MONASH University

Finding Best Paths in Spatio-Textual Queries

Anasthasia Agnes Haryanto

A thesis submitted for the degree of Doctor of Philosophy at

Monash Univesity in 2019

Clayton School of Information Technology

Copyright Notice

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature:

Print Name: Anasthasia Agnes Haryanto

Date: February 25, 2019

Publications during enrolment

Publications arising from this thesis include:

1. Anasthasia Agnes Haryanto, Md. Saiful Islam, David Taniar, Muhammad Aamir Cheema. **IG-Tree: an efficient spatial keyword index for planning best path queries on road networks**. In *World Wide Web (WWW) Journal, 2018.*

2. Anasthasia Agnes Haryanto, David Taniar, Md. Saiful Islam, Muhammad Aamir Cheema. ***w*BestPath: A Spatio-Textual Route Planning Query on Weighted Regions**. Submitted for publication in *Journal of Ambient Intelligence and Humanized Computing (AIHC).*

Thesis including published works declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes 1 original paper published in peer reviewed journal and 1 submitted publication. The core theme of the thesis is spatio-textual route planning queries. The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the Faculty of Information Technology, Monash University, under the supervision of Associate Professor David Taniar, Associate Professor Muhammad Aamir Cheema, and Dr. Saiful Islam.

The inclusion of co-authors reflects the fact that the work came from active collaboration between researchers and acknowledges input into team-based research.

In the case of Chapter 3 and Chapter 4 my contribution to the work involved the following:

Thesis Chapter	**Publication Title**	**Status** *(published, in press, accepted or returned for revision, submitted)*	**Nature and % of student contribution**	**Co-author name(s) Nature and % of Co-author's contribution***	**Co-author(s), Monash student Y/N***
3	*IG-Tree: an efficient spatial keyword index for planning best path queries on road networks*	*Published*	*75%. Problem formulation, technique, algorithm, experiment*	*1. Saiful Islam, input into technique and algorithm 10%* *2. David Taniar, input into problem formulation and algorithm 10%* *3. Muhammmad Aamir Cheema, input into paper 5%*	*No* *No* *No*
4	*wBestPath: a spatio-textual route planning query on weighted regions*	*Submitted*	*80%. Problem formulation, technique, algorithm, experiment*	*1. David Taniar, input into problem formulation and technique 10%* *2. Saiful Islam, input into problem formulation 5%* *3. Muhammmad Aamir Cheema, input into paper 5%*	*No* *No* *No*

I have renumbered sections of submitted or published papers in order to generate a consistent presentation within the thesis.

Student signature: Date: February 25, 2019

The undersigned hereby certify that the above declaration correctly reflects the nature and extent of the students and co-authors contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

Main Supervisor signature: Date: February 25, 2019

Acknowledgments

First and foremost, I would like to thank God for all His love, blessings, and guidance for me in my life. If it was not because of Him, I would not even have pursued PhD study and put this thesis together as a mark of completion of this course.

I am very grateful that He had placed me within the guidance of three of my supervisors which I would like to heartfully express my gratitude for. They have helped me as great mentors.

Associate Professor David Taniar, my main supervisor, has consistently guided me with enormous patience and provided me with encouragement since my Honours year. His works and wisdom motivated me to do PhD and pursue a career in this field of spatial databases. He directed me with my PhD project since the first day until the submission date of my thesis. He constantly gave constructive feedbacks for all my work and helped me encounter exciting opportunities, such as attending and organising conferences as well as teaching. I would not be able to experience all these without him.

Associate Professor Muhammad Aamir Cheema, my co-supervisor, who has directed me especially at the beginning of my PhD to find the topic that I should be focusing on. Throughout my PhD he has also been giving me helpful academic and moral support.

Dr. Saiful Islam, my external supervisor, who has provided me with valuable advices for my works during my PhD candidature. Despite living in a different state, he willed to spare some time for meetings whenever he comes to Melbourne and pushed me to strive for the better.

I am also very grateful for my colleagues in the spatial databases research group: Arif Hidayat, Ammar Sohail, Chaluka Salgado, Lingxiao Li, Nasser Allheeib, Tenindra Abeywickrama, Utari Wijayanti, Xinyu Li and in particular Zhou Shao, who was my classmate since Honours year and officemate for the course of my PhD. My PhD journey would not be memorable without all of their jokes and support.

Also, a heartfelt thanks to the staff in the Faculty of Information Technology, especially for Helen Cridland, Julie Holden, and Danette Deriane. Danette has always constantly provided me with advices and assistance on my PhD candidature. Helen and Julie, who always supported me with both academic and extra-curricular activities. All their kindness and support for me have helped me throughout my PhD.

I would moreover like to thank Monash Graduate Research for supporting me financially through the Monash International Postgraduate Research Scholarship (MIPRS) and Monash Graduate Scholarship (MGS) throughout the course of my PhD.

A special thank you to everyone in Catholics on Campus, especially to Barbara Shea, Fr. Chris Dowd and Fr. Robert Krishna for their constant prayers and support. They were always available in the Chaplains office whenever I need someone to talk to.

Last but not least, I would like to thank my family who has been supporting, praying and loving me unconditionally. They are always there for me in the midst of my difficulties, ready to brighten my day, constantly motivate me in times of need, and taking care of me in all aspects of my life. Words cannot express how thankful I am for having them in my life.

Again, I would like to thank all the people that has been part of my life throughout my PhD. I cannot list all of you one by one but each one of you has contributed in ways that makes me who I am today.

Anasthasia Agnes Haryanto

Monash University

February 2019

Finding Best Paths in Spatio-Textual Queries

Anasthasia Agnes Haryanto
aahar3@student.monash.edu
Monash University, 2019

Supervisor: Associate Professor David Taniar
david.taniar@monash.edu
Associate Professor Muhammad Aamir Cheema
aamir.cheema@monash.edu
Dr. Saiful Islam
mdsaiful.islam@griffith.edu.au

Abstract

Spatial Databases have captured the attention of today's society. People nowadays rely heavily on various applications that uses spatial data to help them on their daily activities, such as navigation. Due to the popularity of Spatial Databases, many search engine providers have started to expand their text searching capability to include geographical information. This causes many new queries on spatial objects affiliated with textual information, known as the Spatial Keyword Queries, to take significant research interest in the past years. Unfortunately, most of existing works on Spatial Keyword Queries only focus on objects retrieval. There is very limited work on route planning queries. Therefore in this study, we propose the **Best Path Query**, which focuses on finding the most optimal route from two different spatial locations that visits or avoids the objects specified by the textual data given by the user.

Computation for finding the Best Path will differ depending on the type of the environment. In this thesis, we will mainly focus on computing the Best Path on two different types of environments: *Road Networks* and *Weighted Regions*. In the Road Networks scenario, we show that the Best Path Query is an *NP-Hard* problem. We propose an efficient indexing technique, namely *IG-Tree*, and three different algorithms with different trade-offs to process the Best Path Queries on Road Networks. Our extensive experimental studies on real road network datasets demonstrate the efficiency and accuracy of our proposed approaches.

Furthermore, a lot of research has been done on Spatial Keyword Queries on Euclidean space and on road networks. However, the study on a more complex environment model

like weighted regions has never been done yet. Therefore in our research, we conduct a study on the Best Path query on Weighted Regions, which is called the wBestPath. Despite the limitation of current works in Spatial Queries on Weighted Regions, we successfully designed a new indexing scheme called the *wIG-Tree* that incorporates Weighted Regions information together with spatio-textual objects. We also develop two algorithms with different trade-offs to answer the wBestPath query while utilising the proposed index. At the end, we report our experimental evaluation and show the efficiency of our proposed solutions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

In this modern day and age, people can share information in a blink of an eye. The amount of information being shared these days are massive and it can be in various forms like words, images, sounds, and even locations. With the advancement of technologies, all this information or data can be easily shared and processed to help us in our daily routine. Even numerous organisations measure their success through their capability of obtaining accurate and timely data of their operations, managing the data proficiently, and to use the data for analysis and guidance purpose [4]. We can see that data is an asset in our life. However, with the amount of data that is constantly being produced every day, it gets even harder to process all the information without managing it. Data that is initially an asset can become a distraction and a liability since the cost of obtaining and maintaining the data can be excessive compared to the value derived from the data itself [4]. Thus to tackle this problem, there have been attempts in managing the tremendous amount of data through database systems.

Database itself is a vibrant discipline as data can come in different forms like words or even geographical entities. One of the rapidly growing area in database systems is the Spatial Databases, in which it mostly deals with geographical spaces. Güting *et al.* [5] describe Spatial Databases as a database system that offers spatial objects in its data model and query language, and it supports the implementation of spatial data, including the indexing for spatial data and efficient algorithm for spatial join. The spatial objects in this case are represented by points, lines and regions. For instance in Figure 1.1, the

points represent the location of facilities or places of interest, the lines represent the roads, while the regions represent the extension of spatial objects, such as a large building, lake, park, or even a country. As we can see here, geographical or spatial information can be very complex and tremendous, in which it is also needed to be managed through database systems in order to keep it as an asset.

There are a lot of applications of spatial databases, such as Geographic Information System (GIS), Computer Aided Design (CAD), robotics, and even on biomedical research area. The accessibility of these applications is also attainable not only on desktop computers, but it also on mobile environments [6, 7, 8, 9, 10, 11, 12]. StatisticBrain shows that about 81% of mobile users use their device for Maps and directions [13]. This expresses how Spatial Databases have taken a big interest in today's society and have been assisting us in our daily activities.

With the popularity of Spatial Databases, the search engine providers, such as Google and Yahoo, have also broadened their text searching capability to provide geographical information [14, 15]. The GlobalWebIndex showed that Google Maps is the most used app as it is used by 54% of the general smartphone users [16]. Thus many new queries on spatial objects affiliated with textual information have been studied in the past years [17]. We call this kind of query as Spatial Keywords Query (or Spatio-Textual Query) as it combines both Spatial Databases and Information Retrieval (IR) System.

In the past, the conventional search engines can only process simple user queries, such as finding a place of interest based on a certain keyword [18]. However the high demand in spatial data compels the search engines to have the capability of processing more than such simple queries. Users may also want to find the nearest objects in the surrounding or find the direction of some locations through specific keywords. A number of researches have been done in order to improve the geographical search engines. Yet there are still many challenges appear in order to combine and process both the textual information and the spatial data. Therefore in this research, we investigate how the two different information, namely the textual and spatial information, can be processed together in order to produce a useful information for the users to fulfill their daily tasks.

This chapter is organised as follow. Section 1.2 explains the motivation behind our research. Section 1.3 describes the objectives of this research. We discuss our contributions in Section 1.4. Then in Section 1.5, the organisation of this thesis is presented.

1.2 Motivation

Nowadays, each spatial object contains one or more meaningful keywords as to represent the object's entities [14, 1]. The keywords may contain country name, city name, address, references to landmark, or even type of road [15]. Through the existence of spatial data and keywords, the Spatial Keyword Queries become varied. Some of the commonly used queries are the *Top-k kNN Query, Boolean kNN Query,* and *Boolean Range Query* [19]. All of these queries require the user to give a spatial location (normally the current user location) and textual data in the form of keywords as the input. The output of these queries are spatial objects that are nearest to the user's location and contain the keywords given. The principle parameter used to identify the nearest objects is based on the shortest path distance. Thus we can see that path construction takes an important role in answering these queries.

So in relation to path construction, a number of problems can be classified together into a Taxonomy of Path Queries as follows.

1. Shortest Path. Shortest Path Problem has been widely studied for the past decade, which is why most of the current Spatial Keyword Queries concentrate more on the traditional Shortest Path problem to determine the point-to-point solution. In Shortest Path problem, given a source and destination locations, we try to establish a path with the minimum sum of distance between these two locations. For instance, a user gives an input to the search engine "find the path to Melbourne's GPO from Woolworths QV". So the search engine has to find the location of Woolworths QV and Melbourne's GPO and then return the most efficient path to the user. Figure 1.1 shows the result of the shortest path between these two locations.

There are various solutions offered to find the shortest paths. From doing some pre-computation indexes to direct query processing without any indexes. Using pre-computation surely increases the query processing time in $O(1)$ time. However the main drawback is when the pre-computation is done, where it typically can occupy up to $O(n^2)$ space. While for no pre-computation approaches, such as the Dijkstra algorithm, mostly run in $O(n \log n)$ time.

While the existing solutions to the shortest path problems are useful, they are not always sufficient for our needs. In real life scenario, we often want to plan our trip before

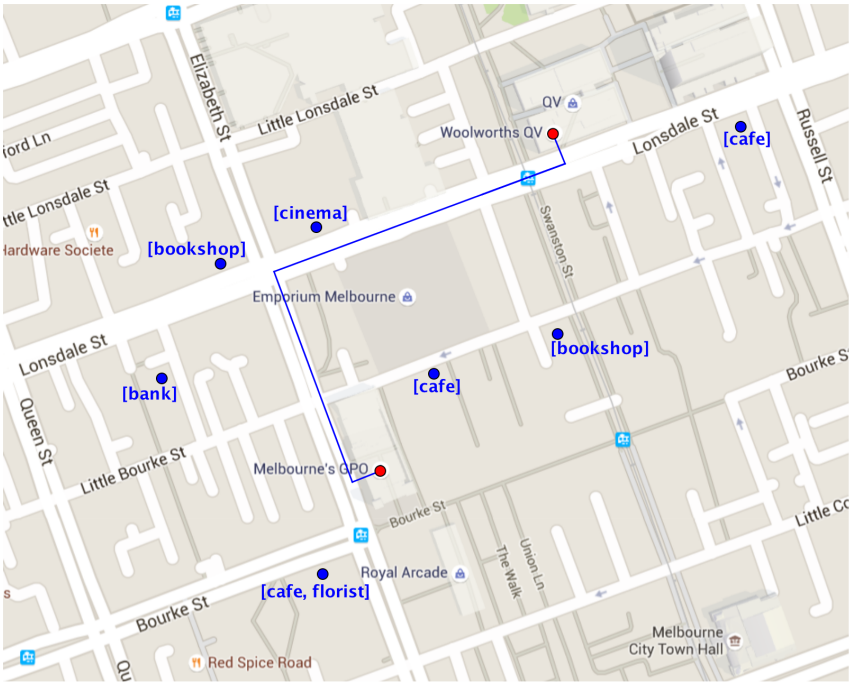


Figure 1.1: Shortest Path from Woolworths QV to Melbourne's GPO

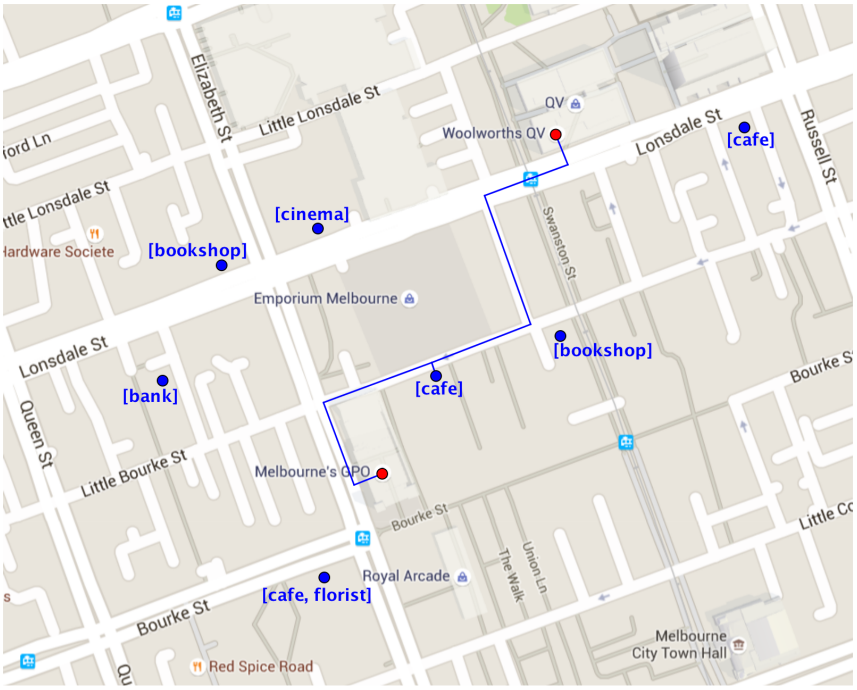


Figure 1.2: Shortest Path from Woolworths QV to Melbourne's GPO while dropping by a cafe

going to a certain place. When we are heading to our destination, we also want to pass through several other places. For example, we want to go from Woolworths QV to Melbourne's GPO but on our way we also want to visit a cafe (Figure 1.2). The solutions that simple shortest path problem offers will not work since they only find a path with

the minimum total distance, there is a chance that it will not pass any cafe. This kind of query is known as a route planning query.

2. Travelling Salesman Problem. One of the most famous route planning queries is the Travelling Salesman Problem (TSP). In TSP, suppose that we have a list of cities and a distance between every pair of the cities, the query is to find the most optimal path that visits each city exactly once and returns to the original location at the end of the trip. Despite its complexity to be an *NP-hard* problem, this query has been intensely studied in computer science. A lot of algorithms have been developed to optimise this problem. From offering exact algorithms to heuristic algorithms. Figure 1.3 shows an example of TSP application.

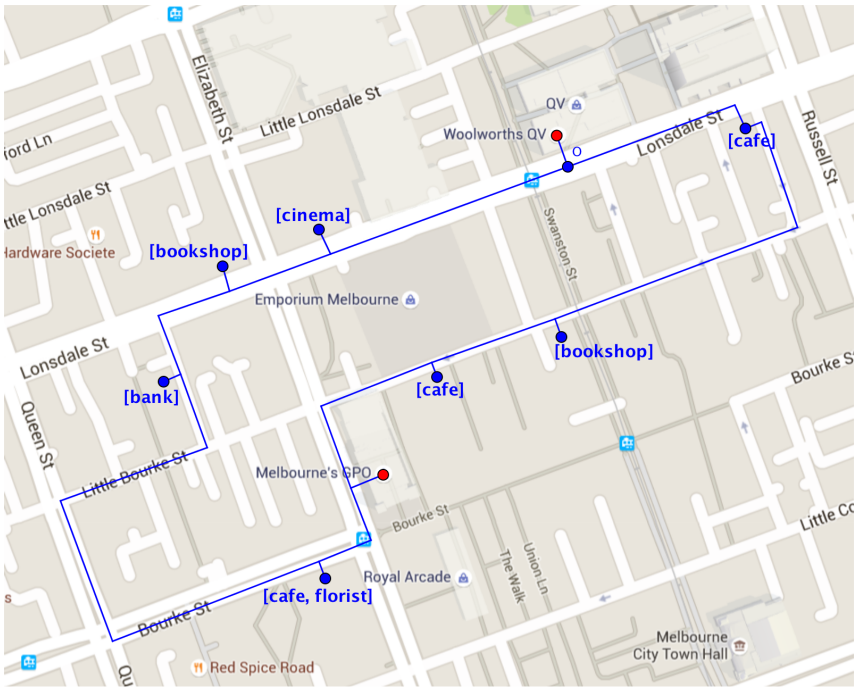


Figure 1.3: Travelling Salesman Problem (TSP)

3. Sequenced Path Planning. Sometimes we want to plan our trip based on our schedule. For example a user would like to go buy a book at 8 a.m. at the nearest bookshop, and then withdraw some money at a bank at 9 a.m. before meeting his friend to watch a movie at the cinema at 11 a.m.. Based on this situation, the route has to be in sequence where we have to go to a bookshop first, then to a bank, and at the end we should go to the cinema. Figure 1.4 shows a planned path in sequential order to answer

this particular query. This particular problem is called sequenced path planning because there is an order in the route that we have to follow.

