjacency list, and the following procedure will be continued on each link. Let an opposite end-point of $r.n$ be n_k and link be l_k. A new record $<n.cost + |l_k|, n_k, l_k>$ is composed and inserted into H and CS. The above procedures are called the node expansion. On the other hand, in the first stage, when the queried TR is not with the first visiting data point p_1, the further procedures do not continue because the node is not included in the SR. Remarkably, even in this case, a part of the link $r.l$ can be included in the SR. Therefore, if the query condition for a part of the link is satisfied, the part will be added into the SR.

Generally, the SR cannot be described as a closed region in the similar way as the region formed by Voronoi decomposition. For example, when data points in C_1 are distributed only around the center of the road network, the TR will contain the same data point as the first visiting data point even when the query point is located far away. In such a case, the SR becomes large, and the processing time suffers drastically because the processing time is directly proportional to the number of nodes contained in the SR. During the node expansion, we assume that the moving object does not veer far away from the queried TR. Therefore, we set an upper limit of the node number contained in SR. Thus, the node expansion for the SR will terminate when the number of node is exceeded the upper limit, and then send the generated SR to the moving object. The typical SR generation approach

does not take into account the number of rival objects and the Definition 5.3 is applicable. Consequently, the size of safe-region becomes large and the processing time increases to generate the safe-region. Therefore, the definition 5.3 is required to reduce the size of the safe-region and shorten the SR generation time effectively.

5.3.2 The Architecture of Our Proposed Strategies

The main concern about the typical safe-region generation approach is the large size of SR and the deficiency on defining the number of the rival objects. In this section, we present a theoretical analysis to evaluate the effectiveness of the safe-region and propose two efficient approaches to generate the SR called the preceding rival addition (PRA) and tardy rival addition (TRA).

Predictable number of rival objects (RO)

As we mentioned how the number of rival objects effects the size of safe-region and the performance efficiency. If the minimal rival object set ($MROS$) is given in advance, the safe-region can be obtained swiftly. In such cases, when the TR starts from each object ro ($\in RO$), $R_{2,\ldots,N}(ro, d)$ and $L_{2,\ldots,N}(ro, d)$ can be searched in advance.

Suppose, the first visiting data object of the queried TR is p and the SR is created around the data object (refer Figure 5.5). In figure, d_p shows the length of the trip between p and the destination d as ($L_{2,\ldots,N}(p, d)$). Five data points from ro_1 to ro_5 are the rival objects of p. As described above, the length of TR from each rival object (ro_i) to d (in figure d_i) can be calculated in advance. Consequently, the SR can be created fast. We verify the TR length from n to the first visiting data point with TR of other rival objects using the precomputed d_i. We assume that n is included in the SR while the equation 5.4 stands. By applying the following inequality, we generate SR.

$$d_N(p,n) + d_p \leq \min_i \{d_N(ro_i,n) + d_i\} \tag{5.4}$$

In next section, we highlight two approaches to accumulate the minimum rival objects.

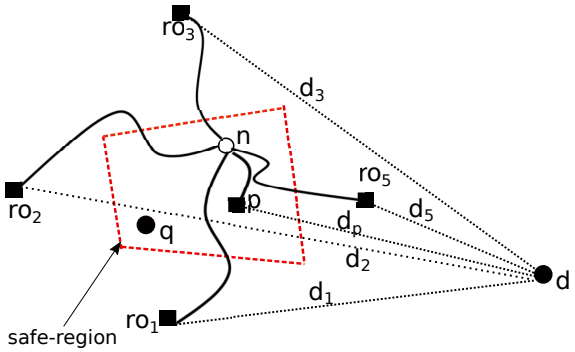


Figure 5.5: Minimal Rival Objects Set

Preceding Rival Addition (PRA)

Finding enough RO is necessary because it effects the shape and size of the safe-region. When a TR is searched in Euclidean distance, we can obtain the candidate of RO easily. The length of the TR in Euclidean distance gives the lower bound of the TR in road network distance, $L_N(q) \geq L_N^E(q)$. Inside the SR, the length of the TR starting from $p_1 (\in C_1)$ and a length of another TR starting from p' $(\in C_1)$ satisfies the following property:

Property 5.3. *Let p_1 be the first visiting point in a TR. When a network node n is included in the safe-region of TR, a data point p' $(\in C_1)$ can be an RO if the following inequality is satisfied.*

$$L_{2,\ldots,N}(p_1) + 2 \times d_N(p_1,n) \geq L_N^E(p_1') \tag{5.5}$$

where the sub-script 2,...,N represents a route starting from the 2^{nd} category to the N^{th} category.

Proof. If the length of a TR passing through p' is shorter than the TR passing through p_1, the following inequality is satisfied. □

$$L_{2,\ldots,N}(p_1) + d_N(p_1, n) \geq L_{2,\ldots,N}(p'_1) + d_N(p'_1, n) \tag{5.6}$$

The triangle inequality implies that

$$d_N(p'_1, n) \geq d_E(p'_1, p_1) - d_N(p_1, n) \tag{5.7}$$

Without loss of generality, from equation 5.6 and 5.7,

$$L_{2,\ldots,N}(p_1) + d_N(p_1, n) \geq L_{2,\ldots,N}(p'_1) + d_E(p'_1, p_1) - d_N(p_1, n) \tag{5.8}$$

$$L_{2,\ldots,N}(p_1) + d_N(p_1, n) \geq L^E_N(p'_1) - d_N(p_1, n) \tag{5.9}$$

$$L_{2,\ldots,N}(p_1) + 2 \times d_N(p_1, n) \geq L^E_N(p'_1) \tag{5.10}$$

Therefore, the both side of the above inequality clearly follows.

The typical SR generation approach described in section 5.3.1. enlarges the search area gradually from the first visiting data point and checks the queried TR from each expanded node to verify the first visiting data point. These procedures will proceed on every expanded node which deviate the performance efficiencies. Therefore, we contrive a method to shorten the processing time to form an SR by reducing the number of rival objects. While enlarging the SR, we retrieve all possible RO that satisfy the property 5.3. We calculate the length of the TR (i.e $L_{2,\ldots,N}(p'_1, d)$) of each RO in advance. By employing the shortest path search algorithm, the shortest TR starting from the expanded node n can be obtained by searching between n and each rival object.

(i) Algorithm

We use R-Tree to index each data point category (C_i). R-Tree contains index

to its entries and minimum bounding rectangle that contain all its data objects. The detail description is given in Chapter 2. Algorithm 5.1 outlines the pseudo code of the PRA algorithm for generating the SR. The parameter values in PRA algorithm are q: the current position of the moving object, d: the destination of the trip and N: the number of visiting categories during a trip. Initially, the lines 3 to 5 initialize a heap (H), the closed set (CS), the result set of the segments to be included in SR and the set of the rival objects.

Next, the function called **INITIALIZE** is invoked. The main procedure of this function can be summarized as follow:

1. Find the optimal TR starting from q to d visiting N categories using the method of $TRPQ$ explained in section 5.1.2.

2. Initialize the H with two records of the road link l on which the first visiting data point of this TR (p_1) exist. For instance, node a and b are the edges of link l. l_a and l_b are the parts of l is divided at p_1. The composed two records are described as follow:

$$<L_{2,\ldots,N}(a) + |l_a|, a, l_a>, <L_{2,\ldots,N}(b) + |l_b|, b, l_b>$$

3. Remove the found p_1 from R-Tree.

The pseudo code of the **INITIALIZE** function is presented in Algorithm 5.2. The entries from the H are de-heaped iteratively until the H becomes empty (line 6 to 22). A record with minimum d value is de-heaped from H described in line 8 and this record is registered into CS to avoid the duplicate accessing. In line 10, **ADDCANDIDATERO** function is invoked from the node $r.n$ de-heaped from H to search the enough RO. To probe the RO, the Euclidean based TR (ETR) will be retrieved from $r.n$ until equation (5.6) is satisfied. The first visiting data point from ETR will be added into the rival object set (ROS). In a special case, the first visiting data point from the **INITIALIZE** function and the rival objects from the

Algorithm 5.1 *PRA*

```
1: function PRA(q, d, N, C_from)
2:     let C_from be C_1
3:     H ← ∅, CS ← ∅
4:     SR ← ∅, RO ← ∅
5:     p_1 ← INITIALIZE(q, d, N, H)                    ▷ Algorithm 5.2
6:     while H not empty do
7:         r ← H.deleteMin()
8:         CS ← CS ∪ r
9:         ADDCANDIDTERO(r, RO, p_1)                   ▷ Algorithm 5.3
10:        minDist ← MINDISTINSET(r, RO)               ▷ Algorithm 5.4
11:        if minDist<r.d then
12:            SR ← SR ∪ CLIP(r.l, minDist)
13:        else
14:            SR ← SR ∪ r.l
15:        end if
16:        for all e ∈ getAdjacentLinks(r.n) do
17:            if e.l is not visited then
18:                H.enHeap(< r.d + |e.l|, e.next, e.l >)
19:            end if
20:        end for
21:    end while
22:    return SR
23: end function
```

first category are removed from C_1 which means they are removed from R-Tree. The pseudo code description of this function is presented in Algorithm 5.3.

Next, the algorithm examines the distance from the current node $r.n$ to each RO and determines the minimum distance among them in line 10 of Algorithm 5.1. If the distance is smaller than $r.d$, it means a route visiting the RO is shorter than the route visiting p_1, in other words, $r.n$ is not included in the SR. In this case, $r.l$ is divided into two segments, and the part TR passing through p_1 which is shorter than the rival object is added into SR (line 12). On the other hand, if the route visiting p_1 is shorter, the whole $r.l$ is added into SR (line 14). All the adjacent links of node $r.n$ are obtained by referring to the adjacency list. The records of the adjacent links are composed as shown in line 18 and inserted into H. The algorithm

Algorithm 5.2 Initialize

1: **function** INITIALIZE(q, d, N, H, C_{from})
2: $TR \leftarrow R_N(q, d, C_{from})$
3: $p_1 \leftarrow TR.p[C_{from}]$
4: $p_l \leftarrow getNearestLinks(p_1)$
5: $H.enHeap(< L_{2,...,N}(p_l[0]) + |p_l[0]|, p_l[0].n, p_l[0] >)$
6: $H.enHeap(< L_{2,...,N}(p_l[1]) + |p_l[1]|, p_l[1].n, p_l[1] >)$
7: **end function**

Algorithm 5.3 AddCandidateRO for PRA

1: **function** ADDCANDIDATERO(r, RO, p_1)
2: $TR \leftarrow R_N^E(p_1, d)$
3: $next \leftarrow TR.p[p_1]$
4: **while** $2 * r.d > TR.length$ **do**
5: **if** $next.p \notin RO$ **then**
6: $next.d \leftarrow R_{2,..,N}(next.p, d)$
7: $RO \leftarrow RO \cup next$
8: **end if**
9: $RTree[p_1].delete(next.p)$
10: $TR \leftarrow R_N^E(p_1, d)$
11: $next \leftarrow TR.p[p_1]$
12: **end while**
13: **end function**

stops when the heap becomes empty.

(ii) Obtaining the Minimum Rival Objects set

Algorithm 5.3 illustrates the **ADDCANDIDATERO** which obtains RO set incrementally. Generally, the new objects which satisfy the Equation 5.5, are obtained and added into RO. As we mentioned, the optimal TR on Euclidean distance from p_1 and visiting N categories is searched incrementally to assemble the RO in line 2, and the first visiting data point ($\in C_1$) is assigned to the next variable. Line 4 to 12 iterate to search new rival objects that satisfy the property 5.3 and it is added into RO. Note that PRA approach removes the rival objects from the R-Tree as described in line 9.

(iii) Analysis of the Minimum Road Network Distance

The PRA algorithm analyzes the minimum network distance among the rival objects. Algorithm 5.4 outlines the procedure of analyzing the minimum distance between an expanded node $r.n$ and each rival objects while the safe-region is gradually enlarged. For calculating the network distance, the existing pairwise $A*$ algorithm can be applied. Though, $A*$ algorithm is efficient enough to search the shortest path (only distance) in the road network, the distance queries have to be repeated several times. Therefore, we employ an efficient method to calculate the distance with less processing time called on-the-fly network distance materialization method.

We first start by describing the conventional pairwise $A*$ algorithm before describing our method. Suppose, two point a and b on the road network are given, the network distance $d_N(a, b)$ is calculated by $A*$ algorithm as explained in Chapter 4. In our pairwise $A*$ algorithm, a heap (H) is used to retrieve the record with minimum cost and the record format in H is $<cost, n, dist>$ where cost is the value of $d_N(s, n) + d_E(n, d)$, n is the current noticed node, and $dist$ is the road network distance between s and n ($d_N(s, n)$). Not only to prevent the duplicate searching but also to reuse the record information, all de-heaped records are stored in closed set (CS).

Initially, the heap is initialized with the record $<d_E(s, d), s, 0>$, then the following steps are repeated.

Step 1 De-heap a record (r) with the minimum cost value from H.

Step 2 Inset $r.n$ into CS only when it is not in CS.

Step 3 Obtain the adjacent nodes of $r.n$ referring to the adjacency list until $r.n$ reaches d. For each adjacent node n', compose the record as $<d_N(r.n, n') + r.d + d_E(n', e), n', d_N(r.n, n') + r.d>$ and en-heap into H.

Step 4 When $r.n$ reaches d, the path to d has been found, then return the $r.d$ value as the shortest path length.

Now, to examine the road network distance between each RO and current node, we adopt the pairwise $A*$ algorithm and lemma 5.2 proposed in [84] and shown as an example in Figure 5.6. In the figure, the space rectangle represents the node in H and cross shows the nodes in CS. The Algorithm 5.4 details the road network distance calculation between $r.n$ and each rival object from RO.