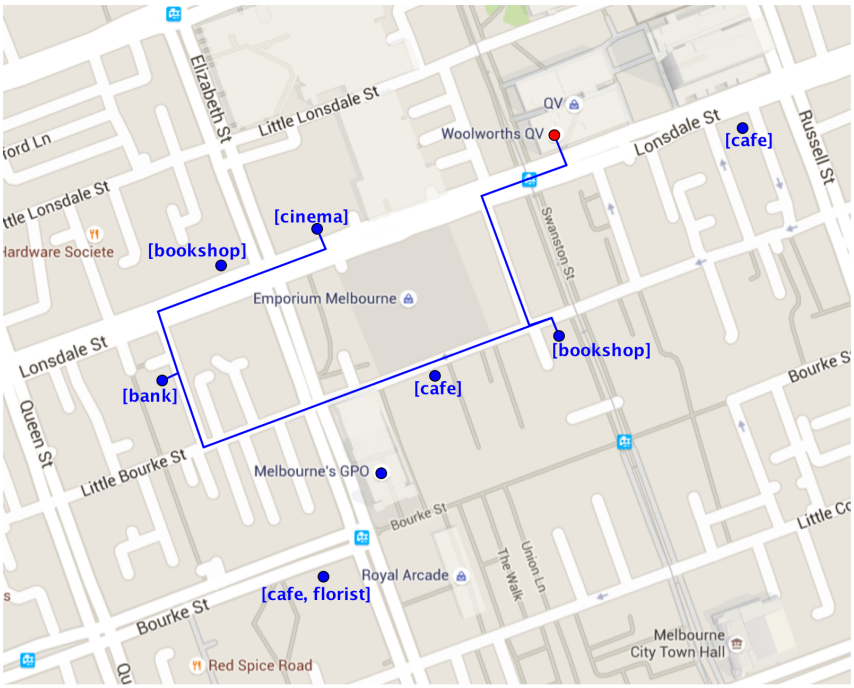


Figure 1.4: Sequenced route planning from Woolworths QV to a bookshop, then bank, and then cinema

4. Non-sequential Path Planning. It is often that we plan our trip without any order to where to go first. We, however, want to optimise our time or distance taken for the whole trip. For instance suppose that a tourist who is currently at Woolworths QV wants to visit a bookshop, a bank, and a cinema. Looking at Figure 1.5, it offers the most optimum path to visit all of the places that the tourist specified. Even though there is a bookshop that is closer to the user's current location, it does not guarantee an optimal total distance when being grouped together with the other criteria. We can see that the query processing time will increase significantly when the number of objects increase. This kind of query is in fact another *NP-hard* problem. It can be proven as *NP* problem by reduction to TSP.

5. Shortest Path with Obstacle. In the previous path queries, the queries only consider objects/keywords that they want to visit. In real life, there are times when we actually have to evade certain situations like a road block. For instance, given a user query "find the shortest path from Woolworths QV to Melbourne's GPO and avoid any road

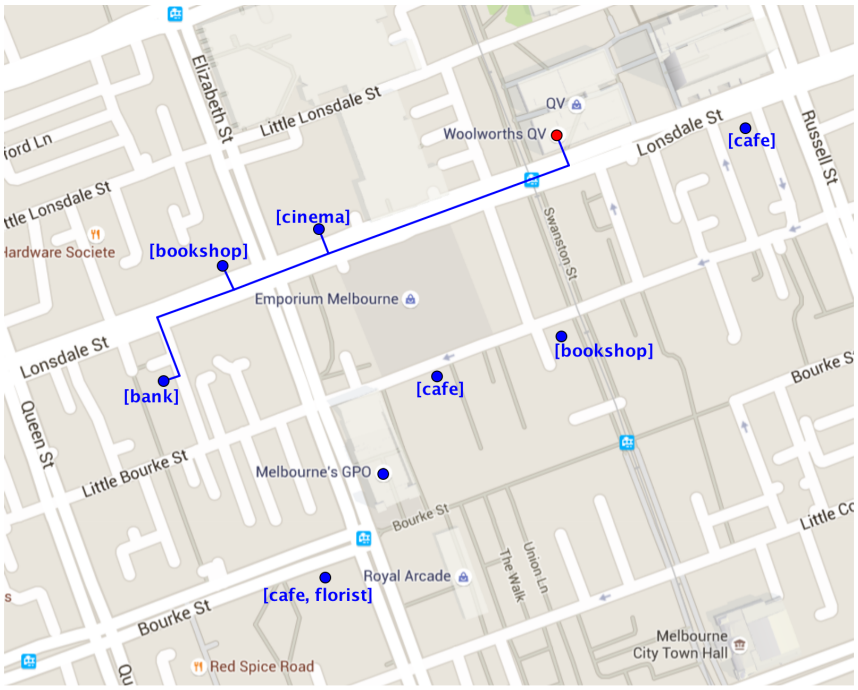


Figure 1.5: Route planning from Woolworths QV to bookshop, bank, and cinema without any sequence

block". Figure 1.6 shows the output of the query. Previously in Figure 1.1, the shortest path is through the road where the road block is located. But since we want to avoid it, then the path is changed. So the object/keyword that we want to avoid can be considered as obstacle.

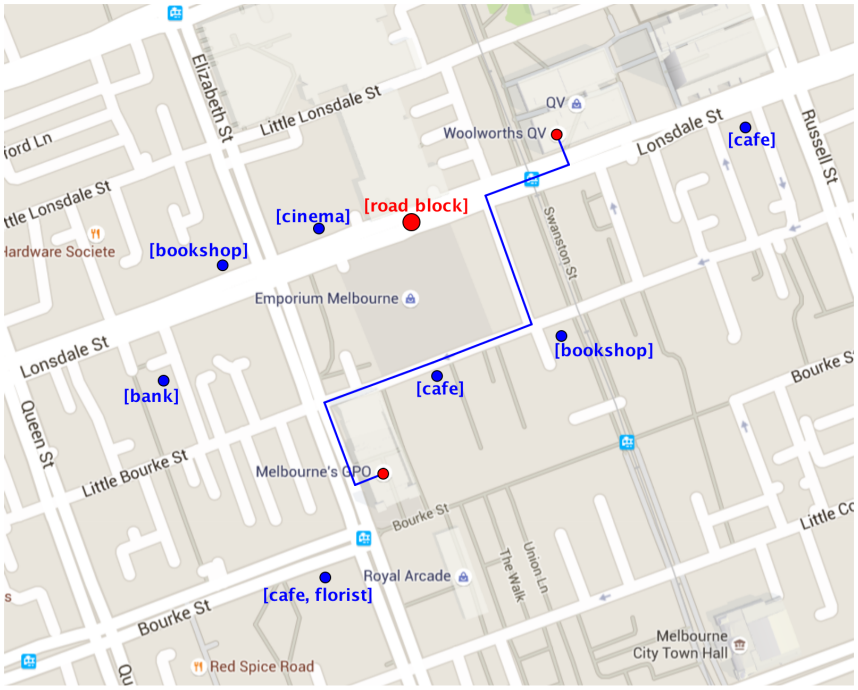


Figure 1.6: Shortest path from Woolworths QV to Melbourne's GPO while avoiding a road block

The time complexity of this kind of problem is actually the same as normal shortest path, which is $O(n \log n)$ without any pre-computation. This is due to the fact that the algorithm used in the normal shortest path will also work on this kind of query. When Dijkstra algorithm is implemented to this case, the path where the obstacle lies will be 'blocked'. Subsequently, we can continue the Dijkstra expansion method until the path is found. The only worst-case scenario when we have a lot of obstacles which may block most of the paths.

6. Route Planning with Obstacle. In this last classification of path queries, the query is a combination of route planning with obstacle. Supposed that we want to find the path from Woolworths QV to a bookshop, bank, and cinema while avoiding a road block. Based on this query, we know that when we construct the path, it has to pass at least one bookshop, bank, and cinema. The obstacle here is the road block. Figure 1.7 shows the output of this particular query. If we compare to Figure 1.5, the path is totally changed because of the obstacle. The final path becomes longer and the order of the places we are going to visit changed. Hence this query's complexity is also *NP-hard* as we can reduce TSP to this query.

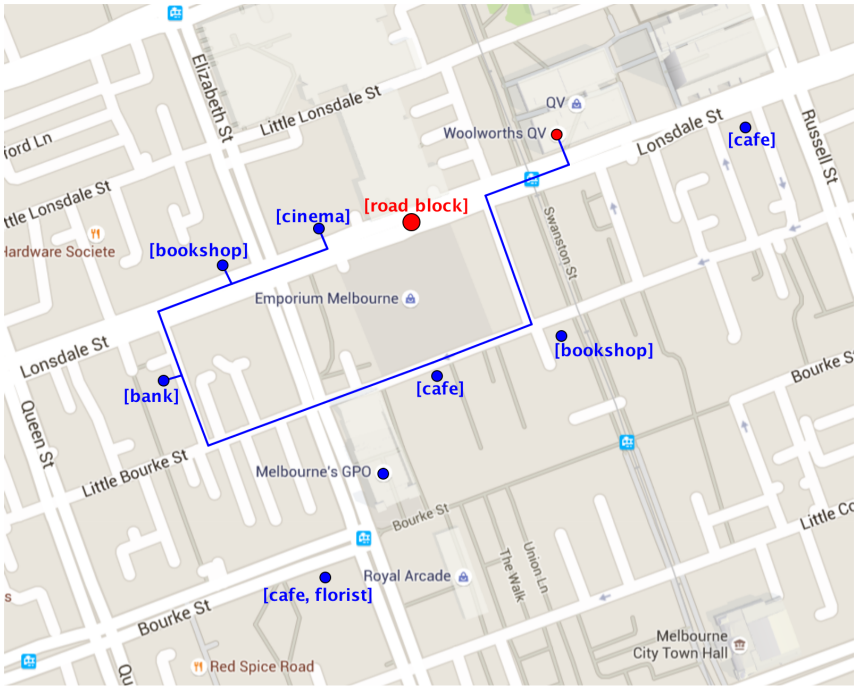


Figure 1.7: Route planning from Woolworths QV to bookshop, bank, and cinema while avoiding a road block

To our knowledge, no one has done any work on this kind of query. Most of the existing works are more focused on the first five classifications. This leaves a gap for us to develop a solution to this kind of query. Thus this particular query is the main focus of our research.

Best Path Query. In this study we introduce a new variant of Spatial Keywords Query, which is the **Best Path Query**. Similar to the Taxonomy number 6, given a user with his/her source location, this user wants to go to his/her destination while stopping by or avoiding several facilities denoted by keywords. In this kind of query, we want to plan an optimal trip that will visit each facility or avoiding them based on the keywords given by the user before arriving to the destination. The solutions to the traditional shortest path problem cannot work on Best Path Query as it only considers two spatial locations and find the shortest paths between these two locations. The traditional shortest path problem also does not take into consideration any trip planning or textual information processing.

In Best Path Query, we are dealing with both spatial and textual data. The **spatial data** will be the *source* and *destination* locations that are given by the user. The **textual data** is the *keywords which are attached to the facilities* that we would like to pass on our way to the destination. Each spatial object has one or more keywords attached. Keywords can be the name of a suburb, road name, type of road, or name of attraction. Based on the user input, the keyword itself can be considered in *positive, negative, or neutral situations*. As in the example before, the user gave a negative keyword, which is not going to pass highway. The positive keyword here is the facility/point that he wants to pass by. And for the neutral keywords are the remainder keywords that are attached to the objects but not chosen by the user. Therefore the definition of Best Path Query is as follow:

Definition 1. *Given source location s_l, destination location d_l, and a set of keywords $K = \{k_1, k_2, ..., k_n\}$, k_n can be positive (k^+) or negative (k^-). Find the Best Path $BP(s_l, d_l, K)$ from s_l to d_l that passes through all k^+ and avoid k^-.*

The main challenge here is in the insufficiency of current solution to trip planning in Spatial Keywords area. In fact, there is no solution to Best Path. Another challenge is that current works do not take into consideration of negative keywords. Having negative keywords might increase the complexity of the problem as we have to make sure that

we can avoid certain paths. There are definitely some cases where the results cannot be retrieved. The common reason is the low textual relevancy at the surrounding area, specifically the area between source and destination locations, which making us to have to avoid all of the paths to the destination. Also depending on the environment, the computation for finding the Best Path will differ. In this study we mainly focus on two types of environment, **(1) the Best Path on Road Networks** and **(2) the Best Path on Weighted Regions**.

In the following subsections, we are going to discuss the limitations of existing works that motivates our study.

1.2.1 Best Path on Road Networks

In the previous studies, the focus is always on the shortest path. However, often in our life we want to go to a certain place but while on our way to our destination we want to pass through several places. A lot of times we want to find a route to our preferred destination that passes certain objects of interest instead of going straight to the destination. And we often would like to avoid some objects that can interfere our activities.

As an example, a user wants to go to his workplace from home but he wants to drop by the gas station first and also wish to avoid any traffic accident around his way. In this query, the user gives the source and destination locations and several keywords, which are gas station and avoid accident. So we need to find the optimum path for the user that satisfies his preferred keywords condition. Therefore we call this kind of query as the **Best Path Query**.

Using the example mentioned before, supposed that user gives his *source* and *destination* location as can be seen in Figure 1.8 and the *keywords*={*gas station, not accident*}. As we have a negative keyword here, which is the *not accident*, we have to avoid passing through any of this point on the road. Thus, the result of this query will be as in Figure 1.10. It is indeed further than the shortest path but it is as what the user wishes to be.

The best path is different with the general shortest path. In the best path, we want to find the path with objects of interest along our way to the destination. The objects of interest are determined by the keywords. The best path itself might be longer than the shortest path, but it can also be the same as the shortest path. Even though the best

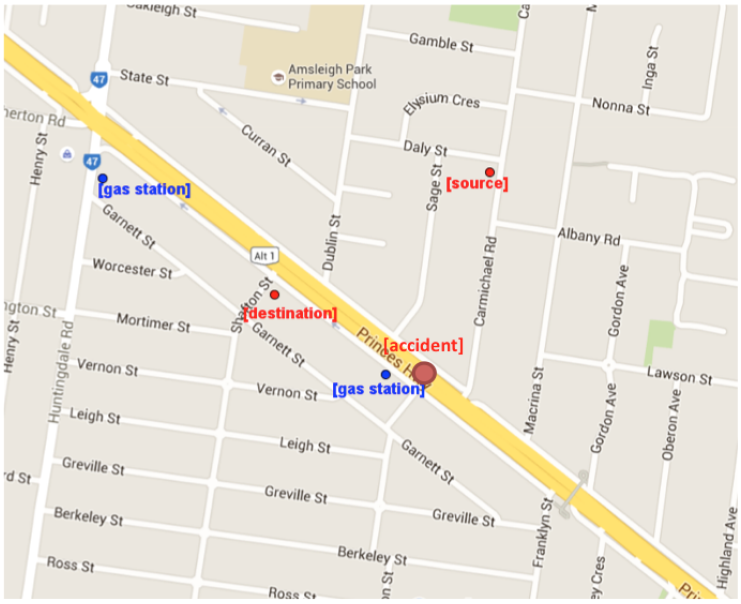


Figure 1.8: Example of user query locations={source, destination} and keywords={gas station, not highway}

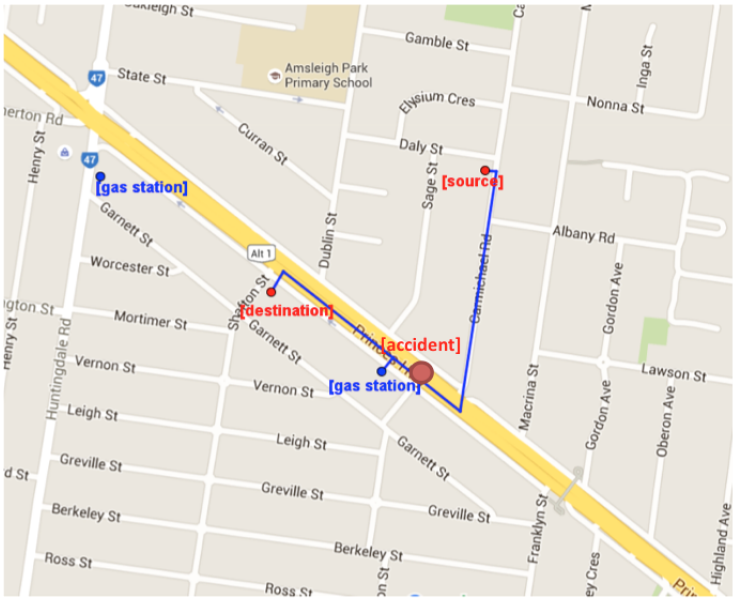


Figure 1.9: Supposedly for shortest path with locations={source, destination} and keyword={gas station}

path can be farther than the shortest path, the best path provides the object of interest that user requested.

When we are working with road networks, the road networks are usually represented as a graph. The edges typically represent the road segments while the vertices represent the road intersections. So the path generation is always limited only to the edges to adjacent vertices. This increases the complexity of our query computation when we are working specifically in this environment. The complexity is also intensified with the types of the

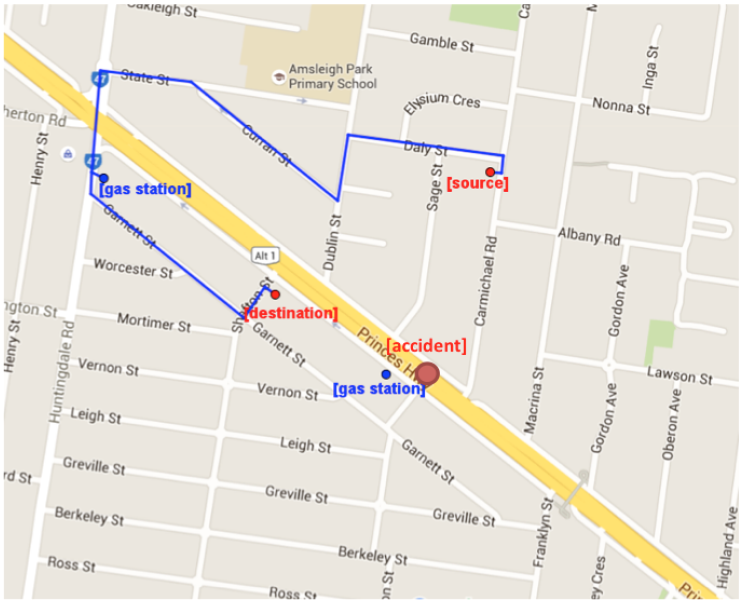


Figure 1.10: Best Path for query locations={source, destination} and keywords={gas station, not accident}

keywords we have. When the keywords are negative, a lot of the paths will be blocked as we have to avoid them. We always have to make sure that we plan the route optimally despite these complexities.

Limitations of Existing Works. There are a number of similar route planning queries as Best Path. The most well known query is the On Trip Planning Route Queries (TPQ), which retrieves the best trip from two different locations that passes at least one point from each of the chosen categories [20]. Another popular route planning query is Optimal Sequenced Route Queries (OSR), a spatial query that finds the minimum route distance from a source location and passing through a set of sequenced categories [21]. However these queries do not consider any keywords processing. Best Path Query focus on Spatio-Textual field, which in this case it needs to process the textual part of the objects. Another difference is that all of these existing works do not take into consideration any negative situation. Everything in the chosen categories in TPQ and OSR must be visited, while in Best Path there are some categories that we have to avoid.

In Spatio-Textual area itself, there is only one study on route planning, which is the Keyword-aware Optimal Route Search (KOR) [22]. It is a query that finds an optimal route that covers a set of user given keywords with a specific budget constraints and objective score [22]. In conjunction to Best Path Query, KOR is different as the query

requires budget constraint for processing. KOR also does not take into consideration negative keywords as what Best Path does.