Lemma 5.2. *Let s be a start point, $r.n$ be a current node and $r.d$ is distance from s to $r.n$. If a record r is included in a closed set CS, then $r.d$ is the shortest distance between s and $r.n$.*

Proof. Suppose, n_p be an adjacent node of p on the shortest path. The distance between the s and p ($d_N(s,p)$) can be calculated by the equation $d_N(s,p) = d_N(s,n_p) + d_N(n_p,p)$. Here, $d_N(s,p)$ is the shortest distance on the road network. Then $d_N(s,n_p)$ is also the shortest distance between s and n_p. □

In Figure 5.6(a), the position of RO (p in figure) is fixed, on the other hand, $r.n$ moves around the surrounding area of q. Nodes n_1 to n_4 are the adjacent nodes of q. Similarly, $d_N(ro,n_1)$ to $d_N(ro,n_4)$ are requested. In this situation, similar operations are repeated when we apply $A*$. This repetition can be avoided by reusing the contents of H and CS. According to the lemma 5.2, the records in CS have $d_N(ro,n)$ as distance value in the record. Therefore, if a destination node is already in CS, $d_N(ro,n)$ can be obtained to refer $r.d$. In addition, the following procedures are needed.

Step 1 Recalculate cost value of all the records in H by

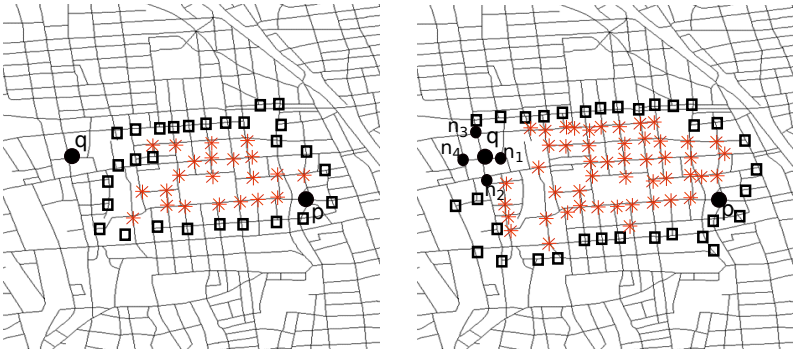
$$cost = d_N(ro, r.n) + d_E(r.n, n)$$

Step 2 Resume the search procedure using recalculated H. Step(1) is executed only for the nodes in the heap and no disk I/O is necessary, therefore, the CPU time of this step is short.

Figure 5.6(b) shows the status after the distance to n_1 to n_4 have been obtained. As shown in this figure, the number of nodes in CS and H are considerably smaller than five times of them in Figure 5.6(a). The difference of node expansion times is directly proportional to the processing time.

Algorithm 5.4 minDistInSet

```
1: function MINDISTINSET(r, RO)
2:     minDist ← maxValue
3:     for all ro ∈ RO do
4:         dist ← AStar.ShortestPath(r, ro)
5:         if dist>minDist then
6:             minDist ← dist
7:         end if
8:     end for
9: end function
```



(a) (b)

Figure 5.6: To improve the efficiency of road network distance calculation, the content of H and CS are reused instead of executing the similar operation.

Tardy Rival Addition (TRA)

In PRA, each ro ($\in RO$) is retrieved by the trip route on Euclidean distance. The TR from each ro to d visiting $N-1$ categories must be determined in the road network distance. The processing time becomes large with the increase in number of RO. To search all possible ro incrementally, the found candidate RO must be removed from R-Tree index. For data object deletion, R-Tree index is needed to be copied into the main memory. When an R-Tree index is referred, a least recently

used (LRU) buffer is used to improve the I/O response. Therefore, this deletion is performed inside the buffer region to avoid the fact of soiling the R-Tree. To avoid deletion procedure, we studied an approximate algorithm called tardy rival addition algorithm.

(i) Algorithm

The overall processing of the TRA is the same with PRA. The main difference is the function called **ADDCANDIDTERO** as described in Algorithm 5.5. In comparing with PRA, TRA does not need to remove the rival object from the R-Tree and the copy of the R-Tree is not necessary.

Algorithm 5.5 ADDCANDIDTERO for TRA

1: **function** AddCandidateRO(r, RO, p_1)
2: $next \leftarrow NN(RTree[p_1], r.n, p_1)$
3: $RO \leftarrow RO \cup next$
4: **end function**

(ii) Obtaining Minimum Rival Objects Set

The principle of searching the minimal rival objects in TRA is finding the rival objects by invoking the nearest neighbor query on road network distance targeting to the first visiting category from the current notice node. The first visiting category is the category that the SR is requested by the MO. While retrieving the RO, the search area is gradually enlarged in the same way as the typical approach. The nearest neighbor objects except the data point probed from the TR from the first category are searched. And then, the found objects are inserted into RO set. In this method, the RO search can be limited by the vicinity of the current location, therefore, the number of the RO is likely to be reduced.

However, this method has possibility of missing the RO which makes an actual shape of the SR. When the number of RO is not sufficient, the size of SR tends to be enlarged compared with the actual SR size. Nevertheless, according to the experimental results, the size of SR becomes a few percentage larger than the

excepted size of SR. We can clarify the condition that overlooks the RO using the Figure 5.7.

In Figure 5.7, when n_1 and n_2 are checked, the route passing through p_1 is considered the shortest because the RO has not been found, and then the expansion is continued. When the node n_4 is checked, the object ro_1 is found as the first rival object according to the NN result. The length of the TR passing through ro_1 is shorter than the TR passing through p_1 from n_4. Therefore, the n_4 is not included in the SR. However, nodes n_1 and n_2 have to verify whether they are included in the SR because there is no rival object found when they are checked. Due to this reason, a check is needed to trace back along the path to reach n_4. When n_1 is verified, the n_1 is closer to p_1 than ro_1. The condition of the line 11 from Algorithm 5.1 is false and the whole link from n_1 will be included in SR. When n_2 is checked, the n_2 is closer to ro_1 than p_1 and the query condition is true in line 11. In this case, the border edge of safe-region will be on the link between n_1 and n_2. The main defect of TRA is it does not guarantee to find enough RO to form an exact shape of SR because some possible RO will be missed.

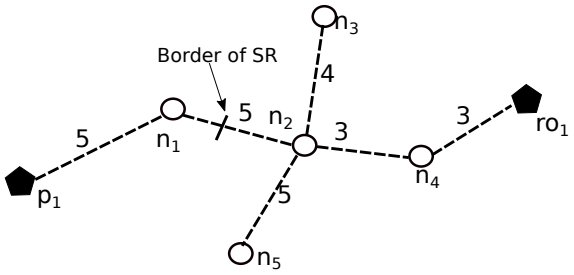


Figure 5.7: Safe-region generation by TRA.

5.4 Performance Study

In this section, we present the performance of each algorithms that we described in previous sections. We experimented with several scenarios to analyze the performance of the query processing and safe-region generation of each algorithms. In the first section of experiments, we conducted for the static TRP query com-

paring the performance of sparse category first (SCF) algorithm with the $OSRA$ algorithm. These algorithms are presented in section 5.1.2 and 5.1.3. Next, we examined the performance of TRP query for continuous processing with safe-region. We performed a comparative experimental study on three safe-region generation approaches mentioned in section 5.3.

5.4.1 Experimental Environment and Settings

All of the experiments were performed on a PC with Intel Core i7-4770 CPU ($3.4GHz$) and $16GB$ memory. All algorithms were implemented utilizing Java language. We evaluated the performance efficiency on two maps viz. small map consisting the center part of the Saitama city, and medium map is a medium size map consisting the center of the city and rural area. The detail information of each road map is described in Table 3.1. In addition, we described the environment of the data object distribution on the road map; we generated 1000 start and destination pairs samples randomly. Table 5.2 summarizes the parameter values used in the all experiments for TRP query.

Table 5.2: Parameters used in performance evaluation of continuous trip route planning queries

Parameter	Setting
Relationship between number of data objects in each category and the density of data objects	24(0.001), 48(0.002), 124(0.005), 249(0.01),498(0.02),1245(0.05)
Default density of sparse data category	0.001
Default density of dense data category	0.02
Number of data object categories	2, 3, 4, 5
Randomly generated start and destination pair	1000

5.4.2 Performance Evaluation

(1) TRP query on static environment

Figure 5.8 illustrates the performance of $SCFA$ and $OSRA$ approaches in which the sparse data category was set as the first visiting data category (C_1). We defined the trip route visits the four data categories (C_1 to C_4) and sparse category (C_1) was 0.001 and C_2 to C_4 were 0.02. We set the start and destination points are same which means a trip starts from "s" and visits the four categories then returns back to "s". In Figure 5.8, the horizontal axes show the trip route length in kilometer and the vertical axes represent the processing time and CPU time in second. The experimental results showed that the $SCFA$ needed longer processing time compared with the $OSRA$ because $SCFA$ needed extra processing time to define the sparse data category.

When the distribution of data categories greatly varies and the sparse data category was first visiting category, $OSRA$ algorithm performed well. In the case when the sparse category was first position, initially, $OSRA$ started from "s" and searched the data objects from the sparse data category concurrently and incrementally until it reached the final destination. In this situation, the number of partial route from "s" with sparse visiting category and expanded number of nodes were less because the late arrival records from the sparse category were omitted. The shortest route was considered as a optimal trip route. Consequently, the algorithm accelerated the searching for the next category until reaching to destination ("d").

When the sparse category was shifted to the next positions from Figure 5.9 to Figure 5.11, the processing time and CPU time of the $OSRA$ approach gradually increased compared with $SCFA$ approach. We can say that $SCFA$ approach is running with stable condition and does not depend on the position of the sparse category. We noticed that the processing time of $SCFA$ approach was relatively high in some situation as shown in the figures. The main reason is that we generate the start and destination sample pair and objects of interest on road network randomly. Therefore, when the existence of objects of interest are very less between the start and destination, a long network distance calculation time is required. Consequently, the total processing time increased.

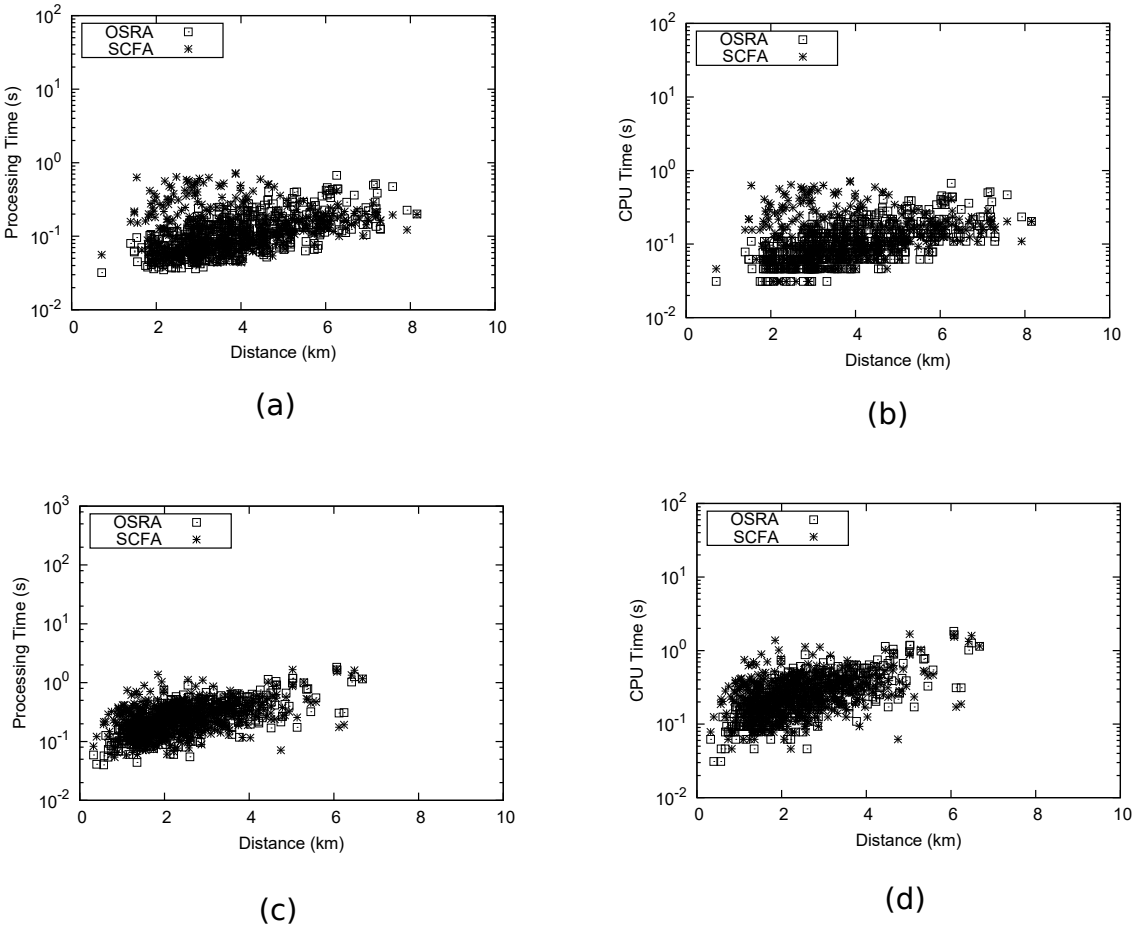


Figure 5.8: When the sparse data category was in first position, $OSRA$ and $SCFA$ approaches for snapshot $TRPQ$ (a) processing time on small map (b) CPU time on small map (c) processing time on medium map (d) CPU time on medium map.