Most of the route planning problems are regarded as a generalisation of Traveling Salesman Problem (TSP) problem [23, 24]. They are NP-hard problems. The solutions offered for these queries are in polynomial time approximation algorithms. Even so, the solutions offered are generally involving no pre-processing. This causes more computation particularly since the computation requires the processing of both spatial and textual relevancy. So having on-the-go solution does not always guarantee performance efficiency, especially on Spatio-Textual field. Therefore in this research we are attempting to provide a better solution to this problem by offering a pre-processing index that incorporate both Keywords and Spatial Road Networks.

1.2.2 Best Path on Weighted Regions

A lot of Spatial Databases works are concentrating on Euclidean space and Road Networks. However in reality, the Euclidean distance-based approach is not always accurate [25, 26]. The earth's surface is not constantly a flat concrete surface. Because of this, the Euclidean distance does not offer an optimal solution in real life. The same thing also applies to road networks. Real road networks only offer the distance of the most common passable roads. Unfortunately the distance is being generalised without considering the impact of the earth terrains. For example the amount of effort to pass a flat surface covered in grass is different from a flat concrete surface. The effort of passing through a slope is totally different from a flat surface. So not all roads are easily accessible. Figure 1.11 shows an example of terrain map, in which the different types of road surface can be seen clearly. The accessibility of roads is particularly crucial in concern of various types of vehicles or even for wheelchair users who has to carry themselves through roads with different types of surface and sloped angles. Due to these reasons, we conduct our study to investigate the impact of the **Best Path Query on Weighted Regions** and propose the best solution for trip planning on a complex environment like this.

Limitations of Existing Works. The main weakness of the existing works is on the fact that they are limited to Euclidean or Road Network environment in which these two space models do not always reflect the real resemblance of earth surfaces. In this study,

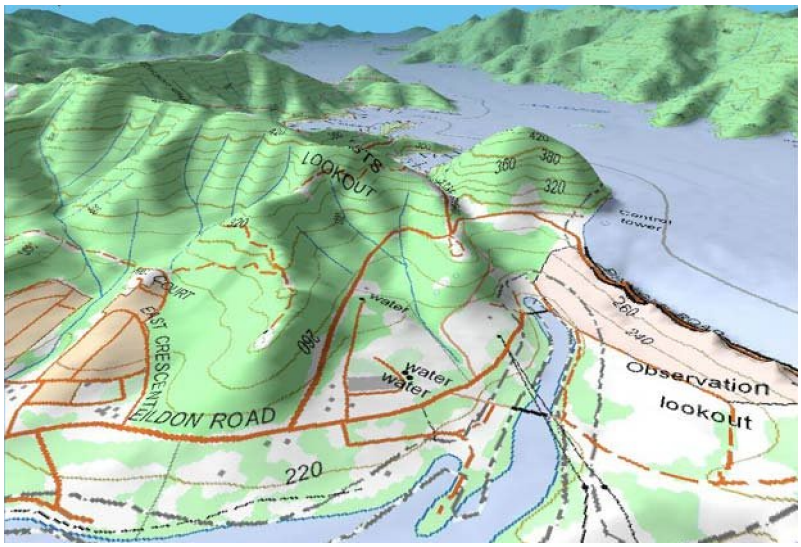


Figure 1.11: Example of terrain map

we are going to enhance the capability of spatial databases to be able to process queries on varied surfaces through the Weighted Regions space model. Some works on finding shortest path on varied surfaces have been significantly studied lately, which is known as the weighted regions problem [27, 28]. So each surface will have its own weight that will affect the effort to pass each of those regions. However the existing studies do not cover any route planning, especially in the context of Spatio-Textual area. Thus this is a great opportunity for us to propose a novel technique on this particular problem.

1.3 Research Objectives

The **primary aim** of this research is to propose efficient techniques to solve the Best Path Query and issues related to finding the Best Path. In order to achieve our primary aim, we propose **two specific objectives** as follow:

1. **Objective 1:** Propose efficient indexing and query processing techniques to solve Spatio-Textual Queries on Road Networks, particularly the Best Path Query.

   (a) **Objective 1a:** Formalise the Best Path Query on Road Networks.

   (b) **Objective 1b:** Propose an efficient indexing technique that incorporate both Spatio-Textual objects and Road Networks.

   (c) **Objective 1c:** Investigate the complexity to solve the Best Path Query with and without any pre-processing.

(d) **Objective 1d:** Propose an efficient query processing algorithm to solve the Best Path Query while utilising the proposed index.

2. **Objective 2:** Propose efficient indexing and query processing techniques to solve the Spatio-Textual Queries on Weighted Terrains, particularly the Best Path Query.

   (a) **Objective 2a:** Formalise the Best Path problem on Weighted Terrains.

   (b) **Objective 2b:** Propose an efficient indexing method that incorporate both Spatio-Textual objects and Weighted Terrains.

   (c) **Objective 2c:** Propose an efficient query processing algorithm to solve the Best Path Query while utilising the proposed index.

1.4 Contributions

In this section, we briefly discuss our contributions in this thesis.

1.4.1 Best Path on Road Networks

We formally define Best Path problem on road networks and prove that this is an NP-hard problem. We develop a novel indexing scheme, called *IG-Tree*, for planning Best Path queries in road networks. We present three approximate algorithms with different trade-offs for searching Best Paths on road networks. Those algorithms are the Optimal Distance Approximation Search, Ancestor Priority Search, and the Euclidean-based Approximation solution. We also demonstrate the effectiveness and efficiency of our algorithms through comprehensive experiments on real datasets.

This work was published in *World Wide Web (WWW) Journal* 2018.

1.4.2 Best Path on Weighted Regions

We formalise the wBestPath, a study on route planning on weighted regions. To the best of our knowledge, we are the first to attempt on this kind of problem. We propose *wIG-Tree*, an indexing technique that incorporates spatio-textual attributes that can be used to solve Best Path problem on weighted regions. We propose two approximation algorithms with different trade-offs that utilise the *wIG-Tree* to process the wBestPath queries, namely the Minimum Distance Approximation (MDA) Algorithm and Minimum

Path Approximation (MPA) Algorithm. We conduct a comprehensive set of experiments to demonstrate the effectiveness and efficiency of our algorithms.

This work has been submitted for publication in *Journal of Ambient Intelligence and Humanized Computing (AIHC)* and currently under review.

1.5 Thesis Organisation

The structure of this thesis is presented as following:

- Chapter 2 presents a review on the existing related works that motivate us to conduct a research on Best Path Query. In this chapter, we explore varieties of spatial-only queries, spatial keyword queries, and route planning queries on different environments like Euclidean space, Road Networks, and the Weighted Regions.

- Chapter 3 describes our work on Best Path on Road Networks, where we propose a new indexing scheme for Spatial Keywords Queries on Road Networks and some solutions to answer the Best Path problem on this particular space model.

- Chapter 4 presents our research on Best Path on Weighted Regions. This chapter also describes our new indexing technique for Spatial Keywords Queries on Weighted Regions and some solutions to solve the Best Path Query on Weighted Regions.

- Chapter 5 concludes our research and describes several possible directions for future work.

Chapter 2

Literature Review

2.1 Overview

This chapter provides some background information on spatial queries and reviews various related works to our study. In Section 2.2, we discuss some *Spatial Queries* that are categorised based on the working space models and various techniques to solve these queries. Section 2.3 presents an overview of different types of *Spatial Keywords Queries* and current indexing techniques in this area. Section 2.4 explores several existing *Route Planning Queries* and some variants to this particular type of query. We also review the previous works on *Weighted Regions* in Section 2.5. In Section 2.6, we summarize and provide some discussions on the existing works that we have reviewed in the previous sections.

2.2 Spatial Only Queries

Various Spatial Queries have been explored in the past decade and many researches have offered solutions to these queries. The study on spatial queries itself can be categorised based on the space model used, as each space model can affect the way we solve a certain spatial query. In this section, we discuss several different spatial queries based on the space they operate, specifically on *Euclidean space* and *Road Networks*. Section 2.2.1 explores various spatial queries on Euclidean space that are commonly used and also several indexing techniques that have been proposed to solve the queries in Euclidean space. Meanwhile, Section 2.2.2 explores the other space model, which is the Road Networks.

2.2.1 Spatial Queries on Euclidean Space

One of the space models that is used in spatial databases is the Euclidean space. In Euclidean space, the distance between two objects in the space can be determined by the length of a straight line between these two objects [29]. So given an object p_1 with location $p_{1_{(x,y)}}$ and object p_2 with location $p_{2_{(x,y)}}$ in R^2 space, the Euclidean distance between p_1 and p_2 can be defined as follow.

$$dist(p_1, p_2) = \sqrt{(p_{1_x} - p_{2_x})^2 + (p_{1_y} - p_{2_y})^2}$$

Most of the early works on spatial queries are based on the Euclidean space, in which they utilise the distance metric calculation as shown above. Many different types of queries are proposed. However, the most fundamental query in spatial databases is **Range Query** [30, 31, 32, 33, 34, 35]. In this query, a range of search (usually within a certain radius) will be specified based on a reference point of location. User will give a specific location as the centre point for the range search and then the spatial range query will search for objects within the specified radius [36]. Thus the input for Range Query is a range based on a location point, whilst all the objects inside the radius are the output. Figure 2.1 shows an example of Range Query. Assume that our query location is identified by the blue point q and the query is to find the nearest restaurants within 500 meters from q. In this case, the input of the query consists of location of q and the range radius, which is

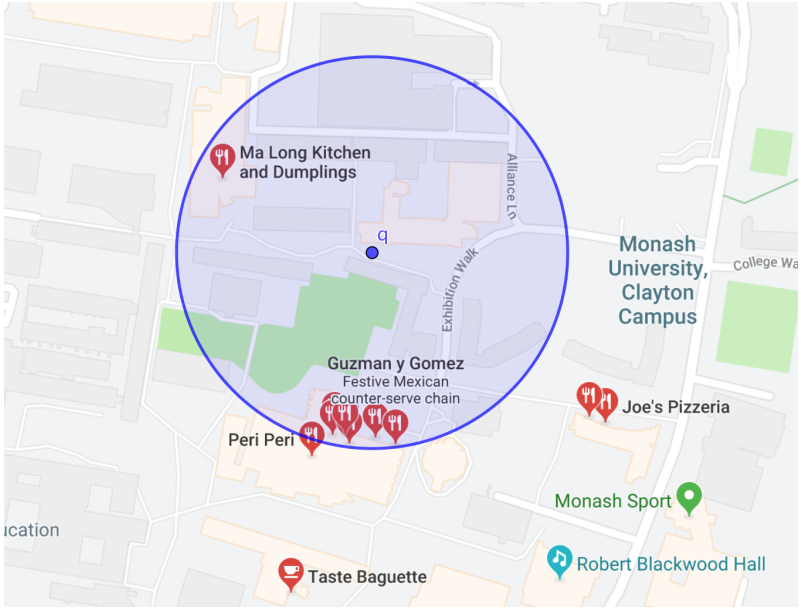


Figure 2.1: Range Query

500 meters. As can be seen in Figure 2.1, the restaurants inside the radius (denoted by the blue circle) are the output to this query. Other restaurants that are outside the range are not considered as the nearest restaurants from our location.

Another traditional spatial query is the **Nearest Neighbour Query** [37, 38, 39]. It is a type of query where we are looking for a certain number of closest objects, that is specified by k, from a given location point [37], which is also known as kNN. The Range Query considers the objects within a certain radius as the nearest objects for the query point, whilst the kNN collects k number of nearest objects measured by their road distance [40, 41]. So instead of getting all objects within a region as what the Range Query does, we specify how many objects that we want to retrieve in kNN without any distance constraint. For example in Figure 2.2, the 3NN from point q are denoted by the blue arrows.

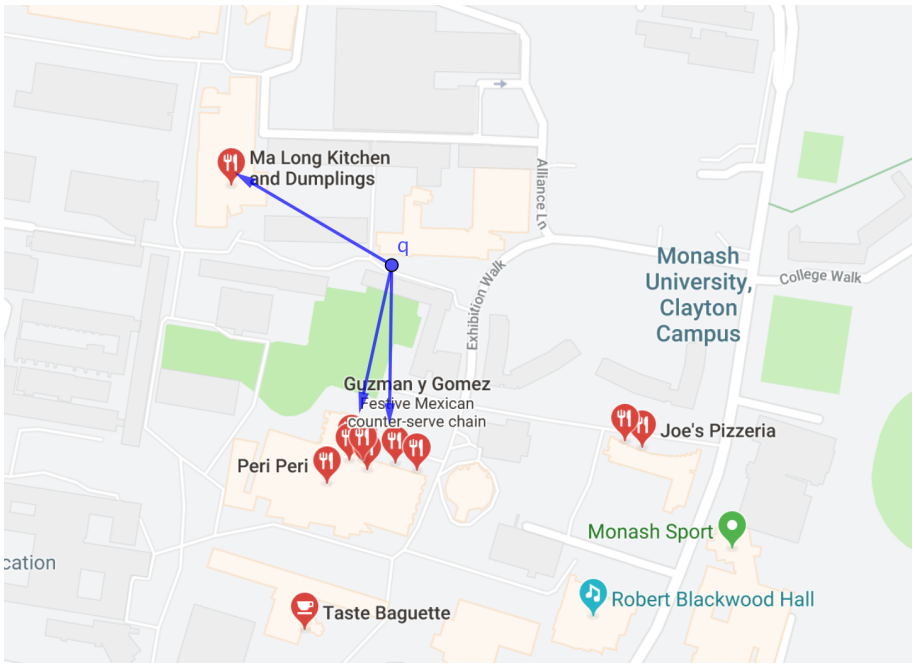


Figure 2.2: k-Nearest Neighbour (kNN) Query

A lot of techniques have been proposed in order to process the *Range Queries* and *k-Nearest Neighbour Queries* in Euclidean space. The most common solution to solve these problems is by using the ***R-Tree*** data structure. The idea of *R-Tree* is to group several spatial objects based on their closeness [30, 42]. Figure 2.4 shows the *R-tree* built from the collection of spatial objects that are grouped as block of rectangles by *Minimum Bounding Rectangle (MBR)* in Figure 2.3. An MBR is a rectangular box that covers the area of objects that are close to each other. One group of MBR can contain several objects. The object inside the MBR can be in a form of a point, a rectangle, or even any irregular

shaped object. In the example in Figure 2.3, we use rectangle as the object form. Each object is grouped into one MBR and then each MBR is then grouped again with the other MBR to create an even bigger collection that can be represented as *R-Tree*.

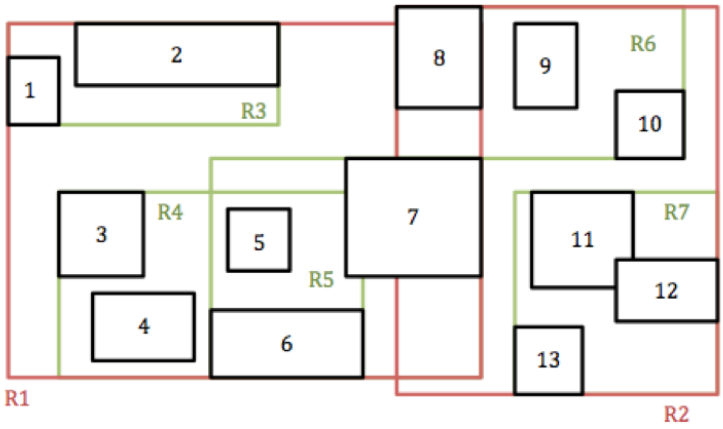


Figure 2.3: Collections of Spatial Objects inside MBR

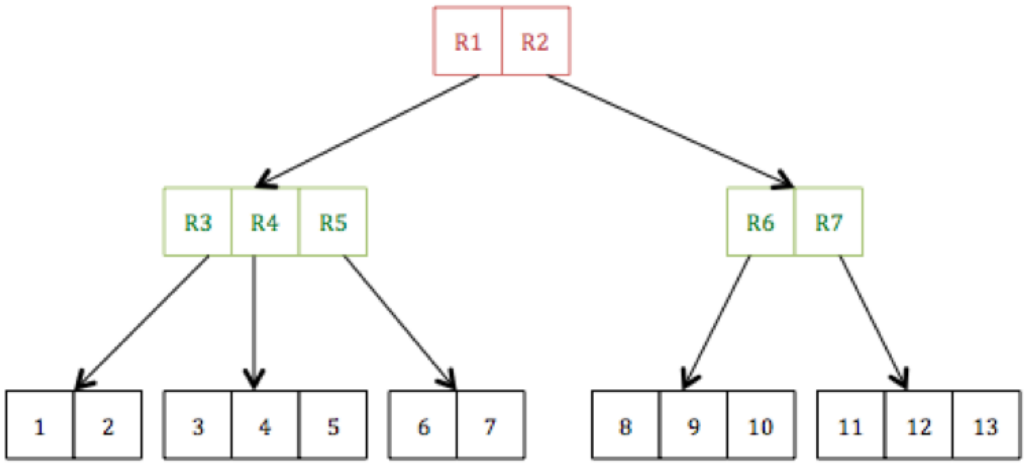


Figure 2.4: R-Tree

The way the *R-Tree* data structure is used to solve the *Range Query* is by giving a range scope that specifies the radius that the user wants and then check if each node in the tree intersects with the query scope. The traversal starts from the root to the leaf node, in which each object in the leaf node will be evaluated whether it overlaps with the query scope. If the object overlaps, then it will be returned as the query result. Meanwhile to answer the *kNN Query* with *R-Tree*, we can perform repetitive range searches in the

R-Tree. The way it works is by specifying a certain point that can be a potential nearest neighbour, and then based on the distance between query point to this chosen point, we can do a range search. If there are some points that have lesser distance than the chosen point, then these points are stored inside a priority queue in which we may be able to get the result of the kNN query. If no other points inside the range between the query point and the chosen point and we need to retrieve more objects to satisfy the number of k, then the range has to be expanded again in which the same method is repeated.

There are a number of existing works that proposed some variants to the *R-Tree* and offer improvements to the existing *R-Tree* index, such as the *R*-Tree* [43] that improves the query processing performance of *R-Tree* despite its higher construction cost, the *R+-Tree* [44] that ensure non-overlapping nodes in the tree which guarantees single path traversal for a point search, the *Hilbert R-tree* [45] that does ordering based on spatial objects instead of region hierarchy, and also the *X-Tree* [46] that is capable to index high-dimensional data.

There are still many other spatial queries on Euclidean distance which are based on the traditional queries, especially on the kNN query. Some of them are the *Reverse Nearest Neighbour (RNN) Query* [47, 48, 49], *Group-KNN* [50, 51], and *The Nearest Neighborhood (NNH) Query* [52]. Different from kNN, the RNN sees the relationship between query objects and its neighbouring objects in an opposite point of view. Meaning that the RNN query returns every neighbouring object that consider the query object as the k closest object. The RNN query has been widely studied and the solutions are varied from point-to-point solutions [53, 54, 55, 56] to region-based solutions [57, 58, 6, 59, 60, 61] that usually based on the Voronoi diagram [62]. Another variant, which is the *Group-KNN* query, handles multiple query objects instead of a single query like the traditional range query and kNN query. In *Group-KNN*, given a set of query points, the query returns a single target object that has the minimum distance to all of the query points. Some solutions to this query are *Group closest pairs method* [40] and *Minimum Bounding Box Method (MBM)* [50]. Another interesting variant of kNN query is the *Nearest Neighborhood Search (NNH)*. In the traditional kNN Query, the k objects may be located far from each other. Most of the time, these objects are scattered everywhere around the space in which is very inefficient if we have to visit each object. For example in Figure 2.2, the 3NN includes Ma Long kitchen. If this particular restaurant is closed, then we have to find

another restaurant in which the distance to go to the next restaurant might be further. So the NNH Query tries to solve this problem by clustering the nearest set of query objects that are close to each other, which is shown in Figure 2.5. In this case, Choi *et al.* [52] proposed a solution of region-of-interest (ROI) instead of the traditional point-of-interest (POI). The ROI itself contains multiple POI.