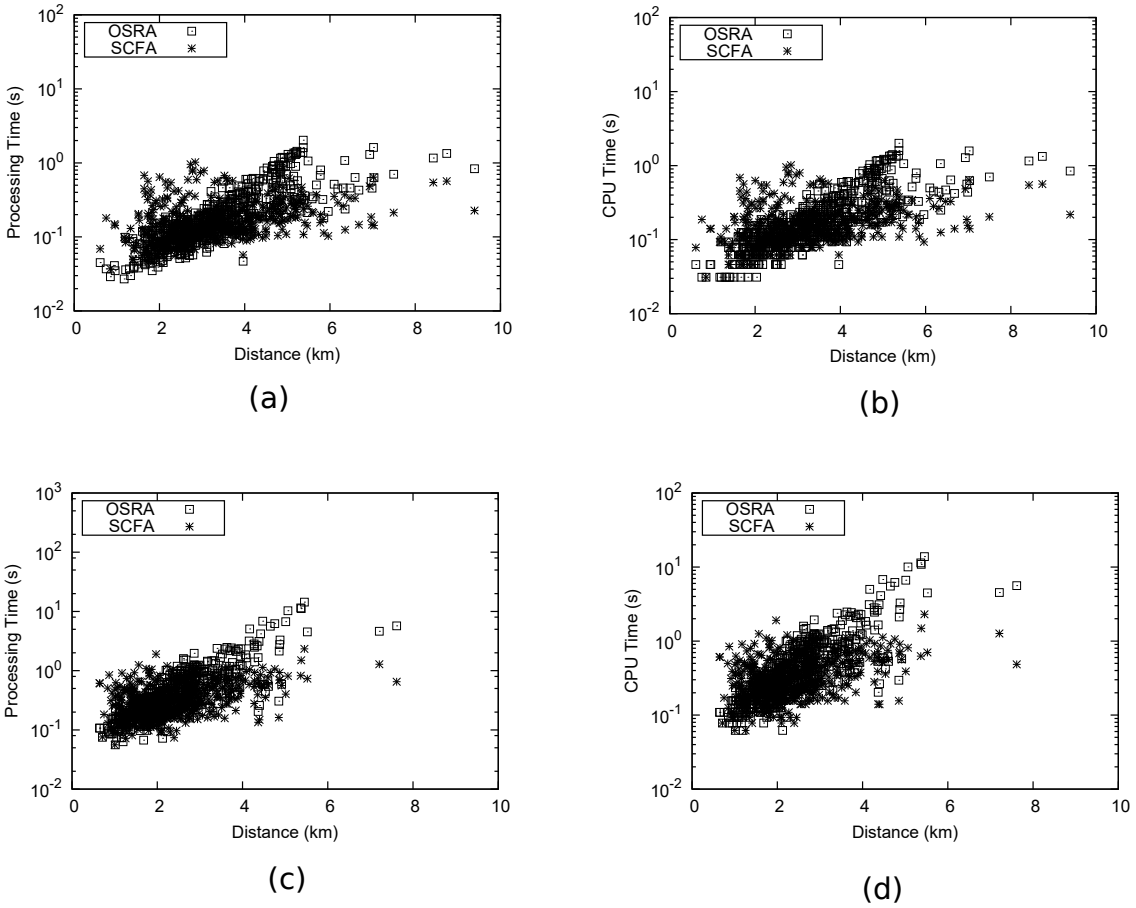


Figure 5.9: As the sparse data category was set at second position, $OSRA$ and $SCFA$ approaches for snapshot $TRPQ$ (a) processing time on small map (b) CPU time on small map (c) processing time comparison on medium map (d) CPU time comparison on medium map.

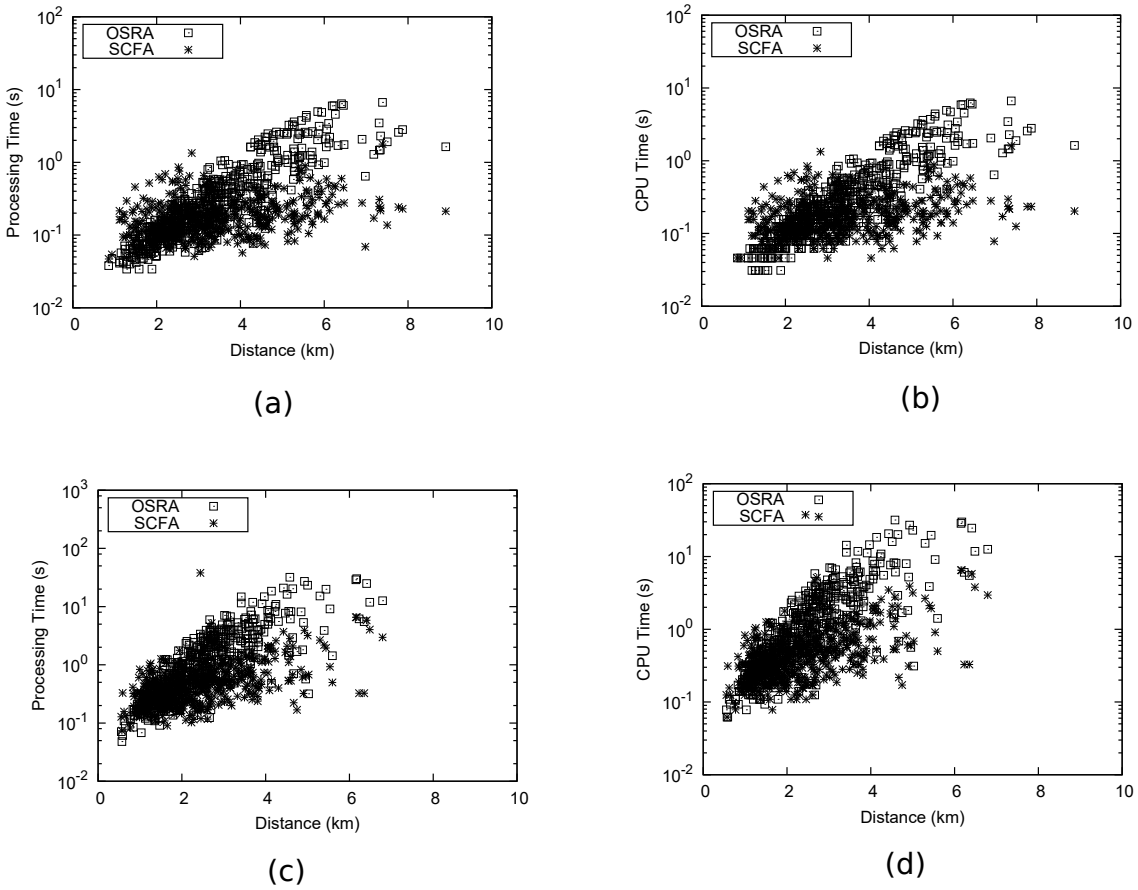


Figure 5.10: The position of sparse data category was moved to third position and $OSRA$ and $SCFA$ approaches for snapshot $TRPQ$ (a) processing time comparison on small map (b) CPU time comparison on small map (c) processing time comparison on medium map (d) CPU time comparison on medium map.

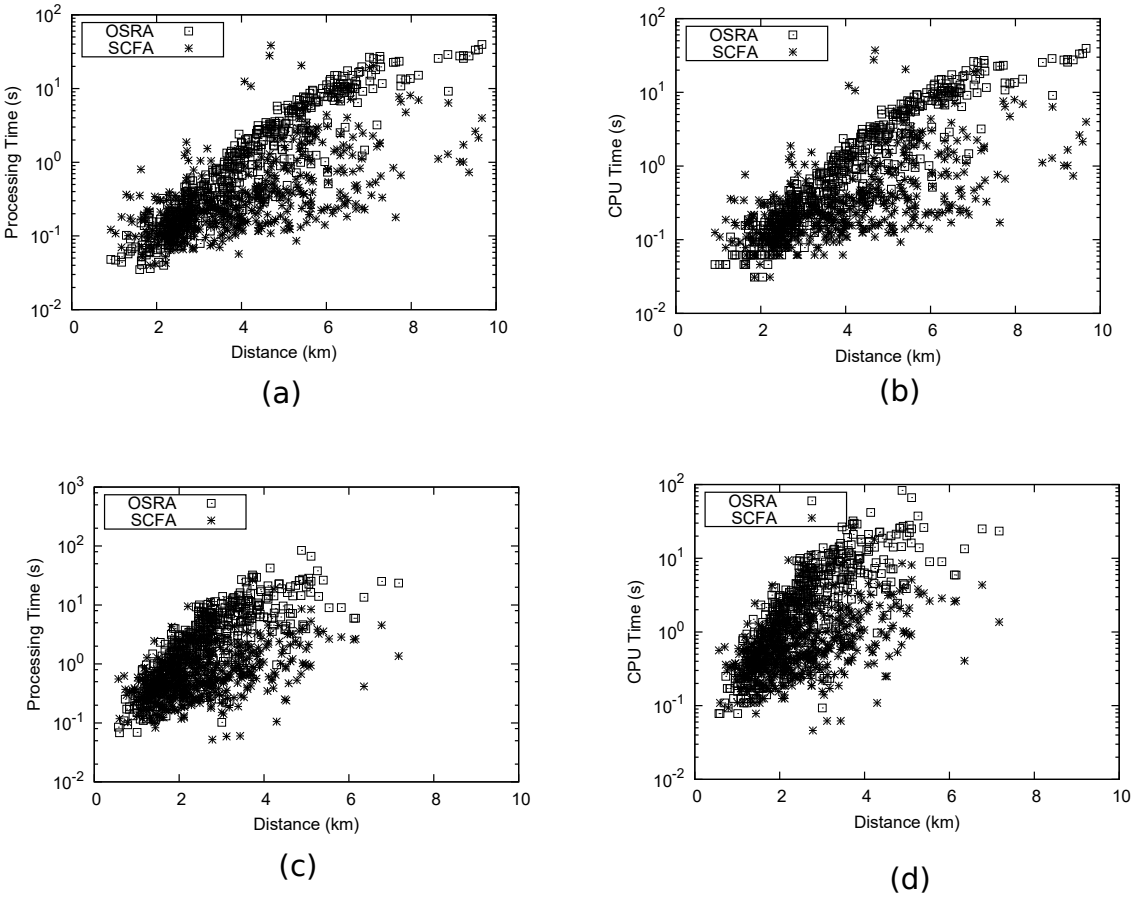


Figure 5.11: The position of sparse data category was changed to fourth position and examined $OSRA$ and $SCFA$ approaches for snapshot $TRPQ$ (a) processing time on small map (b) CPU time on small map (c) processing time on medium map (d) CPU time on medium map.

Figure 5.12 and 5.13 illustrate the average processing time of TR search. Figure 5.12 is the same experiment as the previous and illustrates the combination of previous results throughout the position of sparse category. For this experiment, we set the density sparse data category to 0.001. Figure 5.13 is the experimental result of the density of sparse category changed to 0.002. The results approved that $OSRA$ has a great effect in processing time according to the sparse data category position. On the other-hand, $SCFA$ has relatively stable processing time over any position of the sparse data category.

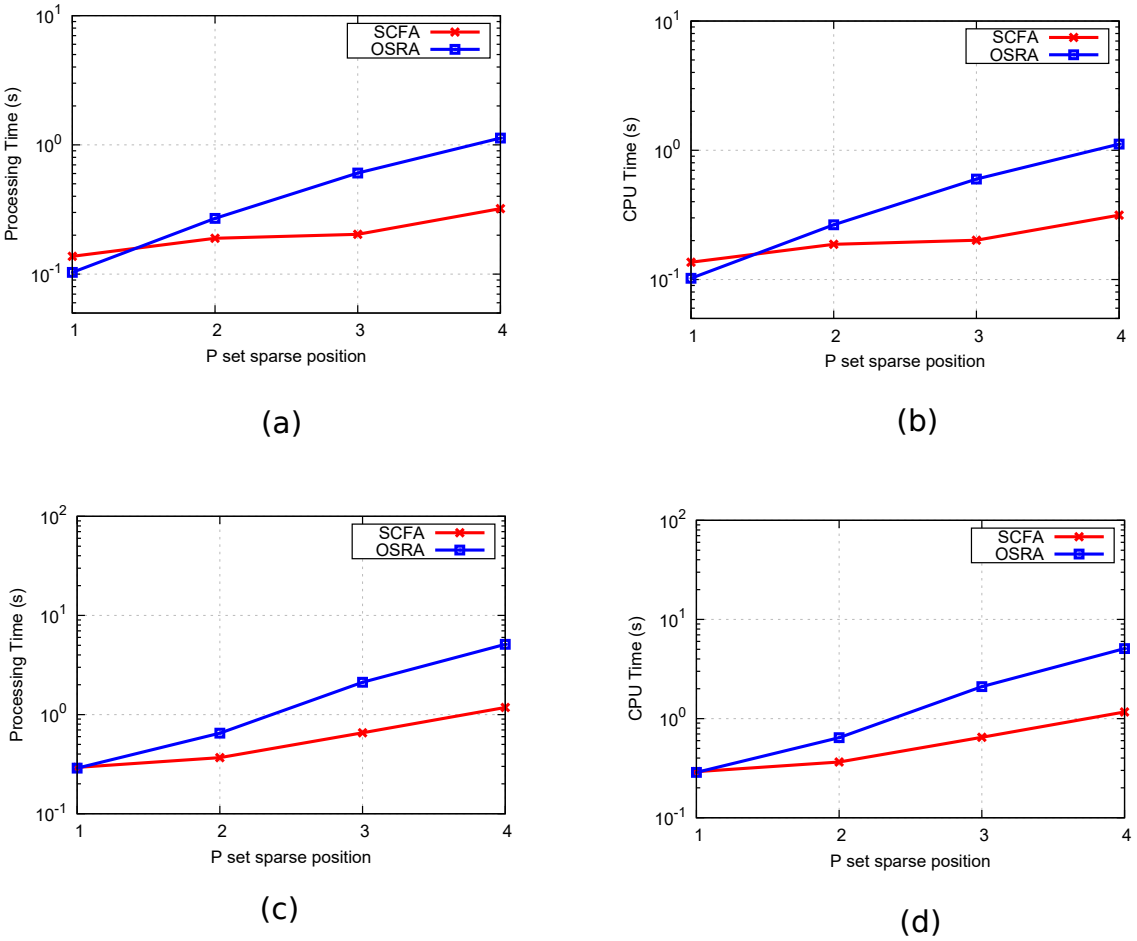


Figure 5.12: For the trip that starts from "s" and comes back "s", $OSRA$ and $SCFA$ approaches varying the position of sparse data category (data object density=0.001) (a) processing time on small map (b) CPU time small map (c) processing time on medium map (d) CPU time on medium map.

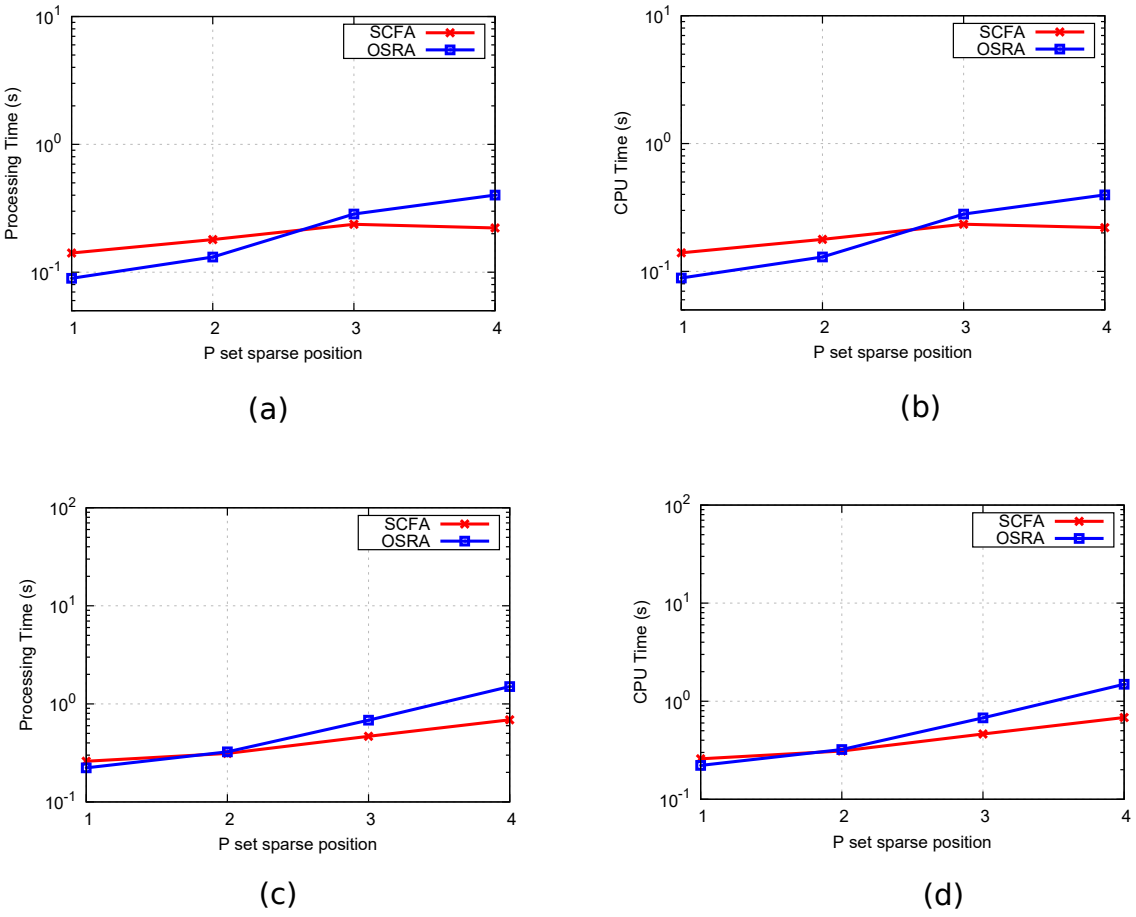


Figure 5.13: For the trip that "s" and "d" are same,, $OSRA$ and $SCFA$ approaches varying the position of sparse data category (data object density=0.002) (a) processing time on small map (b) CPU time small map (c) processing time on medium map (d) CPU time on medium map.

Next, we assumed the trip route as start and destination are different which means the trip does not return back to start position and analyzed the processing time of $OSRA$ and $SCFA$. The experimental setting and procedures are same as the previous experiments. The impact of the difference in start and destination points are illustrated in Figure 5.14 and Figure 5.15. Our conclusion on the $OSRA$ and $SCFA$ approach are still approved according to the results either destination of the trip come back to origin or not. Figure 5.16 shows the impact of the position of the sparse data category on the trip route with different start and destination

position. We examined the efficiencies by setting the density of sparse data category as 0.001 (Figure 5.16(a)-(b)) and 0.002 (Figure 5.16(c)-(d)). According to the result, $SCFA$ approach is suitable for the very sparse condition.

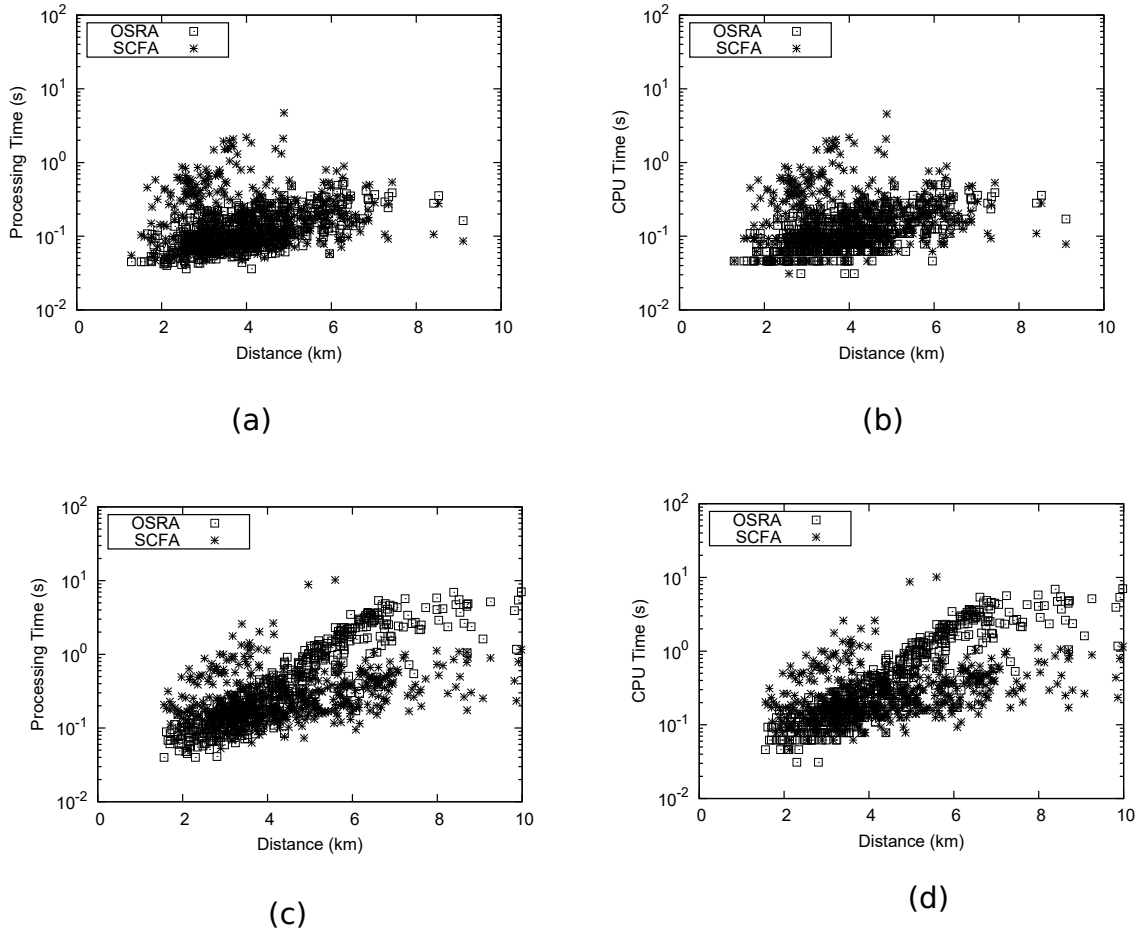


Figure 5.14: $OSRA$ and $SCFA$ approaches for snapshot $TRPQ$ (a) processing time when sparse category is located at 1^{st} position. (b) CPU time when sparse category was located at 1^{st} position (c) processing time when sparse category was located at 2^{nd} position (d) CPU time when sparse category was located at 2^{nd} position.

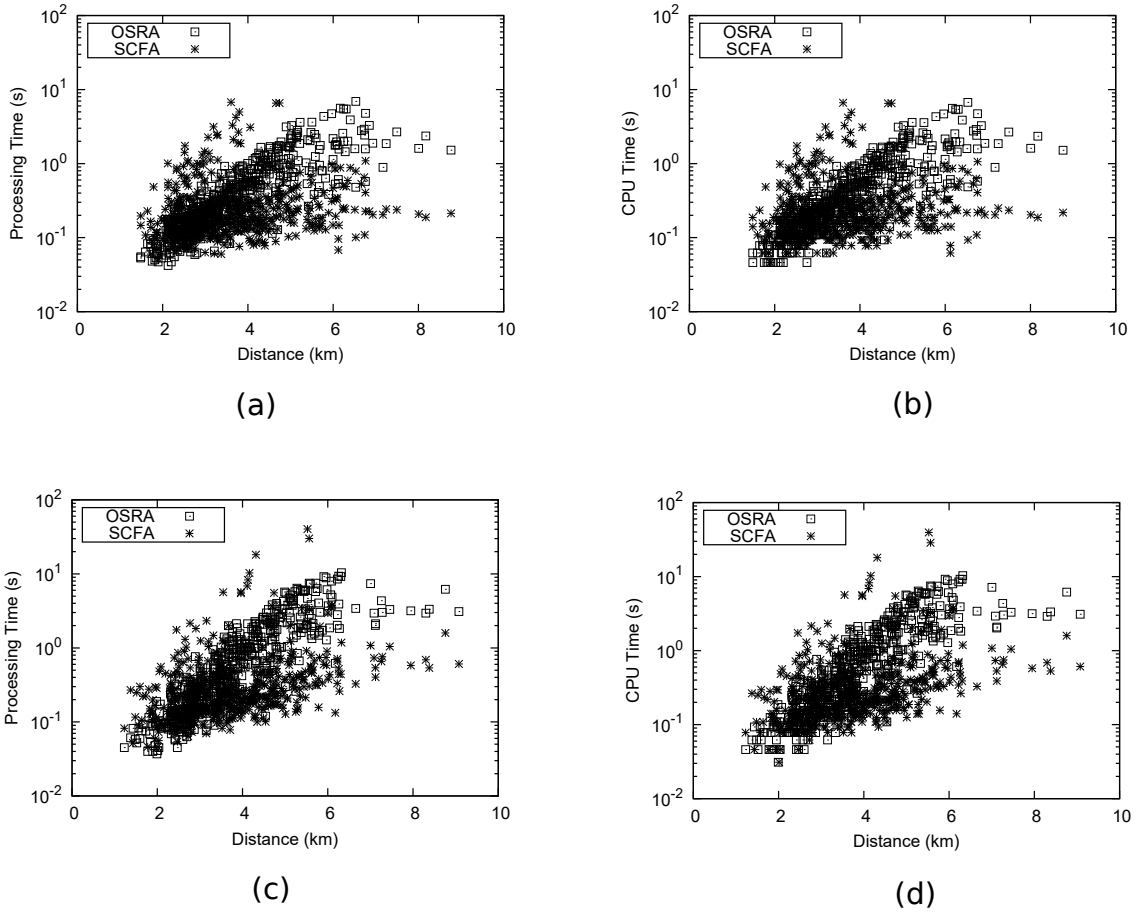


Figure 5.15: $OSRA$ and $SCFA$ approach for snapshot $TRPQ$ (a) processing time when sparse category is located at 3^{rd} position. (b) CPU time when sparse category was located at 3^{rd} position (c) processing time when sparse category was located at 4^{th} position (d) CPU time when sparse category was located at 4^{th} position.