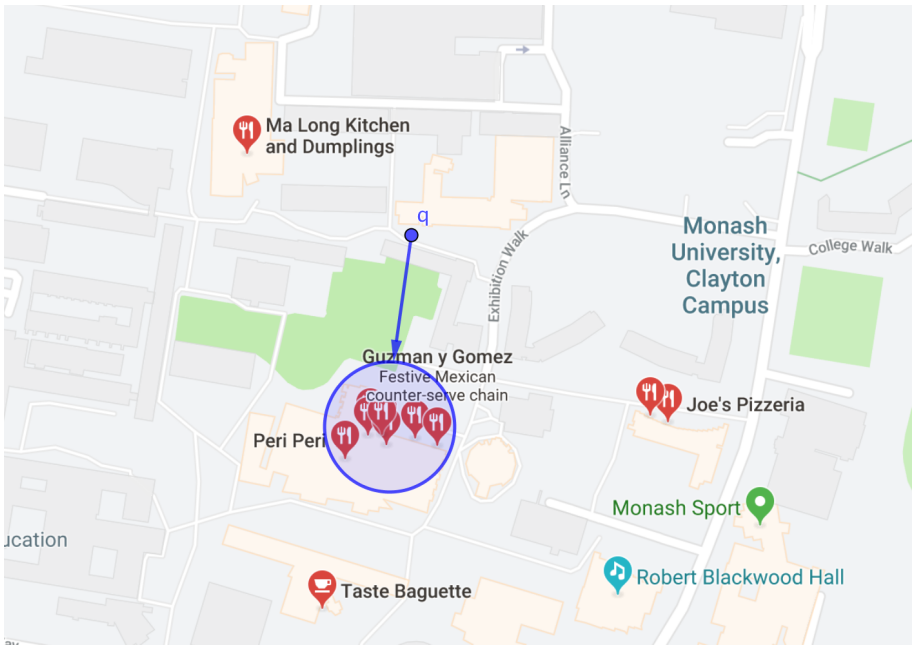


Figure 2.5: Nearest Neighborhood (NNH) Query

2.2.2 Spatial Queries on Road Networks

A popular space model in Spatial Databases area is the Road Networks. In the Euclidean space, the distance between two objects is a straight line. However, this is not the case for road networks. In road networks, the distance between two objects is restricted to a pre-defined path as real road network contains complex road components like intersection, curved road, connected and disconnected roads, etc. So the Euclidean space does not always represent the real world situation.

In spatial databases, a road network is usually represented as a graph that consists of edges and vertices. Each edge in a road network normally has a certain weight that indicates the cost to pass the edge (e.g. distance or time). The spatial objects on road networks can be placed at the vertices or edges. However, most studies simplify the object location to be at the vertices. The distance between two objects in a road network is calculated through the sum of weights of all the edges that connect these two objects.

A fundamental problem in road networks is the **Shortest Path** problem. In shortest path, given a source point and a destination point, find the path with the least amount of distance. This problem has been widely studied and a number of solutions have been proposed, from having an exact solution to heuristic. A famous technique in finding the shortest path is the *Dijkstra algorithm* [63]. Dijkstra algorithm is undeniably one of the best methods to solve the shortest path problem. It is proven from how widely this algorithm is used within the spatial databases community for decades and many applications today are adopting Dijkstra algorithm, including the Google Maps [64, 65]. In Dijkstra algorithm, given a source and destination nodes, the algorithm progressively exploring the graph from the source node to its neighbouring nodes while calculating the amount of distance. Throughout the graph exploration, the algorithm also marks the minimum distance from source node to each visited neighbouring nodes. At the end, the expansion stops when the destination node is found. The worst-case complexity of Dijkstra algorithm is $O(|E| + |V| \log |V|)$, where $|E|$ is the number of edges and $|V|$ is the number of vertices.

Another well-known path finding algorithm is the *A* Algorithm* [66, 65]. It is a heuristic algorithm, where it estimates the lowest distance from start to target node in order to create the path. The A* algorithm uses estimation in order to find the shortest path between the start to target node. It applies the heuristic cost formula $f(n) = g(n) + h(n)$, where $g(n)$ is the minimum cost of the path from start to n, and $h(n)$ is the estimate cost from n to the target node. In terms of running time, the A* algorithm can outperform Dijkstra algorithm. The worst-case performance complexity of A* algorithm is $O(|E|)$, where $|E|$ is the number of edges, while the worst-case complexity for space consumption is $O(|V|)$ for $|V|$ as the number of vertices.

The Dijkstra algorithm and A* algorithm are the type of algorithm that does not involve any pre-processing. Meaning that there is no index structure needed in order to run these algorithms. However, with the higher demand and increasing data size for road networks in spatial databases, the query performance for algorithm without any pre-processing may become poor. Thus a lot of recent studies proposed some indexing techniques that can speed up the query processing performance for spatial queries on road networks. Some of them are Spatially Induced Linkage Cognizance (SILC) [67], G-Tree [68, 69], Route Overlay and Association Directory (ROAD) [70].

The SILC index [67] pre-computes the shortest paths from one of the vertices in the graph to all other vertices, and then does the colouring operation based on the pre-computed paths. Each vertex adjacent to the chosen vertex is assigned with a unique colour. The vertices that are close to each other and have the same colour are stored in a coloured region using the Quadtree index. The SILC framework takes advantage of path coherence between vertices in a spatial network in order to determine the path and distance information between all pairs of vertices. For example, given a road network graph as in Figure 2.6 and we choose vertex A as our source vertex. So in this case, we have to pre-compute the shortest paths from vertex A and then we can start the colouring operation. In Figure 2.7, we start colouring the vertices adjacent to vertex A, which is B, C, H, and J. Then the next nearest vertices are assigned the same colour as the vertices adjacent to A. We keep colouring until all vertices have been assigned a unique colour. After we have the colours for each vertex, we can see that our road network has contiguous coloured regions (Figure 2.8). From these coloured regions, we then can store them in a region Quadtree. The way SILC finds the shortest path between two vertices depends on the colours of the vertices. The colour of the destination vertex determines the first next path from source vertex. Using the example in Figure 2.9, assume that we want to find the shortest path from A to D. D has the same colour as B, therefore the next path from A is through B.

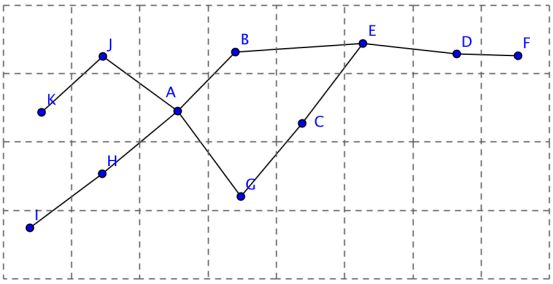


Figure 2.6: Example of a road network

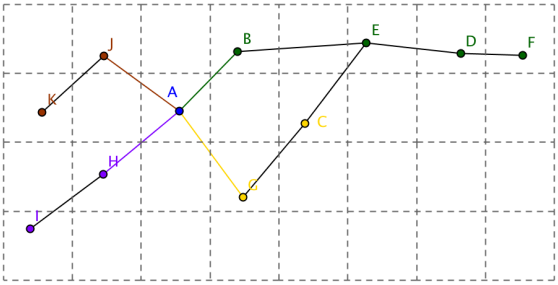


Figure 2.7: Colouring nodes

As SILC involves some pre-processed data to be stored, the space complexity of this method is $O(|V|^{1.5})$. Meanwhile, it takes $O(|V|^2 \log |V|)$ for pre-processing time. In terms of its runtime complexity, the shortest path can be calculated in $O(|E| \log |V|)$, which performs faster than the traditional techniques without any pre-computational index.

Different from SILC, in G-Tree [68, 69], the road network is partitioned recursively into sub-networks. The nodes in G-Tree correspond to a single sub-network, which in

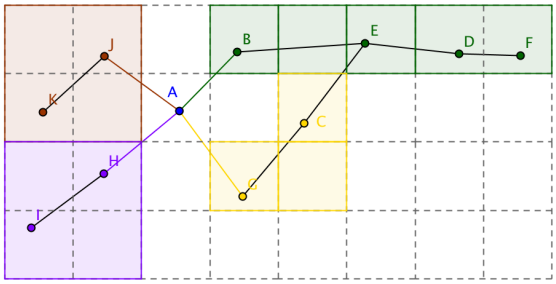


Figure 2.8: Colour region

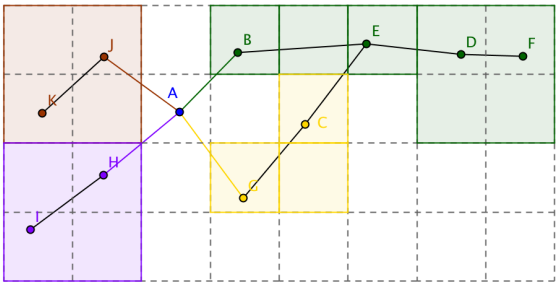


Figure 2.9: Coloured Quadtree region

this case each node must contain 2 or more road network vertices. G-Tree does not store the distance of every vertices but stores the set of borders and the distance matrix. The usage of distance matrices in this type of index is proven to be very efficient in terms of processing the kNN search on road networks [68, 69]. As an example using the road network in Figure 2.6, we try to create the G-Tree. In this case we partition the graph into smaller subgraphs. Figure 2.10 shows the graph partition for the example in Figure 2.6. The graph is divided into equal-sized subgraph and each partition consists of two or more vertices. Then after the partition process, we mark the borders of each node, which is shown in Figure 2.11. The borders here are the vertices that connects their partition to the other partitions. Each partition makes up one node in the G-Tree, and each node contains one or more borders that connect the corresponding partition to other partition. The final G-Tree of graph in Figure 2.11 is shown in Figure 2.12. G-tree itself is a height-balanced tree with a space complexity of $O(|V| \log |V|)$.

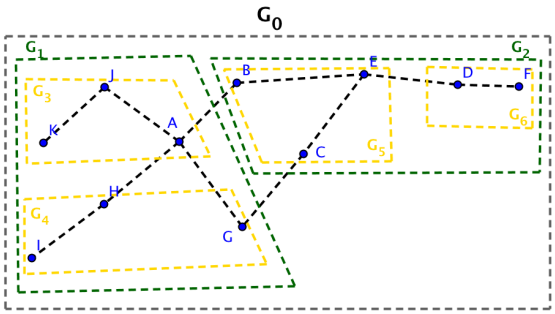


Figure 2.10: Graph partition of Figure 2.6

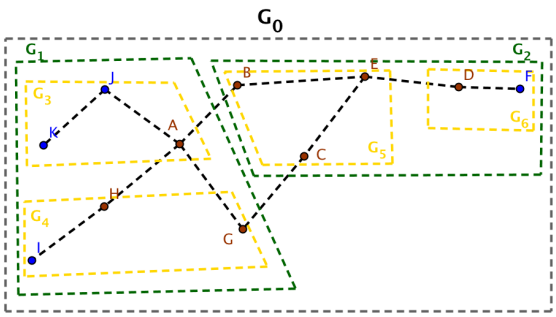


Figure 2.11: Marking the borders of each nodes

The problem in road networks environment is not just about the Shortest Path. Similar to the Euclidean space, the most traditional spatial queries on Road Networks are also the **Range Query** and the **k-Nearest Neighbour Query** [71, 72, 73, 25, 74, 75, 76]. The main difference between the Euclidean space and road networks for these queries are

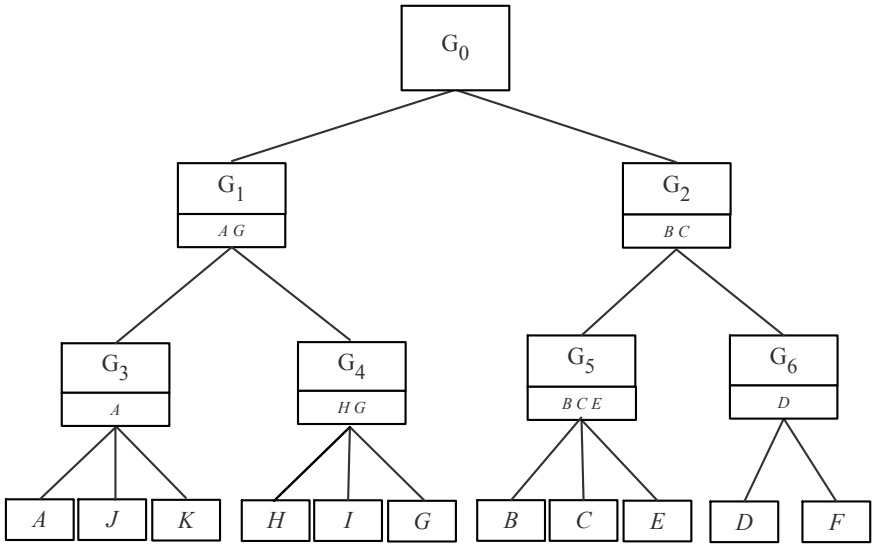


Figure 2.12: G-Tree of the graph partition in Figure 2.11

on the distance metric used. Looking at Figure 2.13, assume that our query location is at the blue point q and we want to find the nearest Coles supermarket from q. In the Euclidean space, the nearest Coles supermarket is the North Store. However, based on the Road Network distance, the nearest one is actually the Centre Rd one. This shows that we cannot always rely on the Euclidean distance in real life.

Figure 2.13: Example on Road Networks

Some approaches for Range Query in Spatial Road Networks are *Range Euclidean Restriction (RER)* and *Range Network Expansion (RNE)* [77]. The main idea behind RER is in the usage of the Euclidean distance range diameter for upper boundary of the search area. So all objects that are outside the upper bound limit will be pruned, while the objects inside the boundary will be examined if they are within the road network range. The RNE on the other hand consists of two basic steps, which are Qualifying Segment and Index Tree Traversal. In the Qualifying Segment step, the paths are expanded from the query point location to the specified range distance. So all objects that are visited in this expansion are included to the result. Meanwhile for the Index Tree Traversal step, using the R-Tree index, we check every node that we visit whether it intersects with the Qualifying Segment's objects.

Similar to Range Search, Spatial Road Network for kNN consists of two algorithms of expansion and restriction, which are *Incremental Euclidean Restriction (IER)* and *Incremental Network Expansion (INE)* [78]. The IER adopts the Euclidean distance as the upper bound range for finding objects of interest [30], while INE adopts Dijkstra like algorithm that expands the query to find the chosen k objects [79, 63]. There is also another solution using Voronoi Diagram. The famous voronoi approach in kNN is the *Voronoi Network-based Nearest Neighbour (VN^3)* [74]. This method uses Network Voronoi Diagrams (NVD) [72] and it requires a pre-computation shortest distance between each spatial object both within the same Voronoi polygon and in different Voronoi polygons [78]. Another Voronoi-based kNN is the Progressive *Incremental Network Expansion (PINE)*, which is an improvement of VN^3 [62]. This method only requires a pre-computed distance for objects within the same Voronoi polygon, while the distance between objects in different Voronoi polygons can be calculated anytime [80].

2.3 Spatial Keywords Queries

With the growth of spatial databases applications these days, a lot of spatial queries are not only considering just the spatial/geographical information, but the queries are also involving other factors like textual information, which is known as Spatio-Textual or Spatial Keyword Queries. The involvement of textual information is usually through the keywords that users input when invoking a spatial query. This highlights the need of

keyword search in the query processing stage. Due to this reason, the complexity escalates for this kind of query as we have to combine techniques from Spatial Databases area and also Information Retrieval (IR) area to process the keywords. Thus in this section, we discuss the existing techniques and some variations to Spatial Keyword Queries.

Some of the basic queries in Spatial Keywords are the **Boolean kNN Query** and **Boolean Range Query** [19, 81, 82]. In the Boolean Range Query, the query specifies a range for searching and then retrieves all objects that match the query keywords within the range. The Boolean kNN Query does a simple k nearest neighbour objects retrieval, which the objects must match with the query keywords. The basic idea behind these queries are similar to the spatial-only Range Query and kNN Query, but with the additional feature of textual dependency from the objects that are affiliated with certain keywords. As both queries are boolean based, the way the query is being processed in terms of the keyword part is through exact matching. A common technique in answering these queries is through creation of index structure, in which the index for Spatial Keyword Queries usually combines spatial index structure together with textual index structure. Most of the available Spatial Keywords index are based on R-Tree [83, 84, 81, 85, 1] and grid [86, 87] for the spatial side, while for the textual side is based on inverted file [88].

Deriving from these simple queries, researchers start to develop more progressive Spatial Keyword Queries that can handle more complex user demands. Most of the early works on Spatial Keyword Queries focus on queries like **Top-k Nearest Neighbor Queries (TkNN)** [1, 19, 2, 82, 89, 85, 90]. In TkNN queries, the goal is to rank objects, measure the keywords similarity (between the object's keyword and query) and the distance from the specified query location, in order to retrieve k number of objects with the highest ranking. This type of query mainly accepts user's spatial location and keywords as input, and produces spatial objects with matching keywords as the output.

A variation of Spatial Keywords Query is proposed by Wu *et al.* [91]. They proposed **joint top-k spatial keyword** that processes multiple queries. Their indexing technique is mainly using the IR-Tree. To process the multiple queries, they proposed a GROUP algorithm which groups the queries in order to find their joint results. Another variation is also done by Wu *et al.* [92], where they are the first to consider **moving object on Spatial Keywords Query**. The authors proposed safe zones, which is when the query is inside the zone, the top-k results will always be correct even though the query moves.

So when the query is outside the safe zone, the top-k results need to be regenerated. They adopt their technique based on the weighted Voronoi diagram, which they named it as *Multiplicatively Weighted Voronoi diagrams.*

As most of the current techniques only retrieve an individual object in top-k manner, the **Collective Spatial Keyword Querying** [93] introduced a technique to retrieve a group of objects together. So for example when a tourist is looking for a hotel, a restaurant, and a public attraction that are close to each other. So their technique is to find objects that are spatially close and covers all the query keywords. They use the IR-Tree, which is R-Tree with extension of inverted files. However, the technique proposed by Cao *et al.* [93] has a weakness in its scalability. It does not guarantee the optimal solution for the query. Thus, Long *et al.* [94] improved the scalability problem. They focus on the distance and try to calculate the closest and farthest possible query distance.

Based on the queries we discussed previously, we can see that many new queries on Spatial Keywords have been explored. But there are still a lot of other Spatial Keyword Queries variants that have been studied. Many of those queries are mainly based on TkNN queries, in which the works on these variants try to improve the TkNN queries to be able to process moving objects [92], continuous objects [95], reverse top-k query [96, 97], joint queries [91, 98], or interactive TkNN queries [99]. Besides the works on Spatial Keyword Queries that focus on TkNN queries, some variants of Spatial Keyword Queries have also been proposed, such as the collective Spatial Keyword querying [93, 94, 100], diversified Spatial Keyword search [101], region-based query [102], scalable continual top-k query [103], reverse spatial and textual k nearest neighbor query [104], spatio-textual data clustering [105], fuzzy keyword search [106], and m-closest keyword queries [107].

As Spatial Keyword Queries become varied, a number of indexing techniques that are able to process both spatial and textual data have been proposed in the past years. A lot of the indexing technique on Euclidean space are utilizing the R-Tree in which they attach additional textual information into the R-Tree to be capable of computing textual data. Some of those R-Tree based indices are IR-Tree [84], bR*-Tree [17], and IR^2-*Tree* [1].

A recent work is done by Zheng *et al.* [99], which highlights that most of the previous works only consider exact matching for the keywords and no interaction between the user and the application. In this paper they proposed interactive solution where the user can give feedback to the results by setting the threshold. The indexing technique here is by

using the $IR^2 - Tree$ in [1]. The $IR^2 - Tree$ itself is first introduced by Felipe *et al.* [1]. It is a Hybrid Indexing approach that combines the R-Tree and information retrieval signature files. The indexing technique is by attaching the inverted index to the R-Tree. Every node in $IR^2 - Tree$ holds the information for both spatial location and keywords. The leaf nodes contain the actual spatial data and keywords. The non-leaf nodes contain the combination of several objects. The spatial information is based on the MBR, while the inverted index of the keywords is calculated using logical OR [1]. The example of $IR^2 - Tree$ can be seen in Figure 2.14. Felipe *et al.* [1] uses incremental algorithm that uses the $IR^2 - Tree$ to solve the spatial keyword query. However the downside of current solution is that the retrieval only considers exact matching. It does not consider the synonyms or typos. For example when we are looking for sneakers in a shopping mall. In this case, sneakers can also be considered as shoes. So we should consider shoes shops in our retrieval process.

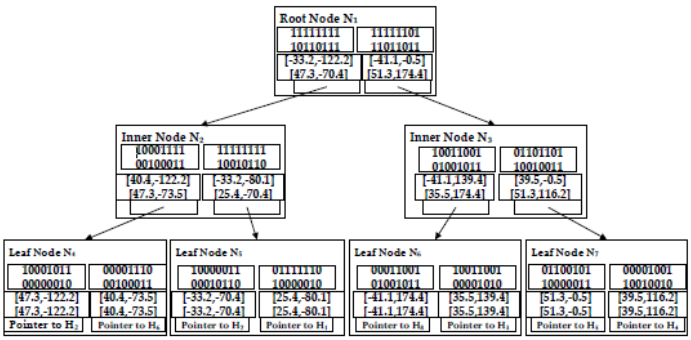


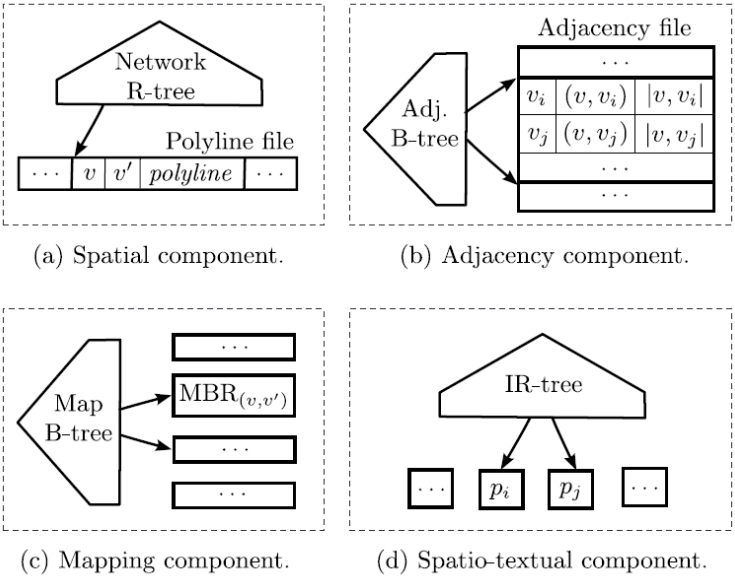
Figure 2.14: $IR^2 - Tree$ [1]

In the work of Alsubaiee *et al.* [106], a method to solve the inexact matching is introduced. The fuzzy keyword search is proposed in order to find keywords without exact spelling, especially for users that do not know the exact words that they are looking for (e.g. unique name of a restaurant). Their method is by having gram based inverted index attached in one level of R*-Tree. However in this technique, it can only solve the inexact matching of similar words, not other keyword variations such as synonyms and homonyms.

A work on spatial keyword query on road network is done by Zhang *et al.* [101]. On this work, they focus on diversified query results in order to improve the quality. Diversified result means that the distance of the retrieved objects is reasonably large so that the result

offers more diverse selection. They proposed a method to always consider user relevancy and the spatial diversity through some criteria. Another variation in spatial keyword query on road networks is by Gao *et al.* [97]. As all of current approaches only solving the top-k spatial keyword queries, [97] proposed a solution to Reverse Top-k Boolean Spatial Keyword Queries. However in their pruning algorithm, they are using heuristic approach.

As the previous indexes are only based on Euclidean distance, Rocha-Junior *et al.* [2] applies spatial keyword query on road networks. Their basic indexing architecture consists of four components (Figure 2.15). The first component is spatial component which is using the Network R-Tree. The second is adjacency component, which it uses adjacency B-Tree to traverse the network. The third component is mapping component, which it uses Map B-Tree to map the adjacency edges with MBR that encloses the edges. The last component is the spatio-textual component, which it stores the spatial and textual properties of the objects. While for the query processing, the authors adopt the Dijkstra's algorithm. Another work on road networks is also done by Luo *et al.* [108]. They introduced a new indexing technique that is very different from Rocha-Junior *et al.* [2]. The proposed index, which is the Node-Partition-Distance (NPD) index, keeps useful distances so that the exact distance and the query keyword coverage can be computed independently.



(a) Spatial component. (b) Adjacency component.

(c) Mapping component. (d) Spatio-textual component.

Figure 2.15: Indexing architecture for road network [2]

As the previous queries are focusing more on object retrieval, Cao *et al.* [22] proposed a route planning query for Spatial Keywords, which is the **Keyword-aware Optimal**

Route Search (KOR). KOR is a query that finds an optimal route that covers a set of user given keywords with a specific budget constraints and objective score [22]. It is similar to TPQ but KOR is more expensive as it includes an additional constraint (the budget constraint). They proposed three approximation algorithms in order to solve the KOR query. The first one is OSScaling, where they scale the objective score of each edge into integers utilizing a parameter ϵ in order to acquire a scaled graph. This method is to reduce the cost of calculating the partial paths that are stored on each node when the brute-force approach is established. The second algorithm is BucketBound, which it splits the traversed partial routes into different "buckets" based on the best objective scores. This method is claimed to be more efficient than the OSScaling. The last algorithm is a greedy technique. The processing starts from the starting location and then it keeps selecting the next location greedily with consideration of all the three constraints in KOR query. The algorithm will keep repeating the step until the target location is reached. In conjunction to Best Path Query, KOR is different as the query requires budget constraint for processing. KOR also does not take into consideration negative keywords as what Best Path does.

Through the above discussion, researchers have been trying to create some indexing techniques in order to process the query easily. But the technique proposed are still treating the Spatial Data part and the IR part as two different entities. Current techniques adopt hybrid indexing, which they always have separate index for the spatial data and the textual data and then combines both indexes.

2.4 Route-Planning Queries

With the advancement in spatial databases area, there is an increasing demand to support complex queries, including route planning queries. A well-known route planning query is the **On Trip Planning Route Queries (TPQ)**. TPQ is a spatial query that retrieves the best trip from two different locations that passes at least one point from each of the chosen categories [20]. So assume that there are a number of spatial objects that are stored in a spatial database and each object belongs to one or more categories from a fix set of categories C. A user provides a source point S, a destination point E, and a subset of categories R ($R \subset C$), the query is to find the best route that starts from S to E that

passes through at least one point from each category in R [20]. TPQ itself is regarded as a generalisation of Traveling Salesman Problem (TSP) problem [23, 24], meaning that it is also an NP-hard problem. The solutions proposed for TPQ are polynomial time approximation algorithms. Li *et al.* [20] utilised R-Tree to index the objects and they also proposed several approximation algorithms based on the triangle inequality. The approximation ratios are depending on the total number of categories and the maximum category cardinality.

In relation to Best Path Query, TPQ and Best Path are similar in finding the best route that passes several specified objects. However there are some differences between these two queries. Best Path Query focus on Spatio-Textual field, which in this case it needs to process the textual part of the objects. TPQ itself is without keywords. Another difference is that TPQ does not take into consideration any negative situation that can obstruct the path.

As the TPQ only considers one single user, Hashem *et al.* [109] proposed **Group Trip Planning (GTP) Query** in order to process multiple users trip. In GTP query, for a set of n users, let S represents a set of source location $S = \{s_1, s_2, ..., s_n\}$ and D represents a set of destination location $D = \{d_1, d_2, ..., d_n\}$. If the group of users plans a trip of m types of data points, the GTP query returns a set of data points $\{p_1, p_2, ..., p_m\}$, $pt \in Dt$, with minimum cost of route. The GTP query itself consists of two types of query, namely the *ordered GTP query* and the *flexible GTP query*. The difference in these two types is in the results returned, where the ordered GTP query returns a sequential data points based on user request, whilst the flexible GTP query returns unordered data points. In previous study, Hashem *et al.* [109] also worked on k group trip planning (kGTP) query, which is a query that returns k sets of data points for a group trip. The algorithm proposed for the GTP problem consists of *Iterative approach* and *Hierarchical approach*. In the Iterative approach, the kGTP query implements the method of group nearest neighbor (GNN) queries, where in this case the query returns the locations of k objects that have the k lowest total travel distances for the group [109]. The Hierarchical approach on the other hand, uses the R*-Tree for the hierarchical properties of the dataset and then uses a modified best first search (BFS) on the R*-Tree to find the k sets of data points that minimise the total travel distances for the group trip [109].

Another popular route planning query is **Optimal Sequenced Route Queries (OSR**) [21]. OSR is a spatial query that finds the minimum route distance from a source location and passing through a set of sequenced categories. OSR is closely associated to TSP problem [21]. However this query is different from TPQ as TPQ does not retrieve its result in sequence. This query however is no longer an *NP-Hard* problem because of the sequence constraint. There are several techniques that can be applied for this query, which are the Dijkstra based solution, OSR in Vector Space: LORD - Light Optimal Route Discoverer, R-LORD - R-Tree based LORD, and OSR in Metric Space [21]. The Dijkstra based solution is a naive approach to process OSR in a weighted directed graph. However this naive approach is impractical for a large graph, especially on real world problems. Therefore Sharifzadeh *et al.* [21] proposed LORD, which employs several threshold values in order to filter possible unimportant points that cannot be on the optimal route and then establishes the optimal route in reverse sequence. Then they proposed the R-LORD, which improved the LORD method by employing the R-Tree index structure for filtering. Sharifzadeh *et al.* [21] also proposed PNE method that can solve OSR in metric spaces (e.g., road networks).

In conjunction with the Best Path Query, OSR and Best Path both are trying to establish a route that passes several categories of object in an optimal distance. However, the main difference between OSR and Best Path is that OSR's result is in sequence. OSR does not perform textual relevancy checking like what Best Path does. The OSR also does not take into consideration negative situation.

Route planning is very popular in which the study also expands to indoor environment [110]. Shao *et al.* [110] was the first to study **indoor trip planning query**, which they called ***i*TPQ**. They utilised VIP-Tree to index indoor spaces and objects. They also proposed an exact algorithm called VIP-Tree neighbor expansion (VNE) that can do two levels of pruning at both pre-processing phase and query processing phase. Even though this study offers an exact solution, it is not compatible with Best Path as we are dealing with a totally different space. The scope in indoor space is smaller than road networks and weighted regions, thus it is possible to propose an exact solution despite the fact that trip planning query is an *NP-Hard* problem.

Another route planning query variant in the area of Spatio-Textual is the **Keyword-aware Optimal Route Search (KOR)** [22]. The goal in KOR is to find an optimal

route that covers a set of user given keywords with a specific budget constraints and objective score. This query has been discussed in the previous section (Section 2.3).

There are more variants to the route planning queries, such as *Multi-Rule Partial Sequenced Route (MRPSR) query* [111] that unified framework and classified both TPQ and OSR, *Keyword-Aware Skyline Routes (KSR) query* [112] for indoor space which returns a set of non-dominated routes (based on the route distance and the number of shops/stores visited) instead of an optimal route, and *Personalized and Sequenced Route (PSR) Query* [113] that considers multiple factors of a route and associates different weights with each category for the route. Some recent studies on other factors that may influence route planning have been done as well, such as a study on TPQ with Location Privacy [114] in order to protect user's location privacy, and also a new tourism personalised route recommendation algorithm [115] that mines travel text data history to establish a more personalised travel route that can increase user satisfaction. However, all of these studies have different goals and settings to the Best Path Query.

The solutions offered for the existing queries on route planning are mostly in polynomial time approximation algorithms. Even so, the solutions offered do not involve significant pre-processing. This causes more computation, particularly since the computation requires the processing of both spatial and textual relevancy. So having an on-the-go solution does not always guarantee performance efficiency, especially on Spatio-Textual field. Therefore in this research we attempt to provide a better solution to this problem by offering a pre-processing index that incorporate both Keywords and spatial information. Our study also takes into consideration of the existence of negative keywords, which to our knowledge has not been considered in any previous works in Spatio-Textual area. Our work also involves different space model like Road Networks and Weighted Regions.

2.5 Weighted Regions

Another space model other than the Euclidean space model and Road Networks model is the Weighted Regions. The idea of weighted region space derives from the fact that the surface of the earth comprises various geographical features (e.g. grasslands, deserts, concretes, etc.) that can affect the travelling cost to pass through each type of surface [27]. The weighted regions model is commonly represented as a planar polygonal subdivision

that consists of a set of edges, vertices, and faces. Each face is associated with a distinct positive unit weight $[1, +\infty]$, which used to model the different geographical features. In this section, we discuss some existing works on weighted regions, particularly on the Shortest Path problem. We also discuss the wNeighbor, which is the only existing study on spatial query in a weighted regions model.

2.5.1 Shortest Path on Weighted Regions

The shortest path problem on weighted region model has been widely studied in the past decade. The problem defines as follow, given a triangulation $DT(P)$ and each face f in $DT(P)$ is associated with a weight $[1, +\infty]$, find a path from point s to point t in $DT(P)$ that gives the minimum cost among all other possible paths. A number of solutions have been proposed to solve this particular problem [116, 117, 118, 119, 27, 120]. However, most of the studies are on finding approximation solution as computing the exact shortest path on weighted region is very costly [117].

A notable solution to the shortest path problem on weighted region is offered by Mitchell *et al.* [116] where they proposed the first shortest path algorithm on weighted regions that runs in $O(n^8 \log \frac{nN\mu}{\varepsilon})$ time. The n in this case is the number of vertices, N determines the maximum integer coordinate of any vertex as all the vertices have integer coordinates in $[0, N]$, while μ indicates the ratio of the maximum region weight to the minimum region weight. Their algorithm fundamentally establishes a shortest path map for the source point s by applying continuous Dijkstra wavefront in $DT(P)$ and exploits the Snells Laws of Refraction on the shortest path implementation.

Another notable approximation solution is proposed by Sun *et al.* [27], where they proposed BUSHWHACK algorithm. The idea behind their algorithm is to dynamically maintain Steiner points in which these points are located in the edges of a region that can be used to approximate the weighted shortest path from s to t. This algorithm runs in $O(\frac{n}{\varepsilon} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$. Subsequently, Aleksandrov *et al.* successfully proposed another algorithm in which its running time has a lower dependence on n and ε. The algorithm runs in $O(\frac{n}{\sqrt{\varepsilon}} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time.

Despite the prominent findings on the solutions for shortest path problem on weighted regions, the study on spatial queries on this particular space model is very limited. Li *et al.* [121] was the first one to do a study on finding k Nearest Neighbour on weighted

regions (WkNN). They introduced Weighted Indexing Map (WIM) to index the weighted regions space and proposed wNeighbors to answer the WkNN problem. A more detailed discussion of wNeighbors is provided in the following section.

2.5.2 wNeighbor

The first work to investigate the k Nearest Neighbour on weighted regions (WkNN) was done by Li *et al.* [121]. In the traditional kNN, given a set of objects in space, find the k nearest objects from a given source location. The nearest objects are usually measured by the distance from the source location, which means that we have to calculate the shortest path from the source location to the objects. However, the space model in WkNN is more complex than the traditional kNN queries, which elevates the computation to solve this problem.

The WkNN query is defined as follow. Given a finite triangulation in the space, edge's weight w_e, face's weight w_f, a query point s, a destination point set D, and a value k for the number of objects to be retrieved. WkNN query finds a result set R of the top k nearest neighbours of point s in a destination point set D, that satisfies

$$\{d(x,q) < d(x',q) | x \in R, x' \in D - R, |R| = k\}$$

where $d(\cdot)$ indicates the weighted distance between two points.

As we have to calculate the shortest path before identifying the nearest neighbours, Li *et al.* [121] did some investigations on the characteristics of Shortest Paths in weighted regions. In their observations, some shortest paths between two points are usually along the edges of the faces, in particular if the faces between these two points are not adjacent to each other. There are also times when the shortest path to a destination point is through one of the vertices of a face where the destination point is located. They also indicate the relationship between Euclidean space and weighted regions space that can help in finding the shortest paths. Deriving from these observations, for a given triangle face f with weight w_f, a source and destination points s and t on different edges (s at the edge e_1 and t at the edge e_2) of face f, β as the intersection angle between e_1 and e_2, and v as the intersecting point of e_1 and e_2, the characters of the shortest path are as follow.

- If $\cos\beta < 1 - 2/m^2$, then the shortest path between s and t must be through the edges of f. The angle β for this case is called the *block angle* as it can block the shortest path that try to cross the adjacent edges of β. If a face contains three block angles, then it is a *block face*.

- If $\cos\beta < 1 - 2/m^2$ is not satisfied, then there is a possibility that the shortest path between s and t is a direct line. When angle β is not a block angle, it is considered as *combination angle* the shortest path can pass through the adjacent edges of β.

In this work, Li *et al.* [121] introduced the concept of Combination Region (CR). It is a combination of region faces that share the same indexing data. A weighted regions plane basically consists of several CRs. CR itself can be identified by an area that contains one block face or faces that are connected by combination angles. In terms of shortest path, given a CR R, if the source point s and destination point s are not located inside R or the neighbouring CR of R, then the shortest path will never cross R. So if a point s in R_s and point d in R_d and $R_s \neq R_d$, then the shortest path between them must go through at least one vertex v_1 in R_s and another vertex v_2 in R_d, which means that the path between v_1 and v_2 are along the edges instead of crossing any regions.

With all of the above characterisations of shortest path in weighted regions, a new data structure called Weighted Indexing Map (WIM) is proposed by Li *et al.* [121]. The WIM stores $crList$, a list of CRs, and also $pList$, an altered R*-Tree that stores the location of vertices and destination points. The WIM is constructed through a three-step algorithm that involves weighted regions plane split (to identify the CRs), calculation of shortest distance from each destination point to the vertices of its CR, and then indexing all the data points (vertices and destination points) into WIM.

Utilising the WIM, wNeighbor algorithm is proposed to answer the WkNN query. wNeighbor adopts both Dijkstra's algorithm [63] and Mitchell's algorithm [116] in its query processing phase. Based on the characterisation of shortest path, if given a query point s in R_s and a destination point d in R_d, R_s is disconnected from R_d, then the shortest path utilises Dijkstra's algorithm. But if R_s and R_d are adjacent to each other, then the shortest path is calculated using Mitchell's algorithm. Based on the shortest paths between the query point to the destination points, we can easily identify the WkNN result.

Looking at the studies on weighted regions, there is no study yet on Spatio-Textual area and route planning. Most of the studies are on finding the best solution to the shortest path problem and only one work on WkNN query. These solutions are not applicable to the Best Path Query as the Best Path deals with spatio-textual objects, weighted regions, and answering route planning queries.

2.6 Summary of Existing Issues

Based on the above discussion of the existing works in Spatial Keyword Queries, there are several limitations that need to be taken into account:

- **L1**: The existing route planning solutions are mainly focused on spatial-only queries. There are very limited solutions towards route planning in Spatial Keywords area, especially the solution to spatio-textual route planning query without any budget constraints.
- **L2**: Current works do not take into consideration of negative keywords that can be obstacle in finding paths.
- **L3**: The existing indexing techniques consider both the spatial information part and textual information part as two different entities.
- **L4**: Current works are only centred on Euclidean and Road Networks space model. There is no work yet on Spatial Keywords Queries on a more complex environment like the Weighted Regions.