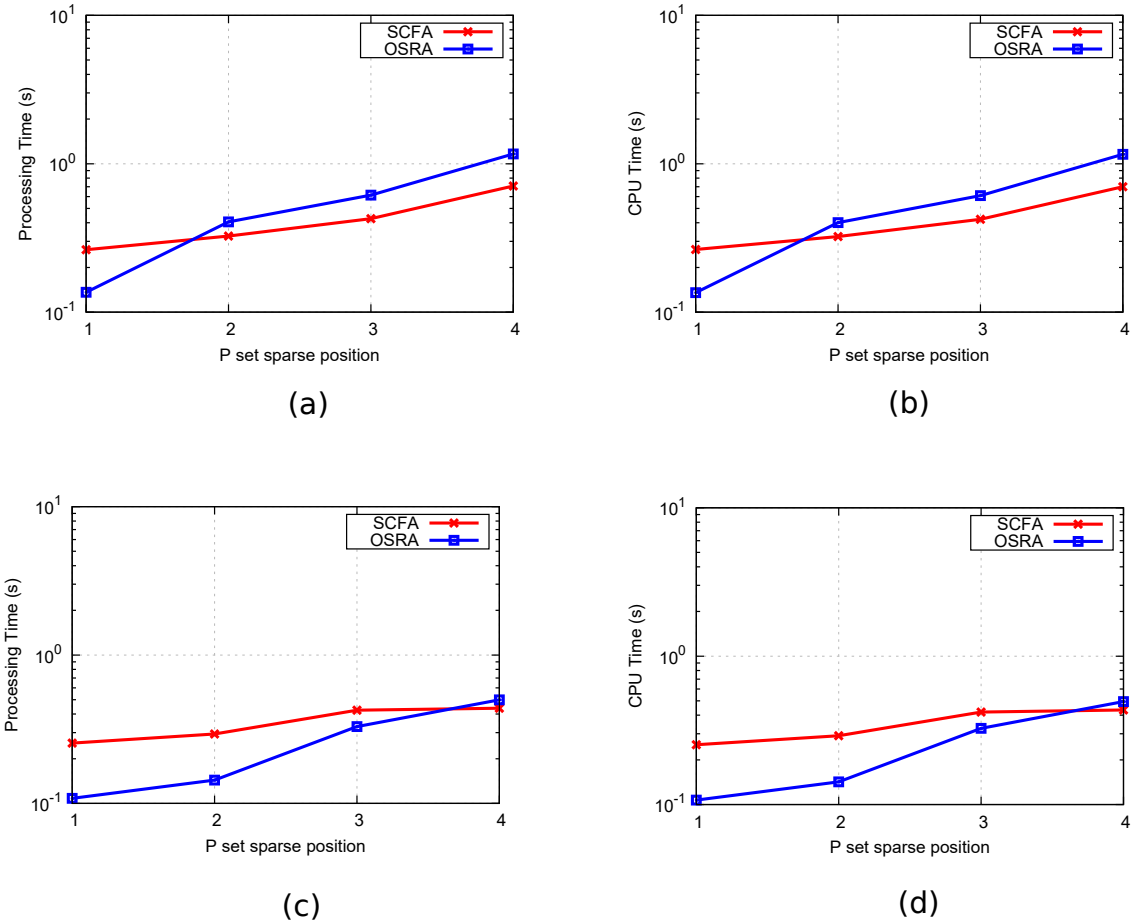


Figure 5.16: (a) Processing time when the density of sparse data category was 0.001. (b) CPU time when when the density of sparse data category was 0.001. (c) Processing time as the density of sparse data category was 0.002. (d) CPU time as the density of sparse data category was 0.002.

(2) TRP query on dynamic environment

In the next section of the experimental evaluation, we analyzed the performance of our approach for processing the TRP query continuously with safe-region. We conducted the experiments with two scenarios:

(1) a trip starts from "s" and visits from 2 to 5 data categories then returns back " s " .

(2) a trip starts from "s" to "d" and stops from 2 to 5 visiting data categories.

For first scenario, the start position and destination were same. While query object is traveling, the algorithms continuously monitor the optimal trip route with the visiting data categories using the safe-region. We made the comparison of three safe-region generation approaches on processing time in Figure 5.17.

In Figure 5.17(a) is illustrated for the result of the experiment is conducted with two data categories. We analyzed the processing time by increasing the number of data categories from 2 to 5. The results are shown in Figure 5.17(b)-(d) respectively. According to the experimental outcomes, TRA approach required least processing time. When the distribution of the data object on the road map was sparse, the processing time of PRA and TRA approaches were not much difference. However, while the data distribution was gradually increased, the processing time of PRA was more than TRA because PRA approach probed all possible rival objects among dense distributed data. As can be seen from the result, the processing time of basic approach (BA) was the highest among them. Remarkably, the trend of the BA and other approaches were radically different. Because BA algorithm runs snapshot TRP query continuously, it consumed huge processing time when the data distribution was sparse. There was a consistent decrease in the processing time with the increase in dense data distribution.

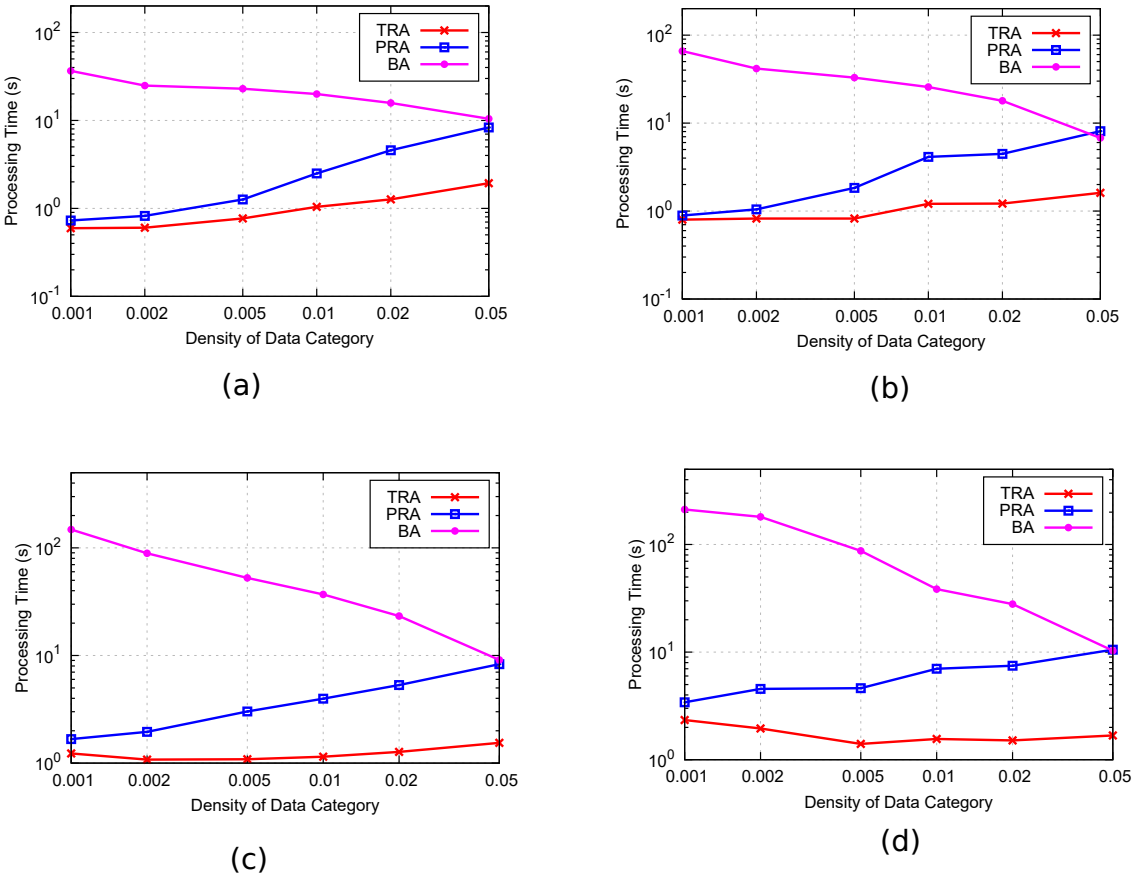


Figure 5.17: Safe-region generation time comparison of three approaches for continuous TRP query on small map (a) conducted the experiment with 2 data categories, (b) conducted with 3 data categories, (c) conducted with 4 data categories, (d) conducted with 5 data categories.

To make detail comparison of the processing time according to the various travel distance between start and destination, we describe the experimental result in Figure 5.18. In this experiment, we run the TRP query with 500 pairs of start and destination that are generated randomly. Based on their various travel distance, the processing time result of TRA and PRA are plotted. In this experiment, we assumed that the trip starts "s" and returns back to the origin and conducted on small map. In Figure 5.18, we confronted the impact of the SR generation time with the sparse and dense visiting data categories. When the density of visiting data category was dense, the trip route visiting data categories between random

start and destination pair was gradually short and the SR generation time of each approach was gradually increased.

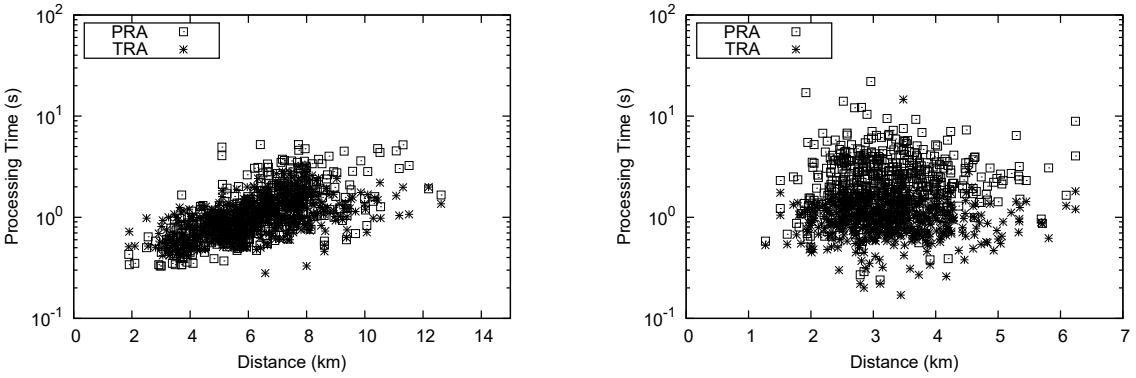


Figure 5.18: The detail SR generation time comparison between PRA and TRA based on the various trip route length when (a) the density of the visiting data category is 0.001, (b) the density of the visiting data category is 0.005

In second scenario, the start and destination location were different. We conducted the experiments using the same procedures with the previous experiments. According to the outcomes, TRA approach needed least processing time compared with others as illustrated in Figure 5.19. To make a comparison between TRA and PRA, we deduced that TRA approach ran fast without much depending on the distribution of the data objects.

Similarly, to compare the individual processing time between our two proposed algorithms for SR generation on the various trip route length, we conducted the same experiment with the Figure 5.18. In this experiment, our trip plan is followed with second scenario and the results are shown in Figure 5.20. According to the result, the processing time was slightly increased compared with the Figure 5.18. When "s" and "d" of a trip are same, the shortest path searching between the current position and "d" are not necessary and the pervious expanded network nodes and segments records inside CS can re-use to go back to the origin "s". Consequently, there is no more node expansion is occurred and we also save the processing

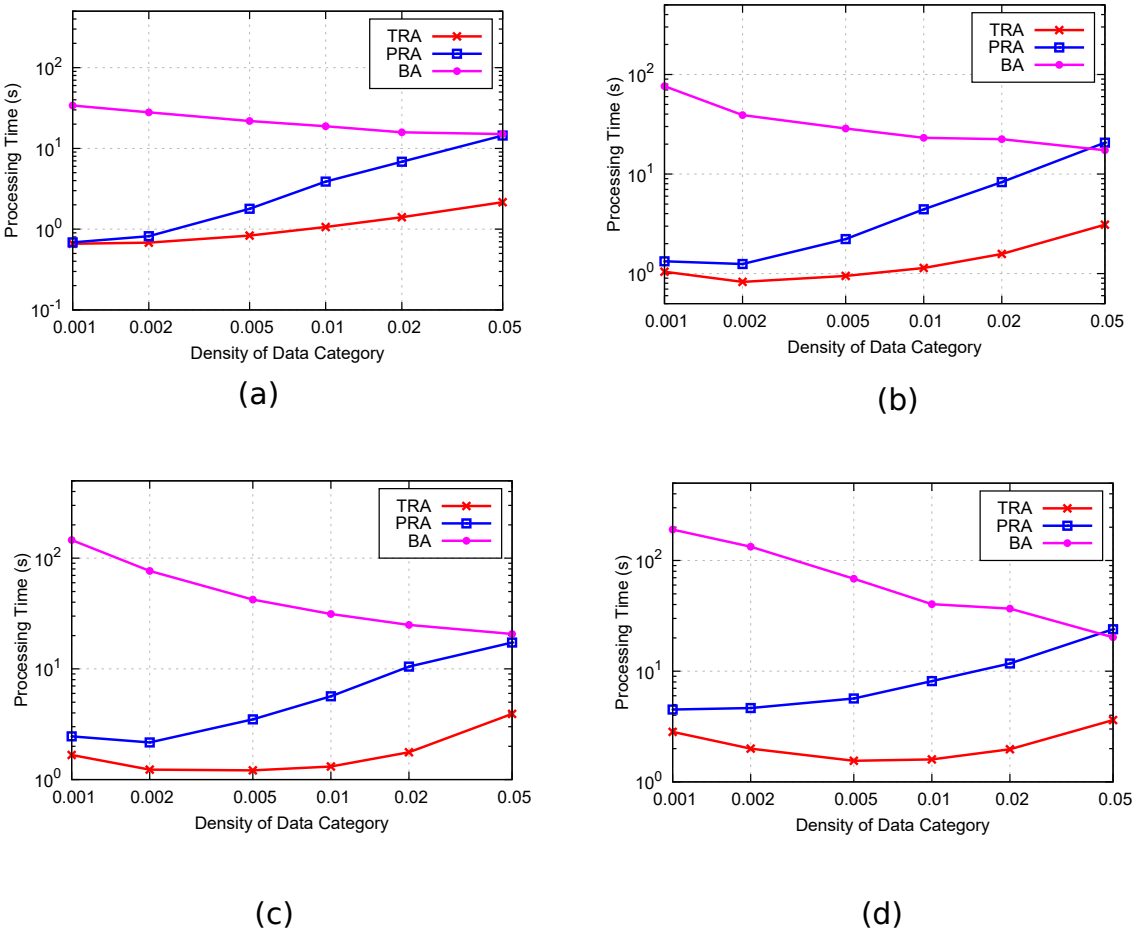


Figure 5.19: For trip starts "s" and ends "d", safe-region generation time comparison for continuous TRP query on small map (a) conducted the experiment with 2 visiting categories, (b) evaluated with 3 visiting categories, (c) evaluated with 4 visiting categories, (d) evaluated with 5 visiting categories.

time. On the contrary, when "s" and "d" are different, the extra processing time is required.