Therefore in this study, we attempt to develop some solutions that can handle the limitations of the existing works as mentioned above.

Chapter 3

Best Path Queries on Road Networks

3.1 Overview

Nowadays, each spatial object contains one or more meaningful keywords as to represent the object's entities [14, 1]. The keywords may contain country name, city name, address, references to landmark, or even type of road [15]. For example in Figure 3.1, the points denote spatial objects and each object is affiliated with one or more keywords. Through the existence of this kind of spatial keywords information, the Spatial Keyword Queries become varied. Some of the commonly used queries are the *Top-k kNN Query, Boolean kNN Query,* and *Boolean Range Query* [19]. All of these queries require the user to give a spatial location (normally the current user location) and textual data in the form of keywords as the input. While the output of these queries is spatial objects that are nearest to the user's location and contain the keywords given. The principle parameter used to identify the nearest objects is based on the shortest path distance, which is basically computing the minimum distance between two location points. Although the existing shortest path-based solutions are useful, they are not always sufficient for our needs. In real life, we often want to plan our trip with the most efficient cost (e.g. time, distance) taken. We may need to stop by several locations in our trip before arriving to the designated destination and there are also times when we would like to avoid some spatial objects that can interfere our activities. Planning a trip is eminently more complex than a simple source-to-destination type of query. Unfortunately, the existing studies on trip planning query in Spatio-Textual

area are not flexible enough to answer this kind of query. Furthermore, often at times researchers only consider users to give keywords just to find POIs. But in reality, this is not always the case. Some query keywords may have negative connotations, such as traffic jams, which means that not all user given keywords can be considered as POI.

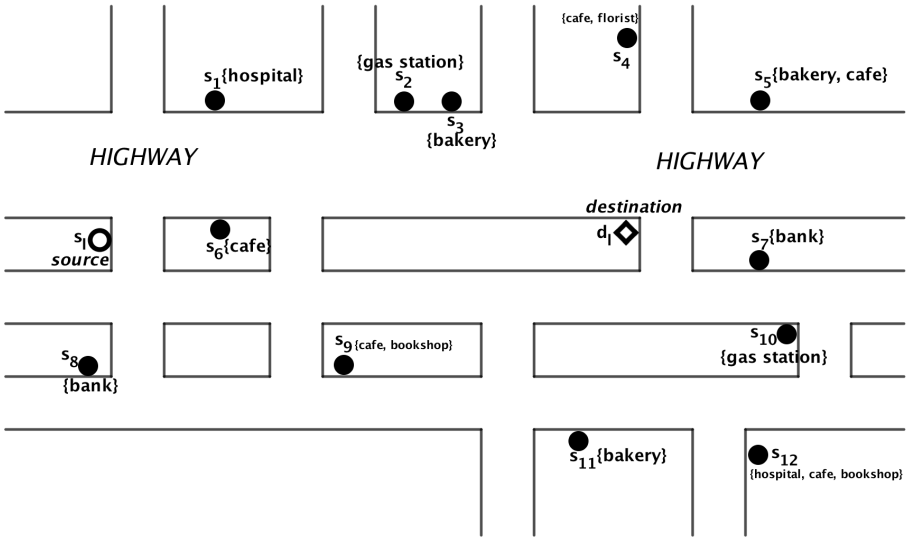


Figure 3.1: Illustration of a road network with spatial keyword objects

In this study, we propose a new variant of *spatial keywords query*. Given a user with his/her location, this user wants to go to his/her destination while stopping by or avoiding several locations denoted by certain keywords. For instance a user wants to go to his workplace from his house, but before arriving to the workplace he wants to stop by a gas station to refill his car fuel, a bakery to get some breakfast, and also wish to avoid any highway along his trip (see Figure 3.1). In this query, the user specifies the source and destination locations and several other keywords. The keywords specified are *gas station*, *bakery*, and avoid *highway*. So we need to find the most optimum path for the user that satisfies his preferred keywords condition. Using the illustration in Figure 3.1, assume that s_l is the user's house (source location) and d_l is his workplace (destination location). There are a number of spatial objects that contain the keyword of *gas station* and *bakery*, such as s_2, s_3, s_5, s_{10}, and s_{11}, hence there are many possibilities of path combination to be established from s_l to d_l passing through at least one of each keyword. Looking at the road network, the path with the shortest distance is from s_l to s_2 to stop by a gas station, then s_3 to stop by a bakery, and then d_l. However this path passes through a highway. One of the keywords specified by the user to avoid is *highway*, which means that the path does not satisfy all the criteria given by the user. Therefore the best path that

offers the least sum of distance and meets the criteria is from s_l to s_{11} for *bakery*, then s_{10} to stop by the *gas station*, then finally the destination d_l (obviously this path also avoids any *highway*). We call this kind of query as the *Best Path Query* (BP).

In Best Path Query, we are dealing with both spatial and textual data. Hence the user given query has two main parts: the spatial data part that consists of the source and destination locations that are given by the user, and the textual data part that consists of the keywords that the user would like to pass or avoid throughout his/her trip to the destination. Based on the user input, the query keyword itself can be classified into *positive* or *negative situations*. As in previous example, the user gave a negative keyword, which is to avoid highway. The positive keyword here is the POI that he wants to pass by, which are the gas station and bakery. The formal definition of Best Path Query is given as follow:

Definition 2. *(**Best Path Query**) Given a source location s_l, a destination location d_l, and a set of keywords $K = \{k_1, k_2, ..., k_n\}$, where each k_i for $1 \leq i \leq n$ can be positive (denoted by k^+) or negative (denoted by k^-), find the Best Path from s_l to d_l, denoted by $BP(s_l, d_l, K)$, that passes through all k^+ and avoid all k^- with optimum cost.*

3.1.1 Challenges

The main challenge in this research is in the insufficiency of current solutions to trip planning in *spatial keywords area*, especially on Best Path Query. Existing studies do not take into consideration negative keywords. Having negative keywords actually increase the complexity of the problem as we have to make sure that we can avoid certain paths. There are definitely some cases where the result cannot be retrieved as we have to avoid all of the paths in between the source and destination locations.

The road networks are usually represented as a graph. The edges typically represent the road segments while the vertices represent the road intersections. So the path generation is always limited only to the edges to adjacent vertices. This increases the complexity of our query computation when we are working specifically in this environment. The complexity is also intensified with the types of query keywords given by the user. When the query keywords are negative, a lot of the paths will be blocked as we have to avoid them. We always have to make sure that we plan the route optimally despite these complexities.

Another challenge is that most of the route planning problems are regarded as a generalization of Traveling Salesman Problem (TSP) problem [23, 24]. They are NP-hard

problems. The solutions offered for these queries are in polynomial time approximation algorithms. Even so, the solutions offered generally involve no pre-processing. This causes more computation particularly since the computation requires the processing of both spatial and textual relevancy. So having an on-the-go solution does not always guarantee performance efficiency. There are also very limited indexing techniques in the Spatio-Textual area, especially on road networks. In this research we attempt to provide a solution to this problem by offering a novel index that incorporate both *keywords* and *spatial road networks information* based on *G-Tree* [68, 69] and IR^2-*Tree*[1].

3.1.2 Contributions

Our main contributions in this chapter are as follows:

- We formally define Best Path problem on road networks and prove that this is an NP-hard problem.
- We develop a novel indexing scheme, called *IG-Tree*, for planning Best Path queries in road networks.
- We present three approximate algorithms with different trade-offs for searching Best Paths on road networks.
- We also demonstrate the effectiveness and efficiency of our algorithms through comprehensive experiments on real datasets.

3.1.3 Organisation

The rest of the chapter is organised as follows. Section 3.2 presents the preliminaries and the query model for Best Path problem on road networks. Section 3.3 discusses the computational complexities of the Best Path problem on road networks. We introduce the *IG-Tree* in Section 3.4 and discuss our algorithms to solve the Best Path problem in Section 3.5. Section 3.6 presents the experimental evaluations of all the algorithms proposed. Finally, Section 3.7 concludes the chapter.

3.2 Preliminaries

This section presents the necessary background information, and the data and query model for *Best Path* problem on road networks.

3.2.1 Road Network

We consider road network as an undirected weighted graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. Each edge $(u, v) \in E$ connects two adjacent vertices $u, v \in V$ and is associated with a non-negative weight $w(u, v) > 0$ that represents distance or travel time.

A path $P(v_1, v_n) = \{v_1, v_2, ..., v_n\}$ is a sequence of vertices such that v_i is adjacent to v_{i+1}, i.e., $(v_i, v_{i+1}) \in E$, for $1 \leq i < n$. The cost of a path P, denoted by $cost(P)$, is the sum of weights of the edges of P. Given vertices u and v, we use $\delta(u, v)$ to denote the shortest path from u to v while we use $dist(u, v)$ to denote the cost of $\delta(u, v)$. Figure 3.2 shows an example of a road network. If given a source vertex v_1 and destination vertex v_4, then $\delta(v_1, v_4) = \{v_1, v_2, v_5, v_4\}$ is the shortest path between v_1 and v_4 and $dist(v_1, v_4) = 6$.

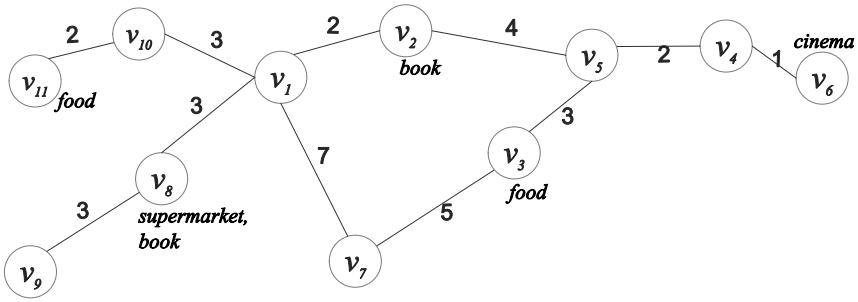


Figure 3.2: Example of a road network

3.2.2 Data Model

A spatio-textual object o is an object with a spatial location from $S = \{s_1, s_2, ..., s_m\}$ that contains a set of keywords from $T = \{t_1, t_2, ..., t_x\}$. We assume that spatio-textual objects are located at vertices in V. The weight of an edge $(u, v) \in E$ is the travel time or road network distance of two spatially-adjacent objects o_1 and o_2 representing u and v, respectively. We use vertex v or spatio-textual object o interchangeably in this chapter.

3.2.3 Query Model

Given a road network G, the user queries consist of a source location s_l, a destination location d_l, and preferred set of keywords $K = \{k_1, k_2, ..., k_n\}$, where each k_i, $1 \leq i \leq n$, can be positive (k^+) or negative (k^-). A positive keyword k^+ means that the keyword satisfies what the user wants, while a negative keyword k^- means that part of the keyword expresses negative connotation which the user wants to avoid. We assume $K \subseteq T$. Having all this information, we want to find the Best Path $BP(s_l, d_l, K)$ that establishes a shortest path from s_l to d_l, and that passes through all k^+ and avoids all k^- keyword matching vertices in G.

Table 3.1 presents the list of mathematical notations used in this chapter.

Table 3.1: List of notations used throughout the chapter

Notation	**Definition**
G	Road network
$\delta(u, v)$	Shortest path between u and v
$dist(u, v)$	Distance between u and v
s_l, d_l	Source location, destination location
K	Query keywords given by user
k^+, k^-	Positive keyword, negative keyword
$BP(s_l, d_l, K)$	The Best Path from s_l to d_l that passes through all k^+ and avoids k^-

3.3 Complexity Analysis

The Best Path problem is different from the general Shortest Path problem. In Best Path problem, we want to find the path with objects of interest along our way to the destination. The objects of interest are determined by the keywords. The result of Best Path itself often is longer in distance than the Shortest Path, but there are also cases where it can have the same result as the Shortest Path. When the user does not specify any keywords at all in the query, then the Best Path is basically the Shortest Path since there are only source and destination locations provided. The Shortest Path problem is solvable in polynomial time. Therefore we can implement the commonly used algorithms for Shortest Path, such as Dijkstra algorithm, for this particular case.

Lemma 1. *Given s_l, d_l, and $K = \{\}$, $BP(s_l, d_l, K) = \delta(s_l, d_l)$.*

Proof. Base Case: If $|P| = 1$ and $K = \{\}$, then $P = \{s_l\}$ and $cost(P) = 0 = dist(s_l, s_l)$. Hence, $\delta(s_l, s_l) = BP(s_l, s_l, K)$.

Inductive hypothesis: Let u be the last vertex added to P, $P' = P \cup \{u\}$. In this case our Inductive Hypothesis is

$$\text{for each } y \in P',\ cost(P'(s_l, y)) = cost(\delta(s_l, y))$$

Inductive step: Suppose that there is a shortest path Q from s_l to u and

$$cost(Q) < cost(P'(s_l, u))$$

Since Q is a shortest path, then $cost(Q) = dist(s_l, u)$.
Assume that the shortest path Q begins at P' and then leaves P' before arriving to the destination u. (y, z) is the first edge in Q that leaves P', and Q_y is a shortest path from s_l to y, so

$$cost(Q_y) + w(y, z) \leq cost(Q)$$

Since according to the Inductive Hypothesis $cost(P'(s_l, y))$ is also the cost of $\delta(s_l, y)$, then $cost(P'(s_l, y)) \leq cost(Q_y)$. So it gives us

$$cost(P'(s_l, y)) + w(y, z) \leq cost(Q_z)$$

As y and z are adjacent vertices, then

$$cost(P'(s_l, z)) \leq cost(P'(s_l, y)) + w(y, z)$$

Since u is part of Q, so

$$cost(P'(s_l, u)) \leq cost(P'(s_l, z))$$

Therefore shortest path Q does not exist, so $cost(P'(s_l, u)) = \delta(s_l, u) = BP(s_l, u, K)$.

□

However, this is not the same when we have a keyword specified by the user. Depending on the positive or negative value, the path may or may not be retrieved. If the query contains only one positive keyword, doing the shortest path search from source to the

nearest vertex that has matching keyword then doing another shortest path search from the nearest vertex with matching keyword to the destination is incorrect. As an example using the road network in Figure 3.1, assume that the source s_l is v_5, destination d_l is v_{10}, and then the preferred keyword given by the user is located at v_1 and v_3. If we choose the nearest keyword match vertex from v_5, v_3 is the nearest since $dist(v_5, v_3)$ is 3, while $dist(v_5, v_1)$ is 6. However if we calculate the total distance from v_5 to v_3 to v_{10}, the total is 18. On the contrary, the total distance from v_5 to v_1 to v_{10} is 9, which is a lot shorter than having v_3 as the chosen keyword match vertex. Hence, choosing the nearest vertex with matching keyword will cause local minimum problem.

Meanwhile if the query keyword does not exist in T, then $BP(s_l, d_l, K)$ is also $\delta(s_l, d_l)$.

Lemma 2. *Given s_l, d_l, and $K = \{k_1\}$, $k_1 \notin T$. Thus $BP(s_l, d_l, K) = \delta(s_l, d_l)$.*

Proof. If $k_1 \notin T$, then $K = \{\}$; which is already proven in Lemma 1. □

If the user query contains a negative keyword, then the vertices that have this particular keyword need to be avoided/blocked. When we do the query processing to find the Best Path, these vertices can be pruned/disconnected from the graph as they are no longer considered as POI. This may also cause a dead-end in the graph since a potential path can be ceased with the disappearance of a vertex. So when an edge of a vertex with negative keyword is a bridge, we will not be able to retrieve any Best Path. Lemma 3 and 4 prove the non-existence of Best Path for this particular case.

Lemma 3. *If there exists a bridge (u, v) in G, and G consists of subgraph H and I that are connected by (u, v). Given $s_l \in H$ and $d_l \in I$, then $(u, v) \subseteq BP(s_l, d_l, K)$.*

Proof. Assume that (u, v) is a bridge in G and $BP(s_l, d_l, K)$ on G does not contain (u, v). Since $BP(s_l, d_l, K)$ is a path that every vertex in it has to be connected with each other, and $BP(s_l, d_l, K) \subseteq G \setminus \{(u, v)\}$ which $G \setminus \{(u, v)\}$ is a disconnected graph, then it must be disconnected. □

Definition 3. *A critical path $cp(v_i, v_j)$, $i \neq j$, is a path that consists of one or more graph bridges between v_i and v_j.*

Lemma 4. *Given s_l, d_l, $K = \{k^-\}$, and $BP(s_l, d_l, K) = cp(s_l, d_l)$. Then $BP(s_l, d_l, K)$ does not exist.*

Proof. Suppose that there exists a bridge (u, v) in $BP(s_l, d_l, K)$ and $k^- \in (u, v)$. Since we have to avoid negative keywords, then (u, v) has to be pruned from $BP(s_l, d_l, K)$. Hence, the graph is now disconnected as proved in Lemma 3. □

Even though negative keywords can cause path blockage, it does not mean that we cannot retrieve any Best Path at all. The path blockage might cause us to re-route to another path even though it may cause a longer path.

Lemma 5. *Given d_l, s_l, and $K = \{k^-\}$, if $BP(s_l, d_l, K) \neq cp(s_l, d_l)$, then there exists $BP(s_l, d_l, K)$.*

Proof. We can establish a path since the graph is still connected even though there is k^-. □

In the case where the user gave a set of positive keywords as the query input and there is no negative keyword at all, then the Best Path's result will be similar to the state-of-the-art Trip Planning Route Queries (TPQ)'s result [20].

Lemma 6. *Given d_l, s_l, and $K = \{k_1^+, k_2^+, ..., k_n^+\}$, $BP(s_l, d_l, K) = TPQ$.*

Proof. When all keywords are positive, then by definition, Best Path is the same as TPQ where we have to find the best trip/route from s_l, passing through one point from each category and then ending the trip at d_l. □

Looking at Lemma 6, it means that Best Path also can be considered as NP-hard problem. Thus in the following Lemma we try to reduce Best Path problem to Traveling Salesman Problem (TSP), to which TSP is a well-known NP-hard problem.

Lemma 7. *$BP(s_l, d_l, K)$ is NP-hard.*

Proof. Assume a road network G of a set of spatio-textual objects with spatial locations $S = \{s_1, s_2, ..., s_m\}$ and each object has a distinct keyword from the keyword set T. Moreover, the user queries consist of a source location s_l, destination location d_l, and preferred keywords $K = \{k_1, k_2, ..., k_n\}$. Again, assume that for K, we need to visit every vertex in G. Now, we reduce the Best Path Problem to TSP. Let $G' = (V', E')$ as the instance of TSP, where $V' = V$ and $E' = (u, v)$ for any $u, v \in V'$. Then for road network G, we complete the graph by connecting all vertices. The cost function between G and G' is as follow:

$$cost(u,v) = \begin{cases} 0, & \text{if } edge(u,v) \in E \\ 1, & \text{if } edge(u,v) \notin E \end{cases}$$

Suppose that Best Path $BP(s_l, d_l, K)$ exists in G and has cost ≤ 0 in G', hence there exists a solution to TSP in G' with cost ≤ 0. □

3.4 Data Index

This section presents the *IG-Tree*, an indexing technique for planning Best Path Queries on Road Networks. Before presenting the *IG-Tree*, we first provide a brief background on *G-Tree* [68, 69] and IR^2-*Tree*[1], which are the indexing techniques that inspired us to develop the *IG-Tree* for processing $BP(s_l, d_l, K)$ queries efficiently.

3.4.1 G-Tree

One of the most efficient indexing techniques on Road Networks is the *G-Tree* [68, 69]. In *G-Tree*, the road network is partitioned recursively into sub-networks. The nodes in *G-Tree* correspond to a single sub-network and each node contains two or more road network vertices. The graph partition process is performed by using the multi-level partitioning algorithm [122], which guarantees that each subgraph will be of almost the same size. Figure 3.3 shows an example of graph partitioning of the road network given Figure 3.2. Here, the original graph is partitioned into two subgraphs, which are shown by G_1 and G_2. In the next level, G_1 is partitioned again into two equal-sized subgraphs G_3 and G_4. Similarly, G_2 is partitioned again into subgraphs G_5 and G_6.

The vertices that connect two sub-networks together are marked as *borders* and they are stored in the *G-Tree* nodes. Figure 3.3 shows the example where vertex v_1 is the border of G_3 since it connects partition G_3 with other partitions G_4 and G_5. The border in partition G_4 consists of v_8 and v_7, the border in partition G_5 consists of v_2 and v_3, and the border in partition G_6 is v_4. In the partition G_1, the borders are v_1 and v_7 as both connects G_1 with G_2. While the borders in G_2 are v_2 and v_3.

G-Tree does not store the distance of every vertex but stores the set of borders and the shortest path distance between borders that is kept in the distance matrix. For example for partition G_3, the border is v_1, so the distance matrix will contain the shortest path from v_1 to all other vertices in the subgraph partition $\{v_1, v_{10}, v_{11}\}$. The distance matrix

itself is proven to be very efficient in terms of processing the kNN search on road networks [68, 69].

Though *G-Tree* is very efficient in indexing and processing nearest neighbor (NN), k nearest neighbor (kNN) and keyword-based kNN queries on road networks, it is not applicable for processing Best Path queries.

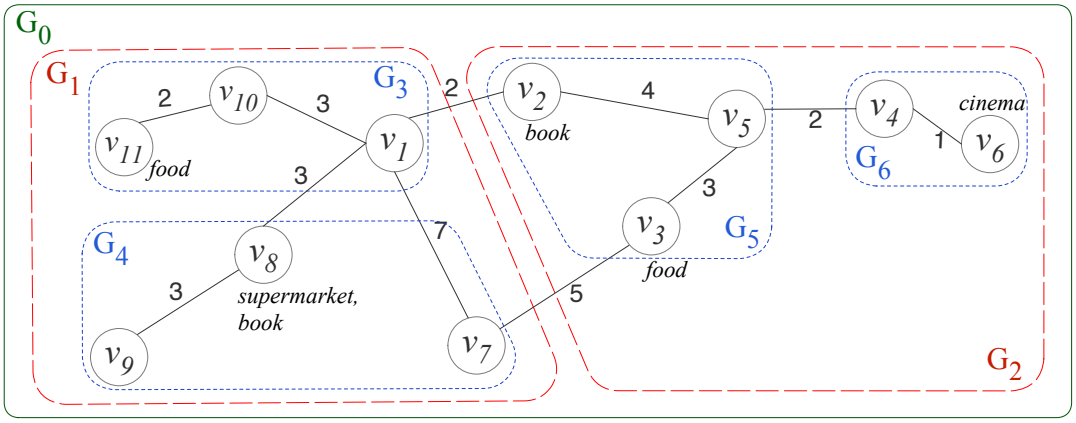


Figure 3.3: Graph partitioning on road network given in Figure 3.2

3.4.2 IR^2-Tree

There are a number of indexing techniques proposed for processing *Spatial Keywords Queries*, one of them is the IR^2-*Tree*. The IR^2-*Tree* is first introduced by Felipe *et al.* [1]. It is a hybrid indexing approach that combines the *R-Tree* [123] and information retrieval signature files. However, this indexing technique is only applicable for spatial data objects in Euclidean space.

The indexing in IR^2-*Tree* is performed by attaching the inverted index to the *R-Tree*, i.e., every tree node in IR^2-*Tree* holds the information for both spatial location and keywords. The leaf nodes contain the actual spatial data and keywords. For example, assume that an object o_1 that contains keyword *book* is located in leaf node N_1 with spatial location of $\left[38, 4\right]\left[93, 9\right]$ (upper right and bottom left coordinate of the *minimum bounding rectangle* (MBR)), while an object o_2 with keyword *supermarket* is located in leaf node N_2 with spatial location of $\left[8, 15\right]\left[41, 32\right]$. Suppose that the inverted index for keyword *book* is 10 and the inverted index for keyword *supermarket* is 01. Thus the leaf node N_1 contains the information of spatial location $\left[38, 4\right]\left[93, 9\right]$ and keyword index 10, while leaf node N_2 contains the information of spatial location $\left[8, 15\right]\left[41, 32\right]$ and keyword index 01.

As the leaf node in IR^2-*Tree* stores the spatial data and keyword index, the non-leaf node contains the combination of several objects. The spatial information is based on the MBR, while the inverted index of the keywords is calculated using logical OR [1]. For example, the leaf nodes N_1 and N_2 from the previous example have the same parent node N_0. So in this case, N_0 contains keyword information of 11 as this node consists of both keywords from N_1 and N_2.

3.4.3 Proposed Data Index: IG-Tree

As previously discussed, the IR^2-*Tree* [1] is used for indexing Spatial Keywords Queries in Euclidean space, while the *G-Tree* [68, 69] is used for indexing Road Networks. As Best Path is a type of Spatial Keywords Query on Road Network, each one of these indexing techniques has its own benefit to Best Path Query. Thus, we adopt these two indexing techniques to develop a new indexing scheme that can improve the processing of Best Path query: *IG-Tree*, a hybrid between IR^2-*Tree* and *G-Tree*.

Using the road network in Figure 3.2, we attempt to create the *IG-Tree*. So following the graph partition technique used in *G-Tree*, we partition the graph into smaller subgraphs. Figure 3.3 shows the graph partitioning of the example road network given in Figure 3.2. The graph is divided into equal-sized subgraphs using the multi-level partitioning algorithm [122] and each partition consists of two or more vertices. At the leaf level of the tree, the subgraph G_3 consists of vertices v_1, v_{10} and v_{11}; subgraph G_4 consists of vertices v_7, v_8 and v_9; subgraph G_5 consists of vertices v_2, v_3 and v_5; and finally, subgraph G_6 consists of vertices v_4 and v_6. Each partition makes up one node in the *IG-Tree*, as presented in Figure 3.4.

After the graph partition, we mark the borders of each partition. Borders are the vertices in one partition that are connecting the road network to another partition. For example the border for partition G_3 is v_1 since v_1 connects the subgraph to partition G_4 and partition G_5. So the borders of G_4 are v_7 and v_8; the borders of G_5 are v_2, v_3 and v_5; while the border of G_6 is v_4. Based on these borders, we create the distance matrices. So the shortest path distances for every border in every node are pre-computed and stored in the matrices. Table 3.2 - Table 3.8 show the distance matrices for each node in *IG-Tree*.

Another aspect of the graph in Figure 3.3 is the keywords. Some vertices contain one or more keywords, thus we also need to index these keywords. The keywords can be turned

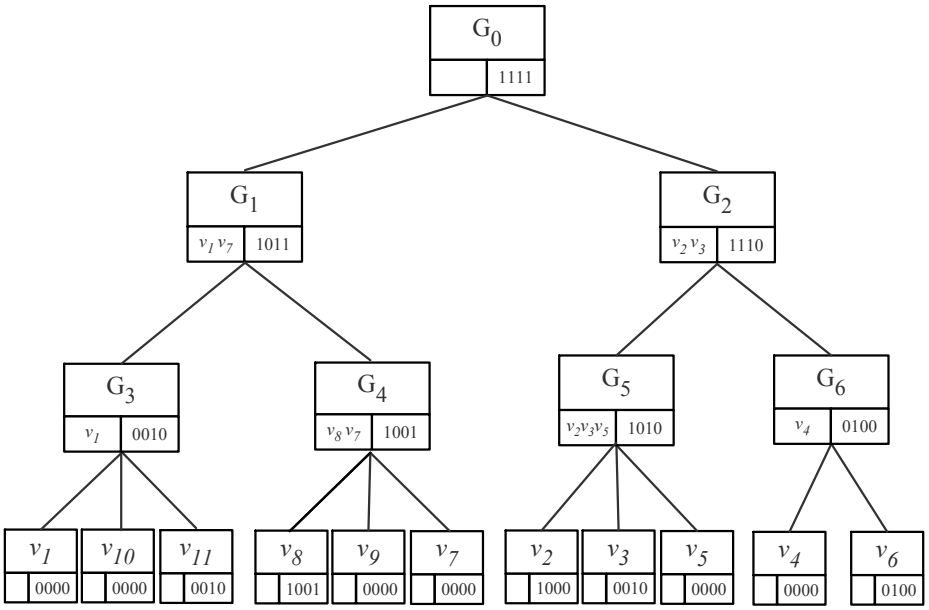


Figure 3.4: IG-Tree

Table 3.2: Distance Matrix for G_0

G_0	v_1	v_7	v_2	v_3
v_1	0	7	2	9
v_7	7	0	9	5
v_2	2	9	0	7
v_3	9	5	7	0

Table 3.3: Distance Matrix for G_1

G_1	v_1	v_8	v_7
v_1	0	3	7
v_8	3	0	10
v_7	7	10	0

Table 3.4: Distance Matrix for G_2

G_2	v_2	v_3	v_5	v_4
v_2	0	7	4	6
v_3	7	0	3	5
v_5	4	3	0	2
v_4	6	5	2	0

Table 3.5: Distance Matrix for G_3

G_3	v_1	v_{10}	v_{11}
v_1	0	3	5

Table 3.6: Distance Matrix for G_4

G_4	v_8	v_9	v_7
v_8	0	3	10
v_9	3	0	13
v_7	10	13	0

Table 3.7: Distance Matrix for G_5

G_5	v_2	v_3	v_5
v_2	0	7	4
v_3	7	0	3
v_5	4	3	0

Table 3.8: Distance Matrix for G_6

G_6	v_4	v_6
v_4	0	1

into inverted list. So the first step is to sort all of the keywords in the graph. Then for each keyword, we assign a binary value based on its existence in each node. For instance vertex v_2 contains only keyword *book*, thus the inverted list for v_2 is 1000. For node v_8, it contains both keyword *book* and *supermarket*, so its inverted list is 1001. The inverted index for all the vertices of the graph in Figure 3.3 is presented in Table 3.9.

Based on the above inverted index list, we attach each inverted index to its corresponding vertices at the leaf nodes. For each parent node, its inverted index is calculated using logical OR of its child nodes. For instance G_3's inverted index is the result of logical OR of the inverted index of v_1, v_{10}, v_{11}. The result of 0000 *or* 0000 *or* 0010 is 0010, thus

Table 3.9: Keyword index

Keyword	*Book*	*Cinema*	*Food*	*Supermarket*
v_1	0	0	0	0
v_2	1	0	0	0
v_3	0	0	1	0
v_4	0	0	0	0
v_5	0	0	0	0
v_6	0	1	0	0
v_7	0	0	0	0
v_8	1	0	0	1
v_9	0	0	0	0
v_{10}	0	0	0	0
v_{11}	0	0	1	0

Table 3.10: Keyword Distance Matrix for G_0

G_0	1000	0100	0010	0001
v_1	2	9	5	3
v_7	9	11	5	10
v_2	0	7	7	5
v_3	7	6	0	11

Table 3.11: Keyword Distance Matrix for G_1

G_1	1000	0100	0010	0001
v_1	2	Ø	5	3
v_8	0	Ø	8	0
v_7	10	Ø	12	10

Table 3.12: Keyword Distance Matrix for G_2

G_2	1000	0100	0010	0001
v_2	0	7	7	Ø
v_3	7	6	0	Ø
v_5	4	3	3	Ø
v_4	6	1	5	Ø

Table 3.13: Keyword Distance Matrix for G_3

G_3	1000	0100	0010	0001
v_1	Ø	Ø	5	Ø

Table 3.14: Keyword Distance Matrix for G_4

G_4	1000	0100	0010	0001
v_8	0	Ø	Ø	0
v_9	3	Ø	Ø	3
v_7	10	Ø	Ø	10

Table 3.15: Keyword Distance Matrix for G_5

G_5	1000	0100	0010	0001
v_2	0	Ø	7	Ø
v_3	7	Ø	0	Ø
v_5	4	Ø	3	Ø

Table 3.16: Keyword Distance Matrix for G_6

G_6	1000	0100	0010	0001
v_4	Ø	1	Ø	Ø

G_3's inverted index is 0010. The same calculation is applied for every non-leaf node. The root node will normally have all 1s for the index.

Even though we have indexed all of the available keywords and assign them to each node in the tree, having these indexes are not adequate. The inverted index only identifies that a certain keyword exists on a node but do not exactly identify the location until we go to the leaf node. Therefore we propose a Keyword Distance Matrix for each node. This *Keyword Distance Matrix* contains the distance of the nearest keyword matching vertex from each border. By having this matrix, the keyword search computation is sped up as we do not need to compute the keyword distance in processing time. The Keyword Distance Matrices for the *IG-Tree* in Figure 3.4 are shown at Table 3.10 - Table 3.16.

Based on the above discussion, there are several important components to build an *IG-Tree*. For every non-leaf node in *IG-Tree* contains the partition name, the border of

each partition, and the inverted index (using the logical OR of its child node). Each non-leaf node also contains two types of matrices, which are the Distance Matrix and Keyword Distance Matrix. For every leaf node, it contains the road network's vertex and inverted list of the corresponding vertex. We also keep the geographic coordinate location of each vertex in the leaf node.

Space Complexity of the IG-Tree

Height. The height of *IG-Tree* is similar to *G-Tree*[68, 69] which is $\mathcal{H} = \log_f \frac{|V|}{\tau} + 1$, where f is the number of partitions for each graph/subgraph, $|V|$ is the number of vertices in the given (road network) graph G, and τ is the number of maximum vertices on leaf node's subgraph.

Number of Nodes. Like *G-Tree*, *IG-Tree* has only one node in level 0, which is the root. In an arbitrary level i of the tree, there are f^i internal nodes as the number of partitions for each graph (at level 0)/subgraph (at level > 0) is s. As τ is the maximum number of vertices on leaf node's subgraph, there are $\frac{|V|}{\tau}$ leaf nodes. As a result, the number of nodes in *IG-Tree* is $\mathcal{O}(\frac{f}{f-1} \cdot \frac{|V|}{\tau}) = \mathcal{O}(\frac{|V|}{\tau})$ which is again similar to that of *G-Tree*.

Number of Inverted Lists. A node in *IG-Tree* contains an inverted list representing the keywords covered in that node. As the number of nodes in an *IG-Tree* is $\mathcal{O}(\frac{|V|}{\tau})$, the number of inverted lists is $\mathcal{O}(\frac{|V|}{\tau}+|V|)$. Thus, the space complexity of maintaining inverted lists in *IG-Tree* becomes $\mathcal{O}(\frac{|V|}{\tau}\cdot|T|+|V|\cdot|T|)$, where $|T|$ is the number of keywords covered in the whole road network G and $|V| \cdot |T|$ is the space complexity of the inverted lists for all vertices in V.

Number of Borders. If we assume the road network to be modeled as a planar graph, the number of borders on average in a node of level i is $\mathcal{O}(\log_2 f \cdot \sqrt{\frac{|V|}{f^{i+1}}})$ as per the calculation conducted in [68, 69]. As there are f^i nodes in a level i, the number of border nodes in an arbitrary level i is $\mathcal{O}(\log_2 f \cdot \sqrt{\frac{|V|}{f^{i-1}}})$. If we sum this measure from level 1 to height of the tree $\log_f \frac{|V|}{\tau} + 1$, the total number of borders in an *IG-Tree* is $\mathcal{O}(\frac{\log_2 f}{\sqrt{\tau}}|V|)$, which is again similar to the *G-Tree* under the planar graph assumption.

Distance Matrices. The total distance matrix size of all leaf nodes is $\mathcal{O}(\sqrt{\tau}|V| \cdot \log_2 f)$ and the total distance-matrix of non-leaf nodes is $\mathcal{O}(|V| \cdot \log_2^2 f \cdot \log_f \frac{|V|}{\tau})$ as per the calculation conducted in [69, 68].

Keyword Distance Matrices. The average number of borders in a leaf node of *IG-Tree* is $\mathcal{O}(\log_2 f \cdot \sqrt{\tau})$[68, 69] and the total number of keywords in G is $|T|$. Thus the keyword distance matrix size in a leaf node is $\mathcal{O}(\log_2 f \cdot \sqrt{\tau} \cdot |T|)$. The total keyword distance matrix size of all leaf nodes becomes $\mathcal{O}(\frac{|V|}{\tau} \cdot \log_2 f \cdot \sqrt{\tau} \cdot |T|)$. Each internal node on level i generates $\mathcal{O}(\log_2 f \cdot \sqrt{\frac{|V|}{f^{i+1}}})$ borders on average [69, 68]. Therefore, the keyword distance matrix size of each node at level i is $\mathcal{O}(\log_2 f \cdot \sqrt{\frac{|V|}{f^{i+1}}} \cdot |T|)$. In *IG-Tree*, there are f^i nodes at level i, therefore keyword distance matrix size at level i is $\mathcal{O}(f^i \cdot \log_2 f \cdot \sqrt{\frac{|V|}{f^{i+1}}} \cdot |T|)$=$\mathcal{O}(\log_2 f \cdot \sqrt{|V| f^{i-1}} \cdot |T|)$. Thus the total keyword distance matrix size of non-leaf nodes is $\mathcal{O}(\sum_{0 \leq i < \mathcal{H}} \log_2 f \cdot \sqrt{|V| f^{i-1}} \cdot |T|)$.

Index Reconstruction for Tree Node with Negative Query Keywords.

In the Best Path Query, the user is allowed to give keywords as input of the query and the query keywords can be positive and negative. The positive keywords denote the spatio-textual objects that the user wants to visit, while the negative keywords denote the spatio-textual objects that the user wants to avoid along his/her trip. Even though *IG-Tree* contains textual information of spatial objects, we still have to check the textual relevancy between the spatial object with the query keywords given by the user in order to consider an object to be visited or avoided. For the objects that contain positive keywords, the *IG-Tree* can compute the path well with the help of the Distance Matrices. For example if we want to compute the path from v_5 to the nearest *book*, we can directly refer to the Keyword Distance Matrix in Table 3.15 to save up some time. But this is different when negative keywords exist. Even though *IG-Tree* is designed to improve path computation on finding the Best Path, it still has a weakness when there is a negative keyword found in the query given by the user. As previously mentioned in Section 3.3, a vertex that holds one or more negative query keywords must be pruned from the road network graph. This is due to the fact that this particular vertex holds a query that the user wants to avoid/block. Currently *IG-Tree* consists of Distance Matrices that store the shortest paths between borders. So when a vertex is pruned from the graph, the shortest path may also change. The existing index has to be modified considering a vertex is gone and the Distance Matrices are no longer storing accurate distances. The new Distance Matrices will replace the existing matrices during the query processing time of the query

with the corresponding negative keywords. The modification however depends on the location of the vertex in the *IG-Tree*:

- **Case-C1.** If vertex v that contains k^- is the border and no other border exists for a node that we must visit, then no path can be established. In this case, v is considered as a bridge. This situation is already proved through Lemmas 3 and 4 in Section 3.3.

- **Case-C2.** If vertex v that contains k^- is the border and there is/are other border(s), then path reconstruction is needed. The path reconstruction will involve the whole tree node where v is located and also the borders on other nodes that are adjacent to v. For example if v_3 in Figure 3.3 contains k^-, then the path reconstruction occurs on the whole G_5 tree node and its adjacent borders. The adjacent borders in this case can be identified through parent nodes of G_5, whether the parent nodes have v_3 as one of their borders. G_2 and G_0 are indeed sharing v_3 as their border, so the path reconstruction will involve these two tree nodes as well. As v_3 is pruned from the graph, v_3 is then omitted from the Distance Matrices of G_5, G_2 and G_0. The border-to-border distances of these matrices are also affected because of the omission of v_3, thus the entire matrices have to be recalculated because of the changes in the shortest path between these borders. The path reconstruction itself can be obtained using Dijkstra algorithm. Table 3.17 - Table 3.19 shows the Distance Matrices after path reconstruction.