We noticed that the size of the safe-region generated by the TRA approach was slightly larger with small percentage (3% to 7%) than the PRA approach as illustrated in Figure 5.21. Note that the number of rival objects effect the size of the safe-region. The PRA approach incrementally assembles the enough rival objects until it satisfies the property 5.3. On the contrary, searching the rival object in TRA approach failed to assemble the actual rival objects that we explained in

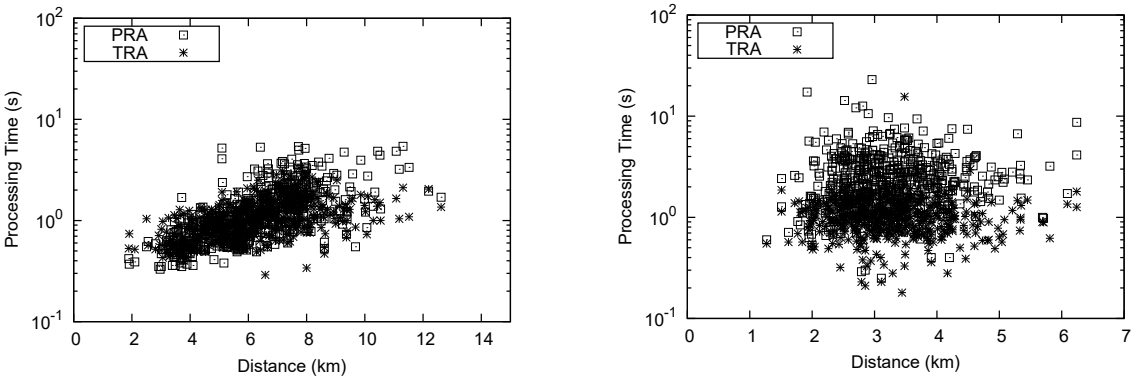


Figure 5.20: The detail SR generation time comparison between PRA and TRA based on the various trip route length when (a) the density of the visiting data category is 0.001, (b) the density of the visiting data category is 0.005

theoretical sections. When the number of rival objects was not adequate, the size of the safe-region intended to be large will be the main reason. In addition. we investigated that the size of safe-region is slightly large when the number of visiting data categories is less. When, the number of objects of interest existence on road network is less and the more the number of road segment it expands, the more the size of the SR it will be. The impact of the size of the SR according to the number of visiting categories and the density of objects of interest distribution is illustrated in Figure 5.21(a)-(d).

Finally, we compare the expanded node numbers while generating the safe-region of the current visiting data category. We examined the number of the expanded network nodes varying on the density of the visiting data category. In section 5.3.2, we have described that the way of node expansion of TRA approach is same as basic approach (BA). The average expanded node numbers of TRA and BA are same as shown in Figure 5.22. The average expanded nodes of PRA approach are slightly lesser than TRA and BA.

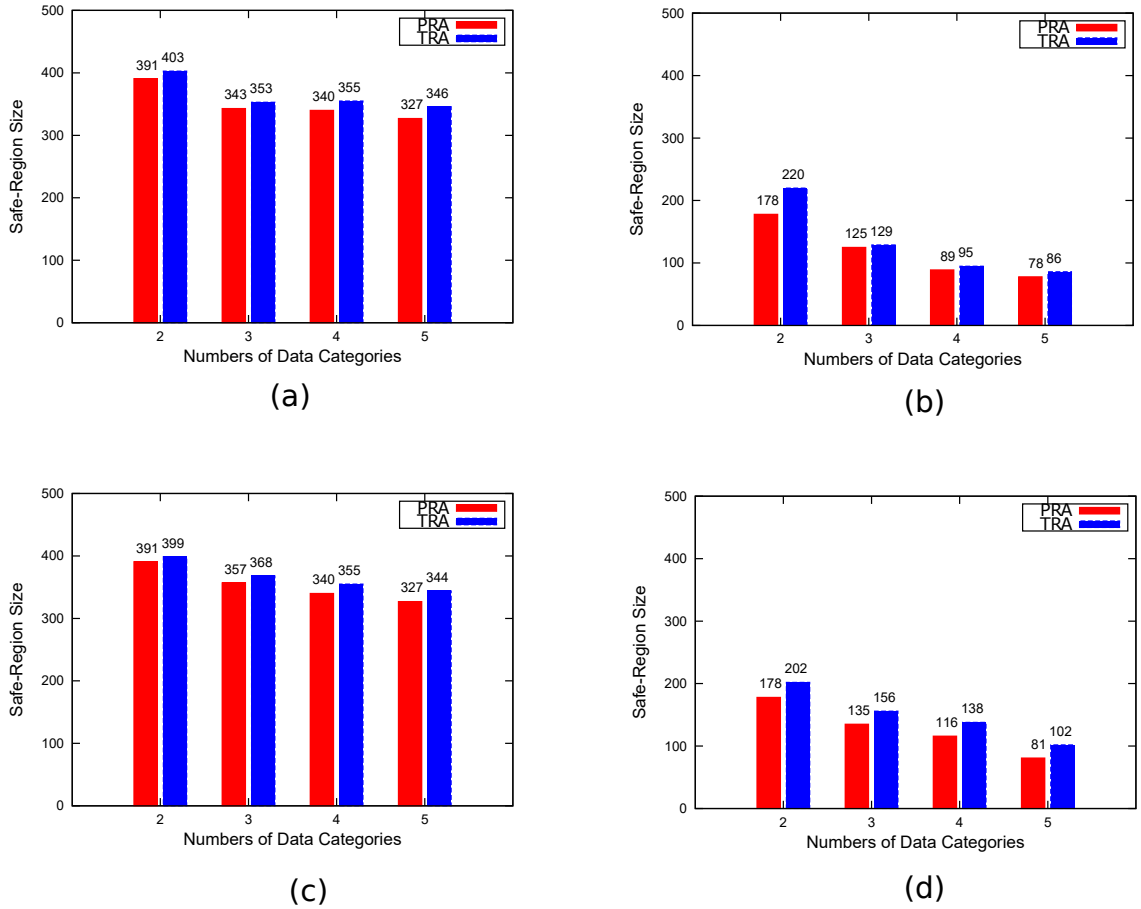


Figure 5.21: Safe-region size comparison of PRA and TRA approaches on varying the visiting data categories (a) evaluated with data point density was 0.005 ("s"= "d"), (b) evaluated with the density of data point was 0.05, (c) evaluated with data point density 0.005 ("s" $\neq$ "d"), (d) evaluated with the density of data point 0.05.

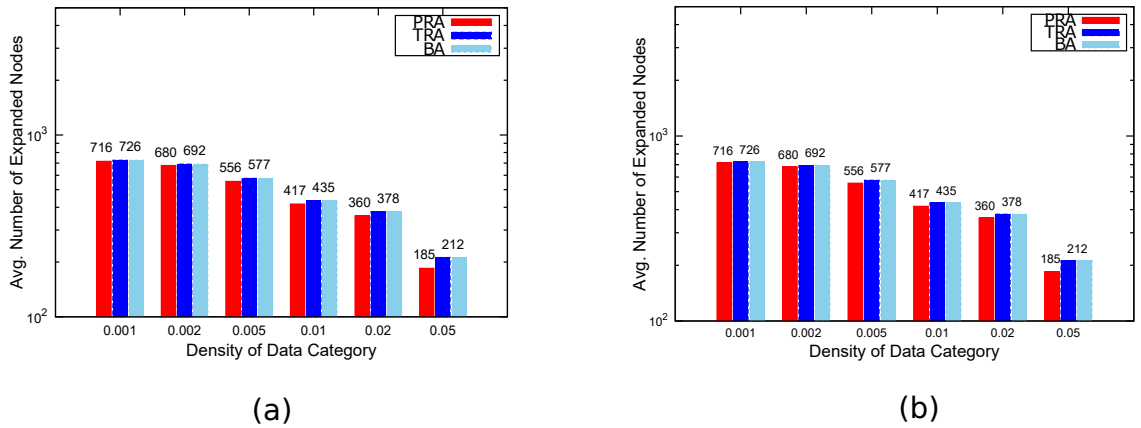


Figure 5.22: The average number of expanded nodes comparison for SR generation in $CTRPQ$ when (a) start and destination of trip are same, (b) trip starts at "s" and ends at"d".

5.5 Summary

The trip route planning query finds the optimal trip route with the multiple stopovers during the start and destination. It is a kind of complex and time consuming query even for static environments. Due to the rapid increase the number of location aware devices, the studies of the spatial queries with the moving objects are becoming essential. The spatial queries are issued by the moving objects or need to monitor the query result within a specified period to keep the latest query results are called continuous queries. There are several methodologies applied to perform the continuous query proficiently. Among them, safe-region approach is preferable to reduce the work load due to the several request to the server. A safe-region is an area in which the query result is valid as long as the moving object in it. The moving object issues location updates to the server only when it moves out of the safe-region.

In this chapter, we studied TRP queries for both static and moving objects. For static $TRPQ$, sparse first category is presented as a new approach to address the processing time issue when the distribution of data object is very sparse and the distribution of data object are distinct among the several categories. For continuous $TRPQ$, we mainly discussed the effective safe-region generation approaches. We presented two SR generation methods called the preceding rival addition (PRA) and tardy rival addition (TRA). Although the former approach (PRA) gives an accurate safe-region, it needs moderate processing time. The later algorithm runs fairly fast, however, it provides the approximate safe-region. We evaluated several experiments for performance efficiencies of our proposed approach by comparing with the typical safe-region generation approach.

CHAPTER 6

Conclusion and Future Work

With the rapid development in positioning technologies such as LBS application and mobile technologies, tracking the spatial objects such as road information and objects has become widespread in navigation systems. However, the main challenges in spatial data processing are implementing efficient and robust network distance calculation, efficiently manage the static or mobile objects location updates and lessen the query processing time. To cover these challenges, initially, we studied the snapshot $RkNN$ queries for stationary spatial objects using the materialized distance approach which is a kind of pre-computation approach utilizes the pre-computed distance tables. To broaden our studies, we extended our knowledge to process various spatial queries generated by the mobile query objects which is more complex because query object is not static. In addition, due to mobile query objects, its location and results are needed to be updated continuously to maintain correctness optimal result. For a such kind of spatial queries which require update frequently and heavy computation, in general, we focused to apply with on-the-fly query processing approach. We studied prevalent spatial queries such as range, NN queries and complex and time consumed query such as trip route query with mobile query objects.

Reverse k Nearest Neighbor Queries on Road Network

For static $RkNN$ query, our studies covered two cases namely monochromatic $RkNN$ ($MRkNN$) and bichromatic $RkNN$ ($BRkNN$) queries for road network. The basic principle of our approach for the $RkNN$ queries is to expand the search area in concentric circles. The algorithm searches the data point in a circle with the centered at the query point and radius equal to the distance between the query point and the border node of the subgraph. The data points found within the range need to examine whether they are the reverse data point of query object or not. The range searching is invoked on the border node reduces the overall processing time and the IER framework is adopted to perform searching and verifying the query results. In addition, to compute the road network efficiently, we utilized the distance materialization approach called $SMPV$ structure which is a kind of pre-computing approach. We compared our approach with the competitive method called eager algorithm. According to the experimental evaluations of $MRkNN$ and $BRkNN$ queries, eager algorithm requires longer processing time because the kNN search is invoked on every visited node. It causes a large amount of node expansion and consume longer processing time especially when the distribution data points are sparse.

Our proposed strategy was verified with the experimental results that it can retrieve the $RkNN$ objects in road network quickly especially when the data point distribution is sparse, and the arbitrary value k is large which is the main deficiencies of the existing methods. The good point of Eager algorithm is that when the data point density is high, it performs quickly as less searching range is necessary in the road network. The processing time of our proposed approach does not depend on the density of data point distribution. Therefore, when the data point is distributed densely, our proposed approach does not have great effect. Hence, we can direct the future work to advance towards an innovative approach by the combination of our method and eager algorithm to obtain an efficient and adaptive query which does not have direct impact due to data point distribution.

Safe-Region Generation Method for Versatile Continuous Vicinity Queries

We also expanded our studies in continuous spatial queries. Unlike the snapshot queries, continuous queries need to be evaluated at every time instance in order to ensure the correctness and validity of the query answer. In addition, if a query object is mobile object whose location changes continuously with time, it requires special handling from the database system. Using the traditional approaches such as time-based, distance-based update strategy causes high communication cost between the server and mobile object and update rates which deteriorate the database efficiencies. To guarantee providing an up-to-date query result and reducing the server workload and communication cost, we studied about safe-region based continuous queries in second section of this thesis. Although, there are several related researches works integrated with the safe-region and continuous queries there is little interest on the safe-region generation applicable for multiple types of spatial queries. Therefore, we proposed a versatile safe-region generation for various vicinity queries including $set - kNN$, $ordered - kNN$, $RkNN$ and distance range queries.

In our approach, we assumed that query object moves in any direction and the data objects are stationary. We constructed the safe-region corresponding to the query object location which satisfies with the specific query types. Only when the query object moves with invalid query condition (explained in chapter 4), the moving object requests a new safe-region and query result to the server. During the generation of the safe-region, the algorithm gradually expands the road network node and segments from the mobile query objects until the query condition satisfied. We guarantee that our approach does not access repeatedly the same network node during the node expansion by keeping the expanded nodes in the closed set and auxiliary object set. In addition, we proposed a run-time distance materialization approach to advance the network distance calculation time. Note that a safe-region corresponds to a region in a high order Voronoi diagram. Therefore, when the data points are biasedly distributed, the size of the safe-regions is affected by the

distribution of the data points located around the query point. When the data points are distributed sparsely, the size of safe-region will enlarge due to the less occurrence of the rival objects environ in the safe-region. Consequently, it increases the processing time. One possible limitation of our approach is that it is a little sensitive to the size of the safe-region. Therefore, practically, the upper limit of the size of the safe-region should be considered to avoid such case. There are some directions of future research: generalizing safe-region for more complicated spatial queries and enhancing the approach to apply over the moving data objects.

Continuous Trip Route Planning Query (CTRPQ)

As next extended studies in this thesis, the trip route planning query ($TRPQ$) for both static and continuous querying are presented. The $TRPQ$ is applied to retrieve the shortest trip route with multiple stopovers between start and destination. We have known that the main challenges of the trip route query either in snapshot or continuous queries is the requirement of the large processing time especially when the data points are distributed sparsely on the road network. To solve this issue, first, we attempted for snapshot $TRPQ$ and introduced the new concept called sparse category first (SCF) approach in order to shorten the processing time on the sparse road network. Our proposed approach is intended to apply when the data point distribution of each data point category is not the same. The experimental results proved that our approach achieved our main purpose. For these experiments, we made the performance comparison of SCF algorithm with the $OSRA*$ that probes the trip route based on the $A*$ algorithm.

Additionally, there is no previous studies for TRP query to handle continuous processing. We investigated continuous TRP query where a query object is moving arbitrary directions and data objects on the road network are stationary. The efficient way to update and monitor the MO continuously is using safe-region approach, therefore, we proposed two approaches for generating the safe-region. One is called preceding rival addition (PRA) which gives an accurate safe-region

and next approach is named tardy rival addition (TRA) which can generate the safe-region fast, but the safe-region is only for approximate. Although PRA can generate the accurate safe-region, the processing time inclines linearly according to the dense data point distribution on the road network. In contrast, TRA algorithm does not depends on the distribution of the data objects. However, the safe-region generated by TRA becomes about 3% to 7% larger than the safe-region of PRA. The size of the safe-region is relating to the number of occurrence rival objects environs the mobile query object.

We conducted several experiments to evaluate our proposed methods with the basic algorithm called BA. The basic algorithm finds many redundant rival objects when the density of the data point is high. This characteristic requires longer time and declines the performance gradually. We guarantee that our approach does not have duplicate accessing. PRA searches the rival objects in Euclidean distance, and then compares the trip route length between the target object and the rival objects until the equation is satisfied. PRA retrieves all rival objects while TRA retrieves only those rival objects which lies in the neighborhood of the target objects. Therefore, the processing time is short and the main issue of TRA is that it does not guarantee to find the minimal rival objects. There are some possible directions for future works: (1) examining our approaches on both query and data objects are mobile, (2) Analyzing the TRA issue in finding the minimal rival objects and providing the accurate safe-region efficiently, and (3) implementing in real world location-aware devices.

In a nutshell, this thesis addresses several spatial queries algorithms to deal with various spatial objects (static and moving object) in a road network. Our approaches and proposals are targeted to address the main challenges in spatial data processing for the various aspects of the spatial objects and spatial database. According to the performance studies of our proposed approaches, the results indicate that our methodologies are effective and feasible for lightweight location-based service applications.

References

[1] E.W.Dijkstra.: A note on two problems in connexion with graphs. Numerische Mathematik, Vol. 1(**1**), pages 269-271,1959.

[2] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Trans. Systems Sci. Cybernet, Vol. 4(**2**), pages 100-107,1968.

[3] R. Bayer and E. M. McCreight.: Organization and Maintenance of Large Ordered Indexes. Acta Informatica, 1(**3**), pages 173-189, 1972.

[4] Antonin Guttman and Michael Stonebraker: R-Trees: a Dynamic Index Structure for Spatial Searching. In Proceeding of ACM SIGMOD Conference on Management of Data, pages 47-57, 1984.

[5] Robert Sedgewick and Jeffrey Scott Vitter: Shortest path with Euclidean graphs. Algorithmica, Vol. 1(**1**), pages 31-48, 1986.

[6] Hanan Samet: The design and analysis of spatial data structures. Addison-Wesley Publishing Co., 1989.

[7] Rakesh Agrawal and H.V. Jagadish: Materialization and incremental update of path information. In: Proceedings of 5^{th} International Conference on Data Engineering, Vol. 1, pages 374-383, 1989.

[8] Nobert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger: The R*-tree: An Efficient and Robust Access Method for Points and Rectan-

gles. In Proceeding of SIGMOD international conference on Management of data, pages 322-331, 1990.

[9] Bing Liu and J.Tay: Using Knowledge about the road network for route finding. In Proceeding of the 11^{th} Conference on Artificial Intelligence for Applications, pages 306-312, 1995.

[10] Nick Roussopoulos, Stephen Kelley, and Frederic Vincent: Nearest neighbor queries. In Proceedings of ACM SIGMOD, Vol. 24(**2**), pages 71-79, 1995.

[11] Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner: Hierarchical optimization of optimal path finding for transportation applications. In Proceedings of the 5^{th} International Conference of Information and Knowledge Management, pages 261-268, 1996.

[12] Stefan Berchtold, Bernhard Ertl, Daniel A. Keim, Hans-Peter Kriegel, and Thomas Seidl: Fast Nearest Neighbor Search in High-Dimensional Space. In Proceedings of the 14^{th} International Conference on Data Engineering, pages 209-218, 1998.

[13] Gisli R. Hjaltason and Hanan Samet: Distance Browsing in Spatial Databases. ACM Transactions on Database Systems, Vol. 24(**2**), pages 265-318, 1999.

[14] Atuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu: Spatial Tessellations, Concepts and Applications of Voronoi Diagrams, 2^{nd} ed.. John Wiley and Sons, 2000.

[15] Filip Korn and S. Muthukrishnan: Influence Sets Based on Reverse Nearest Neighbor Queries. ACM SIGMOD Record, Vol. 29(**2**), pages 201-212, 2000.

[16] Ioana Stanoi, Divyakant Agawal, and Arm El Abbadi: Reverse Nearest Neighbor Queries for Dynamic Databases. In Proceedings of ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pages 44-53, 2000.

[17] Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopezz: Indexing the positions of continuously moving objects. In Proceedings of 2000 ACM SIGMOD International Conference on Management of Data, Vol. 29(**2**), pages 331-342, 2000.

[18] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker: The Skyline Operator. Proceeding of the 17^{th} International Conference on Data Engineering, pages 421-430, 2001.

[19] Donald Kossmann, Frank Ramsak, and Steffen Rost: Shooting stars in the sky: An online algorithm for skyline queries. In Proceedings of the 28^{th} International Conference on Very Large Database, pages 275-286, 2002.

[20] Iosif Lazaridis, Kriengkrai Porkaew, and Sharad Mehrotra: Dynamic queries over mobile queries. In Proceedings of the 8^{th} International Conference on Extending Database Technology, LNCS, Vol. 2287, pages 269-286, 2002.

[21] Yufei Tao, Dimitris Papadias, and Qiongmao Shen: Continuous nearest neighbor search. In Proceeding of the 28^{th} International Conference on Very Large Database, pages 287-298, 2002.

[22] Rimantas Benetis, Christian S. Jensen, Gytis Karciauskas, and Simonas Saltinis: Nearest neighbor and reverse nearest neighbor queries for moving objects. In Proceedings International Database Engineering and Applications Symposium, pages 44-53, 2002.

[23] Sunil Prabhakar, Yuni Xis, Dmitri Kalashnikov, Walid G. Aref, and Susanne E. Hambrusch: Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. IEEE Transactions on Computers, Vol. 51(**10**), pages 1124-1140, 2002.

[24] Yufei Tao and Dimitris Papadias: Time-parameterized queries in spatial-temporal database. In SIGMOD Conference, pages 334-345, 2002.

[25] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger: An optimal and progressive algorithm for skyline queries. In Proceedings of the ACM SIGMOD international conference on Management of data, pages 467-478, 2003.

[26] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao: Query processing in spatial network databases. In Proceedings of the International Conference on Very Large Databases, Vol. 29, pages 802-813, 2003.

[27] Yufei Tao and Dimitris Papadias: Spatial Queries in Dynamic Environments. ACM Transactions on Database Systems, Vol. 28(**2**), pages 101-139, 2003.

[28] Glenn S. Iwerks, Hanan Samet, and Ken Smith: Continuous k-Nearest Neighbor Queries for Continuously Moving Points with Updates. In Proceedings of the Very Large Database, Vol. 29, pages 512-523, 2003.

[29] Jun Zhang, Manli Zhu, Dimitris Papadias, Yufei Tao, and Dik Lun Lee: Location-based spatial queries. In Proceedings of the ACM SIGMOD international conference on Management of data, pages 443-454, 2003.

[30] Yufei Tao, Dimitris Papadias, and Jimeng Sun: The TPR*-tree: an optimized spatio-temporal access method for predictive queries. In Proceedings of the International Conference on Very Large Databases, Vol. 29, pages 790-801, 2003.

[31] Yufei Tao, Dimitris Papadias, and Xiang Lian: Reverse kNN search in arbitrary dimensionality. In Proceedings of the 30^{th} International Conference on Very Large Databases, Vol. 30, pages 744-755, 2004.

[32] Mohammad R. Kolahdouzan and Cyrus Shahabi: Voronoi-based k nearest neighbor search for spatial network databases. In Proceedings of the 30^{th} International Conference on Very Large Databases, Vol. 30, pages 840-851, 2004.

[33] Mohammad R. Kolahdouzan and Cyrus Shahabi: Continuous k Nearest Neigh-

bor Queries in Spatial Network Databases. In Proceedings of the n^{nd} International Workshop on Spatio-Temporal Database Management, pages 33-40, 2004.

[34] Bugra Gedik and Ling Liu: Mobieyes: distributed processing of continuously moving queries on moving objects in a mobile system. In Proceedings of International Conference on Extending Database Technology, LNCS, Vol. 2992, pages 67-87, 2004.

[35] Mohamed F. Mokbel, Xiaopeng Xiong, and Walid G. Aref: SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In SIGMOD Conference, pages 623-634, 2004.

[36] Yufei Tao, Christos Faloutsos, Dimitris Papadias, and B. Liu: Prediction and indexing of moving objects with unknown motion patterns. In Proceedings of the ACM SIGMOD international conference on Management of data, pages 611-622, 2004.

[37] Glenn S. Iwerks and Hanan Samet, and Kenneth P. Smith: Maintenance of Spatial Semijoin Queries on Moving Points. In Proceedings of the 30^{th} International Conference on Very Large Database, Vol. 30, pages 828-839, 2004.

[38] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng: On trip planning queries in spatial databases. In Proceeding SSTD'05 Proceedings of the 9^{th} international conference on Advances in Spatial and Temporal Databases, LNCS 3633, pages 273-290, 2005.

[39] Mehdi Sharifzadeh, Mohammad Kolahdouzan, and Cyrus Shahabi: The optimal sequenced route query. The International Journal on Very Large Databases, pages 765-787, 2005.

[40] Kyriakos Mouratidis, Marios Hadjieleftheriou, and Dimitris Papadias: Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In Proceedings of the ACM SIGMOD international conference on

Management of data, pages 634-645, 2005.

[41] Mohammad R. Kolahdouzan and Cyrus Shahabi: Alternative solution for continuous k nearest neighbor queries in spatial network databases. Geoinformatica, Vol. 9(**4**), pages 321-341, 2005.

[42] Maytham Safar: k nearest neighbor search in navigation systems. Mobile Information Systems, Vol. 1(**3**), pages 207-224, 2005.

[43] Haibo Hu, Jianliang Xu, and Dik Lun Lee: A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. In Proceedings of the ACM SIGMOD international conference on Management of data, pages 479-490, 2005.

[44] Kyriakos Mouratidis, Dimitris Papadias, S. Bakiras, and Yufei Tao: A Threshold-Based Algorithm for Continuous Monitoring of k Nearest Neighbors. IEEE Transition on Knowledge and Data Engineering, Vol. 17(**11**), pages 1451-1464, 2005.

[45] Man Lung, Nikos Mamoulis, and Dimitris Papadias: Aggregate Nearest Neighbor Queries in Road Networks. IEEE Transition on Knowledge and Data Engineering, Vol. 17(**6**), 2005.

[46] Fuyu Liu, Tai T. Do, and Kien A. Hua: Dynamic range query in spatial network environments. Database and Expert Systems Applications (DEXA), pages 254-265, 2006.

[47] Man Lung Yiu, Dimitris Papadias, Nikos Mamoulis, and Yufei Tao: Reverse Nearest Neighbor in Large Graphs. IEEE Transaction on Knowledge and Data Engineering, Vol. 18(**4**), pages 540-553, 2006.

[48] Haojun Wang, Roger Zimmermann, and Wei-Shinn Ku: Distributed continuous range query processing on moving objects. Database and Expert Systems Applications (DEXA), LNCS Vol. 4080, pages 655-665, 2006.

[49] Xiaobin Ma, Shashi Shekhar, Hui Xiong, and Pusheng Zhang: Exploiting a Page-Level Upper Bound for Multi-Type Nearest Neighbor Queries. In Proceedings of the 14^{th} ACM International Symposium on Geographic Information Systems (ACM-GIS), pages 179-186, 2006.

[50] Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias, and Nikos Mamoulis: Continuous Nearest Neighbor Monitoring in Road Networks. In Proceedings of the 32^{nd} International Conference on Very Large Databases, pages 43-54, 2006.

[51] James M. Kang, Mohamed F. Mokbel, Shashi Shekhar, Tian Xia, and Donghui Zhang: Continuous Evaluation of Monochromatic and Bichromatic Reverse Nearest Neighbors. IEEE 23^{rd} International Conference on Data Engineering (ICDE'07), pages 806-815, 2007.

[52] Dragan Stojanovic, Apostolos N. Papadopoulos, Bratislav Predic, Slobodanka Djordjevic-Kajan, and Alexandros Nanopoulos: Continuous range monitoring of mobile objects in road networks. Data and Knowledge Engineering, Vol. 64(**1**), pages 77-100, 2007.

[53] Reynold Chenga, Kam-Yiu Lamb, Sunil Prabhakarc and Biyu Liang: An efficient location update mechanism for continuous queries over moving objects. Information Systems, Vol. 32(**4**), pages 593-620, 2007.

[54] Hanan Samet, Jagan Sankaranarayanan, and Houman Alborzi: Scalable Network Distance Browsing in Spatial Databases. In Proceedings of the ACM SIGMOD Conference, pages 43-54, 2008.

[55] Wei Wu, Fei Yang, Chee-Yong Chan, and Kian-lee Tan: FINCH: Evaluating Reverse k-Nearest-Neighbor Queries on Location Data. In PVLDB'08, Vol. 1(**1**), pages 1056-1067, 2008.

[56] Wei Wu, Fei Yang, Chee-Yong Chan, and Kian-lee Tan: Continuous Reverse k-Nearest Neighbor Monitoring. In 9^{th} International Conference on Mobile

Data Management, pages 132-139, 2008.

[57] Sarana Nutanong, Rui Zhang, Egemen Tanin, and Lars Kulik: The V*Diagram:A Query Dependent Approach to Moving kNN Queries. In Proceedings of the VLDB Endowment, Vol. 1(**1**), pages 1095-1106, 2008.

[58] Huan-Liang Sun, Chao Jiang, Jun-Ling Liu, and Limei Sun: Continuous reverse nearest neighbor queries on moving objects in road networks. In 9^{th} International Conference on Web-Age Information Management, pages 238-245, 2008.

[59] Haiquan Chen, Wei Shin Ku, Min Te Sun, and Roger Zimmermann: The Multi-Rule Partial Sequenced Route Query. ACM GIS'08, pages 65-74, 2008.

[60] Mehdi Sharifzadeh and Cyrus Shahabi: Processing Optimal Sequenced Route Queries Using Voronoi Diagrams. GeoInformatica, pages 411-433, 2008.

[61] Xiang Lian and Lei Chen: Monochromatic and bichromatic reverse skyline search over uncertain database. In Proceedings of the ACM SIGMOD international conference on Management of data, pages 213-226, 2008.

[62] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein: Introduction to Algorithms, 3^{rd} ed.(MIT Press). Harcourt, 2009.

[63] Maytham Safar, Dariush Ibrahimi, and David Taniar: Voronoi-based reverse nearest neighbor query processing on spatial networks. Multimedia Systems, Vol. 15(**5**), pages 295-308, 2009.

[64] Quoc Thai Tran, David Taniar, and Maytham Safar: Reverse k nearest neighbor and reverse farthest neighbor search on spatial networks. Large-Scale Data and Knowledge-Centered Systems, LNCS, Vol. 5740, pages 353-372, 2009.

[65] Mahady Hasan, Muhammad Aamir Cheema, Xuemin Lin, and Ying Zhang: Efficient construction of safe regions for moving $k$nn queries over dynamic datasets. In Proceedings of the 11^{th} International Symposium of Spatial and

Temporal Database, pages 373-379, 2009.

[66] Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, WeiWang, and Wenjie Zhang: Lazy updates: An efficient technique to continuously monitoring reverse $k$nn. In Proceedings of the VLDB Endowment, Vol. 2(**1**), pages 1138-1149, 2009.

[67] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, and Jeffrey Xu Yu: Monitoring Path Nearest Neighbor in Road Networks. In Proceedings of the ACM SIGMOD International Conference on Management of data, pages 591-602, 2009.

[68] Kefeng Xuan, Geng Zhao, David Taniar, Bala Srinivasan, Maytham Safar, and Marina Gavrilova: Network Voronoi Diagram based Range Search. In Proceedings of International Conference on Advanced Information Networking and Applications, pages 741-748, 2009.

[69] Ke Deng, Xiaofang Zhou, Heng Tao Shen, Shazia Sadiq, and Xue Li: Instance Optimal Query Processing in Spatial Networks. The VLDB Journal, Vol. 18(**3**), pages 675-693, 2009.

[70] Quoc Thai Tran, David Taniar, and Maytham Safar: Bichromatic reverse nearest-neighbor search in mobile systems. IEEE Systems Journal, Vol. 4(**2**), pages 230-242, 2010.

[71] Sarana Nutanong, Egemen Tanin, Mohammed Eunus Ali, and Lars Kulik: Local Network Voronoi Diagrams. International Symposium on Advances in Geographic Information Systems 18^{th} ACM, pages 109-118, 2010.

[72] David Taniar, Maytham Safar, Quoc Thai Tran, J. Wenny Rahayu, and Jong Hyuk Park: Spatial network RNN queries in GIS. Computer Journal, Vol. 54(**4**), pages 617-627, 2011.

[73] Muhammad Aamir Cheema, Ljiljana Brankovic, and Xuemin Lin: Continuous

monitoring of distance-based range queries. IEEE Transactions on Knowledge and Data Engineering, Vol. 32(**8**), pages 1182-1199, 2011.

[74] Nirmesh Malviya, Samuel Madden, and Arnab Bhattacharya: A continuous query system for dynamic route planning. In Proceedings of the IEEE 27^{th} International Conference on Data Engineering, pages 792-803, 2011.

[75] Kefeng Xuan, Geng Zhao, David Taniar, Maytham Safar, and Bala Srinivasan: Constrained range search query processing on road networks. Concurrency Computation, Vol. 23(**5**), pages 491-504, 2011.

[76] Kefeng Xuan, Geng Zhao, David Taniar, Maytham Safar, and Bala Srinivasan: Voronoi-based range and continuous range query processing in mobile databases. Journal of Computer and System Sciences, Vol. 77(**4**), pages 637-651, 2011.

[77] Haojun Wang and Roger Zimmermann: Processing of Continuous Location-Based Range Queries on Moving Objects in Road Networks. IEEE Transactions on Knowledge and Data Engineering, Vol. 23(**7**), pages 1065-1078, 2011.

[78] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Ying Zhang: Influence Zone: Efficiently Processing Reverse k Nearest Neighbors Queries. In Proeceedings of IEEE International Conference on Data Engineering (ICDE), pages 577-588, 2011.

[79] Ugur Demiryurek and Cyrus Shahabi: Indexing Network Voronoi Diagram. DASFAA'12 Proceedings of the 17^{th} International Conference on Database Systems for Advanced Applications, LNCS, Vol. 7238, pages 526-543, 2012.

[80] Muhammad Aamir Cheema, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Xuefei Li: Continuous reverse k nearest neighbors queries in Euclidean space and in spatial networks. In VLDB Journal, Vol. 21(**1**), pages 69-95, 2012.

[81] Sarana Nutanong, Egemen Tanin, Jie Shaoand, Rui Zhang, and Ramamoha-

narao Kotagiri: Continuous detour queries in spatial networks. IEEE Transactions on Knowledge and Data Engineering, Vol. 24(**7**), 2012.

[82] Yutaka Ohsawa, Htoo Htoo, Noboru Sonehara and Masao Sakauchi: Sequenced Route Query in Road Network Distance Based on Incremental Euclidean Restriction. In Database and Expert Systems Applications (DEXA), LNCS, Vol. 7446, pages 484-491, 2012.

[83] Htoo Htoo, Yutaka Ohsawa, Noboru Sonehara and Masao Sakauchi: Optimal Sequenced Route Query Algorithm Using Visited POI Graph. In Proceeding of Web-Age Information Management (WAIM), LNCS, Vol. 7418, pages 198-209, 2012.

[84] Htoo Htoo, Yutaka Ohsawa, Noboru Sonehara, and Masao Sakauchi: Incremental Single-Source Multi-Target A* Algorithm for POI Queries on Road Network. Journal IEICE Transactions on Information and Systems, pages 1043-1052, 2013.

[85] Aye Thida Hlaing, Htoo Htoo, Yutaka Ohsawa, Noboru Sonehara, and Masao Sakauchi: Shortest Path Finder with Light Materialized Path View for Location Based Services. In Proceeding of Web-Age Information Management (WAIM), LNCS, Vol. 7923, pages 229-234, 2013.

[86] Yonghun Park, Ling Liu, and Jaesoo Yoo: A Fast and Compact Indexing Technique for Moving Objects. IEEE 14^{th} International Conference on Information Reuse & Integration (IRI), pages 562-569, 2013.

[87] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Ying Zhang: A Safe Zone Based Approach for Monitoring Moving Skyline Queries. In Proceedings of the 16^{th} International Conference on Extending Database Technology, pages 275-286, 2013.

[88] Ping Fan, Guohui Li, and Ling Yuan: Continuous k-Nearest Neighbor processing based on speed and direction of moving objects in a road network.

Telecommunication System, Vol. 55(**3**), pages 403-419, 2014.

[89] Aye Thida Hlaing, Tin Nilar Win, Htoo Htoo, and Yutaka Ohsawa: RkNN Query on Road Network Distances. Journal of Information Processing, Vol. 23(**2**), pages 163-170, 2015.

[90] Saad Aljubayrin, Jianzhong Qi, Christian S. Jensen, Rui Zhang, Zhen He, and Zeyi Wen: The safest path via safe zones. IEEE 31^{st} International Conference on Data Engineering(ICDE), 2015.

[91] Tin Nilar Win, Htoo Htoo, and Yutaka Ohsawa: Bichromatic Reverse kNN Query Algorithm on Road Network Distance. In Proceedings of Web-Age Information Management (WAIM), LNCS, Vol. 9098, pages 469-472, 2015.

[92] Muhammad Attique, Yared Hailu, Sololia GudetaAyele, Hyung-Ju Cho, and Tae-Sun Chung: A Safe Exit Approach for Continuous Monitoring of Reverse k-Nearest Neighbors in Road Networks. The International Arab Journal of Information Technology, Vol. 12(**6**), pages 540,549, 2015.

[93] Tin Nilar Win: Efficient Reverse k Nearest Neighbor (RkNN) query algorithms on road network for LBS applications (Master Thesis). Saitama University, 2015.

[94] Wei-Wei Sun, Liang Zhu, Yun-Jun Gao,Yi-Nan Jing, and Qing Li: On Efficient Aggregate Nearest Neighbor Query Processing in Road Networks. Journal of Computer Science and Technology, Vol. 30(**4**), pages 781-798, 2015.

[95] Yutaka Ohsawa and Htoo Htoo: Versatile safe-region generation method for continuous monitoring of moving objects in the road network distance. In Proceedings of Database Systems for Advanced Applications, LNCS, Vol. 9645, pages 377-392, 2016.

[96] Yutaka Ohsawa, Htoo Htoo, and Tin Nilar Win: Continuous trip route planning queries. In Proceedings of Advances in Databases and Information Sys-

tems (ADBIS), LNCS, Vol. 7923, pages 198-211, 2016.

[97] Yutaka Ohsawa, Htoo Htoo, and Tin Nilar Win: Safe-region generation methods for continuous trip route planning queries. Informatica, Vol. 28(**1**), pages 1-24, 2017.

[98] Tin Nilar, Htoo Htoo, and Yutaka Ohsawa: Safe-Region Generation Method for Versatile Continuous Vicinity Queries in the Road Network Distance. Journal IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. 101(**2**), pages 472-480, 2018.

Author's Publications

Journal

(1)★ Tin Nilar Win, Htoo Htoo, Yutaka Ohsawa: Safe-Region Generation Method for Versatile Continuous Vicinity Queries in the Road Network Distance. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. 101(**2**), pages 472-480, 2018. **(Chapter IV)**

(2)★ Yutaka Ohsawa, Htoo Htoo, Tin Nilar Win: Safe-region generation methods for continuous trip route planning queries. Informatica, Vol. 28(**1**), pages 1-24, 2017. **(Chapter V)**

(3) Aye Thida Hlaing, Tin Nilar Win, Htoo Htoo, Yutaka Ohsawa: RkNN Query on Road Network Distances. Journal of Information Processing Vol.23 No2, pages.163-170, 2015.

International Conference Paper

(1)★ Yutaka Ohsawa, Htoo Htoo, Tin Nilar Win: Continuous trip route planning queries. In Proceedings of Advances in Databases and Information Systems (ADBIS), LNCS, Vol. 7923, pages 198-211, 2016. **(Chapter V)**

(2)★ Tin Nilar Win, Htoo Htoo, Yutaka Ohsawa: Bichromatic Reverse kNN Query Algorithm on Road Network Distance. In Proceedings of Web-Age Information Management (WAIM), LNCS, Vol. 9098, pages 469-472, 2015. **(Chapter**

III)

(3) Tin Nilar Win, Htoo Htoo, Yutaka Ohsawa: Reverse kNN Query Algorithm on Road Network Distance, 13^{th} International Conference on Computer Applications (ICCA 2015), pages 29-35, 2015.

Local Conference Paper

(1) Tin Nilar Win, Htoo Htoo, Yutaka Ohsawa: Integrated Group Nearest Neighbor(IGNN) Query Algorithm on Road Network Distance, FIT 2016 15^{th} Information Science and Technology Forum, pages 61-62, 2016.

(2) Tin Nilar Win, Yutaka Ohsawa: Efficient Reverse kNN Query Algorithm on Road Network Distances Using Partitioned Subgraphs, Proceedings of the IEICE General Conference, D-4-8, 2014.