- **Case-C3.** If vertex v that contains k^- is in the leaf node (not the border), then distance has to be recalculated. The index recalculation for this case does not affect the whole tree, but only on the tree node where the vertex with negative keyword lies. Similar to the previous case, the path reconstruction can be obtained using Dijkstra algorithm. When v is in the leaf node, we can focus on its own subgraph partition as it does not affect the other partitions like in the previous case. For instance, assume that v_{10} in Figure 3.3 contains k^-. v_{10} is a leaf node as it does not connect any sub-networks. v_{10} is located in partition G_3, thus only this partition will need to be reconstructed which is shown in Table 3.20.

Table 3.17: Reconstructed Distance Matrix for G_5

G_5	v_2	v_5
v_2	0	4
v_5	4	0

Table 3.18: Reconstructed Distance Matrix for G_2

G_2	v_2	v_5	v_4
v_2	0	4	6
v_5	4	0	2
v_4	6	2	0

Table 3.19: Reconstructed Distance Matrix for G_0

G_0	v_1	v_7	v_2
v_1	0	7	2
v_7	7	0	9
v_2	2	9	0

Table 3.20: Reconstructed Distance Matrix for G_3

G_3	v_1	v_{11}
v_1	0	Ø

3.5 Query Processing

We propose three Best Path query processing algorithms that can be applied on *IG-Tree*, namely the Optimal Distance Approximation Search, Ancestry Priority Search, and the Euclidean-based Approximation. A baseline algorithm is also provided in this section. The baseline algorithm offers precise solution, while the other three proposed algorithms offer approximation solution with different trade-offs. In each subsection, we discuss on how each algorithm works and their trade-offs.

3.5.1 Baseline Algorithm

In this section, we discuss on the baseline algorithm that can be used on *IG-Tree* to find the Best Path. This algorithm is able to compute the result of Best Path query accurately. The key/main idea is to find the permutation of all possible combinations of positive keywords and then compare them in order to find the one that has the most efficient cost (least distance).

As an example, assume that we want to find the best path from v_1 to v_4 while passing through *cinema* and *book*. In this case, $sl = v_1$, $dl = v_4$, and $keywords = \{cinema, book\}$. The first step here is to turn the preferred keywords into inverted index so that we can check its relevancy with the inverted index in *IG-Tree*. The preferred keywords are *cinema* and *book*, therefore the inverted list is 1100 ($K = 1100$). Then we have to find the partition that contains the source and destination in the *IG-Tree*, where v_1 is located under the partition G_3 and v_4 is located under the partition G_6. After the source and destination

Algorithm 1 Shortest Path Search $\delta(s_l, d_l)$ on IG-Tree

Input:$IG-Tree$, s_l, d_l, $inverted\ file\ K_{if}$ in
Output:$\delta(s_l, d_l)$ out
1: Find partition G_{s_l} that contains s_l on $IG-Tree$
2: Find partition G_{d_l} that contains d_l on $IG-Tree$
3: Current source node $v_{s_l} = s_l$
4: Current source node $v_{d_l} = d_l$
5: **while** G_{s_l} != G_{d_l} **do**
6: Find nearest border b_{s_l} from v_{s_l}
7: **if** b_{s_l} contains k^- **then**
8: **if** There is other border **then**
9: Path reconstruction for partition G_{s_l}
10: **else**
11: return -1
12: **end if**
13: **end if**
14: $dist(d_l, b_{s_l}) += dist(v_{s_l}, b_{s_l})$
15: $v_{s_l} = b_{s_l}$
16: Find nearest border b_{d_l} from v_{d_l}
17: **if** b_{d_l} contains k^- **then**
18: **if** There is other border **then**
19: Path reconstruction for partition G_{d_l}
20: **else**
21: return -1
22: **end if**
23: **end if**
24: $dist(d_l, b_{d_l}) += dist(v_{d_l}, b_{d_l})$
25: $v_{d_l} = b_{d_l}$
26: G_{s_l} = parent node of v_{s_l}
27: G_{d_l} = parent node of v_{d_l}
28: **end while**
29: **if** $G_{s_l} == G_{d_l}$ **then**
30: Check Distance Matrix to get the shortest path $dist(v_{s_l}, v_{d_l})$ from v_{s_l} to v_{d_l}
31: $dist(s_l, d_l) = dist(s_l, b_{s_l}) + dist(d_l, b_{d_l}) + dist(v_{s_l}, v_{d_l})$
32: return $\delta(s_l, d_l)$
33: **end if**

locations are found, then we can start finding the best path $BP(v_1, v_4, 1100)$ that visits the chosen keywords.

Scanning through every single vertex in the leaf node that holds inverted index of 1100. The inverted index of 1000 can be found at v_2 and v_8, while the inverted index of 0100 can be found at v_6. Knowing the exact locations of the keywords, we can do cartesian product between each set of keywords. In this case, the cartesian product will be between $\{v_2, v_6\}$ and $\{v_8, v_6\}$. Then based on the cartesian product, we have to get the permutation to help computing the path with the least distance. The permutations for this case consist of $\{v_2, v_6\}$, $\{v_6, v_2\}$, $\{v_8, v_6\}$, and $\{v_6, v_8\}$. Based on these permutations, we can find the shortest path from s_l to each permutation, and then from the permutation to d_l. In this case, we will have four possible paths: $v_1 \rightarrow v_2 \rightarrow v_6 \rightarrow v_4 = 10$, $v_1 \rightarrow v_6 \rightarrow v_2 \rightarrow v_4 = 22$,

Algorithm 2 The Baseline

Input:$IG-Tree$, s_l, d_l, K in
Output:$BP(s_l, d_l, K)$ out
1: Index K to inverted file K_{if}
2: Find s_l on $IG-Tree$
3: Find d_l on $IG-Tree$
4: Scan through the leaf nodes to find vertices V_k with keywords that match to k^+ in K_{if}.
5: Find Cartesian Product $cart\_product(V_k)$
6: **for** each $cart\_product(V_k)$ result cp **do**
7: Find possible permutations of cp $Permutation(cp)$
8: **for** each permutation $perm$ of $Permutation(cp)$ **do**
9: $v_{start} \leftarrow 0$
10: **for** each vetex v in $perm$ **do**
11: **if** $v_{start} = 0$ **then**
12: Find $\delta(s_l, v)$;
13: **else**
14: Find $\delta(v_{start}, v)$;
15: **end if**
16: $v_{start} = v$
17: **end for**
18: Find $\delta(v_{start}, d_l)$;
19: **if** current path is the shortest **then**
20: Best Path $BP(s_l, d_l, K)$
21: **end if**
22: **end for**
23: **end for**
24: return $BP(s_l, d_l, K)$;

$v_1 \rightarrow v_8 \rightarrow v_6 \rightarrow v_4 = 16$, $v_1 \rightarrow v_6 \rightarrow v_8 \rightarrow v_4 = 32$. Algorithm 1 shows the pseudocode for shortest path search in *IG-Tree*. While calculating the shortest paths, we also need to keep track of the path with the least sum of distance. At the end, we will obtain the Best Path with the most accurate solution. For this example, the Best Path $BP(v_1, v_4, 1100) = v_1 \rightarrow v_2 \rightarrow v_6 \rightarrow v_4$ with the least total distance of 10.

This algorithm guarantees the accuracy of finding the Best Path on road networks. However since the Best Path query is an NP-Hard problem, this algorithm definitely runs in non-polynomial time especially on a large datasets. In our experiment, it can spend up to 17 hours merely to find the Best Path with 5 query keywords even in a very small datasets with only 100 vertices. This is certainly impossible to be applied for our daily use. The pseudo-code of the baseline algorithm is given in Algorithm 2.

3.5.2 Optimal Distance Approximation Search

Because of the non-polynomial time complexity of the baseline algorithm, we propose an approximation algorithm to compromise the runtime. This algorithm is a lot faster than the baseline one but its result is not 100% accurate.

When multiple keywords are involved in the query, the complexity rises as we have to know all possible combinations of the keywords in order to get the most optimal solution (distance-wise). However when there is only one keyword involved in the query, the query can be retrieved in polynomial time. Thus in this approximation algorithm we utilize this situation in order to retrieve the multiple keywords query. The way this algorithm works is that for each query keyword given by the user, we find the best path between the source location to the query keyword and then to the destination. By getting the best path for each keyword, we can locate the best possible location of each keyword that will give the shortest distance of source-keyword-destination. Then after we have the best candidate of each keyword, we find the path from source location to its nearest candidate, then from the nearest candidate to its next nearest candidate. We keep doing this until all keywords are covered, then finishing the path to the destination location.

For example a user wants to find the best path from v_{10} to v_7 while passing through a bookstore and a cinema. In this case, $s_l = v_{10}$, $d_l = v_7$, and $K = \{book, cinema\}$. In order to find the best path, we have to transform the query keywords given by the user into inverted list. Since the keywords are $\{book, cinema\}$, thus the inverted list is $K_{if} = 1100$.

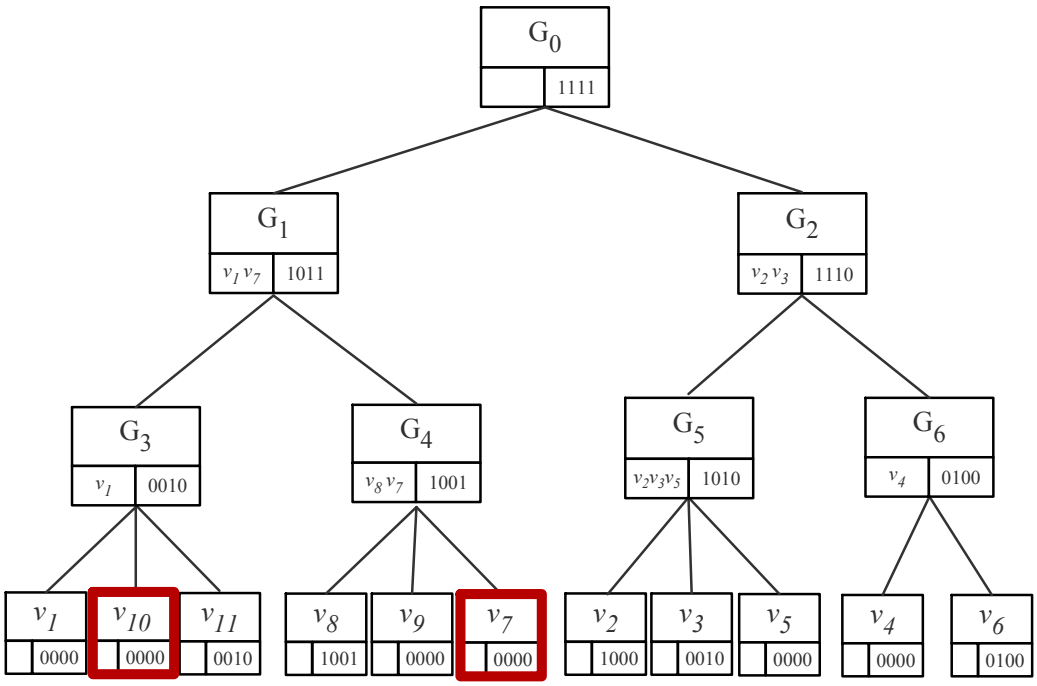


Figure 3.5: Finding location of v_{10} and v_7

In this algorithm, we have to firstly find the locations of s_l and d_l. Looking at the *IG-Tree*, s_l is located within partition G_3, while d_l is located within partition G_4 as shown

in Figure 3.5. Now for each keyword k_n in K_{if}, we have to find the best path from $s_l - k_n - d_l$. Assume that the first keyword that we want to find its best path is *book* to which its inverted index is 1000. In the road network, there are actually several vertices that contain the keyword *book*. So we have to calculate the total shortest path distance for each keyword location and then finding the one that has the least amount of distance. A naive solution here is to find the shortest path between s_l to k_n and then add up the shortest path between d_l to k_n. Since our current keyword index is 1000, v_8 and v_2 have the same index. Hence we have to establish the shortest path $\delta(v_{10}, v_8)+\delta(v_8, v_7)$ and also $\delta(v_{10}, v_2) + \delta(v_8, v_2)$. The way the shortest path works is similar to the previous section. The best path distance for visiting v_8 is $dist(v_{10}, v_8) + dist(v_8, v_7) = 6 + 10 = 16$, while the best path distance for visiting v_2 is $dist(v_{10}, v_2) + dist(v_2, v_7) = 5 + 9 = 14$. Based on these calculations, v_2 has the best path from v_{10} to v_7 as depicted in Figure 3.6 and Figure 3.7. Thus, we can store v_2 to a candidate queue Q_k as the candidate vertex to find the multiple keywords best path query. The same process also goes for keyword *cinema*. The inverted index of keyword *cinema* is 0100. There is only one vertex in the road network that contains 0100, which is v_6. Therefore the best path is going to be based on $\delta(v_{10}, v_6)+\delta(v_8, v_6)$. Hence, v_6 can be stored to a candidate queue Q_k as another candidate vertex for finding the multiple keywords best path query, specifically for keyword *cinema*.

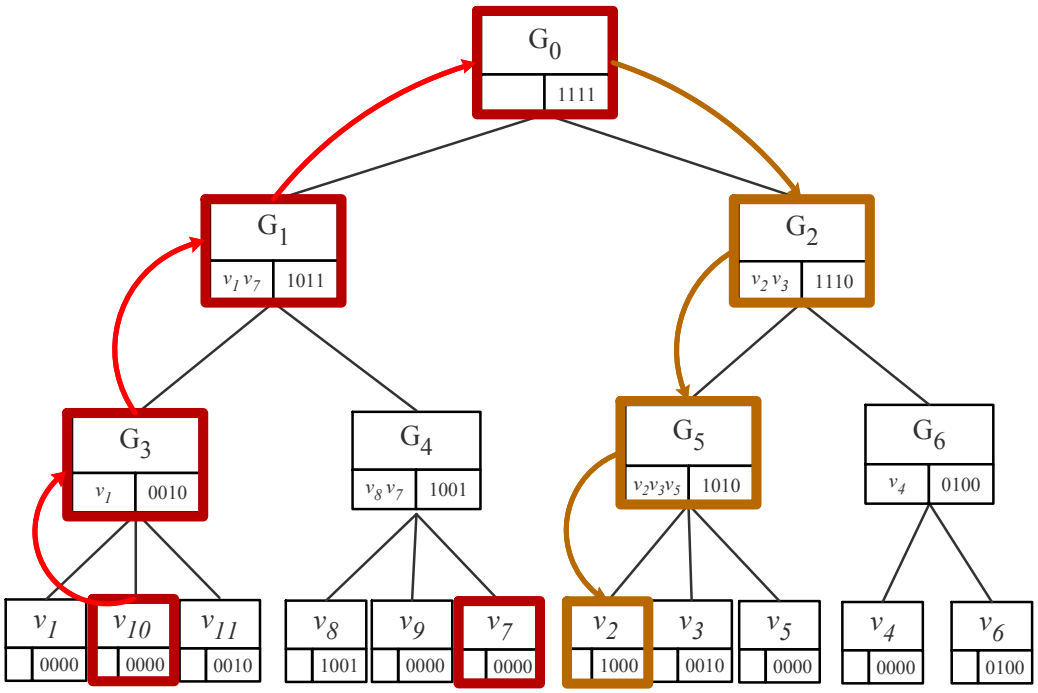


Figure 3.6: Path from v_{10} to nearest node with keyword *book*

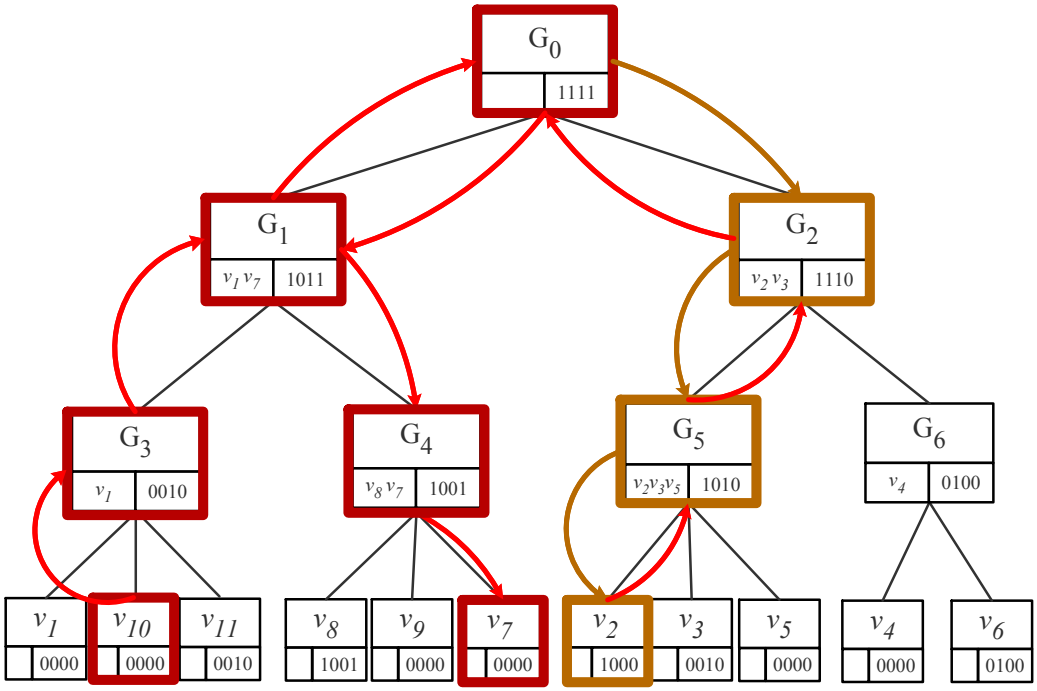


Figure 3.7: Path from v_{10} to v_7 passing through keyword *book*

As we have found the candidates for each keyword specified by the user, we can do best path search from s_l to the candidates, then to d_l. Q_k consists of v_2 and v_6. So what we have to do is to find which vertex in the candidate queue Q_k is the nearest from s_l ($v10$). In this case, v_2 is the nearest so we have to find the shortest path $\delta(v_{10}, v_2) = \{v_{10}, v_1, v_2\}$ with total distance $dist(v_{10}, v_2) = 5$. Next, we have to find the nearest next candidate from v_2, which is v_6. Then we establish another shortest path $\delta(v_2, v_6) = \{v_2, v_5, v_4, v_6\}$ with total distance $dist(v_2, v_6) = 7$. Since there is no more candidate in the queue, then it means that we have found all the keywords specified by the user in our path. Thus we can establish the final path from the last candidate to the destination d_l ($\delta(v_6, v_7) = \{v_6, v_4, v_5, v_3, v_7\}$ with total distance $dist(v_6, v_7) = 11$). The result of the best path $BP(v_{10}, v_7, \{book, cinema\}) = \{v_{10}, v_1, v_2, v_5, v_4, v_6, v_4, v_5, v_3, v_7\}$.

Based on our experiment, this algorithm runs faster than the baseline algorithm even though the approximation is not 100% accurate. However the approximation result is close to the baseline result even when the algorithm runs with a large dataset. The pseudo-code of this algorithm is given in Algorithm 3.

Algorithm 3 Optimal Distance Approximation Search

Input:$IG-Tree$, s_l, d_l, K in
Output:$BP(s_l, d_l, K)$ out
1: Index K to inverted file K_{if}
2: Find s_l on $IG-Tree$
3: Find d_l on $IG-Tree$
4: **for** each inverted keyword k_n in K_{if} **do**
5: **if** k_n+ **then**
6: Find shortest path $\delta(s_l, k_n)$
7: Find shortest path $\delta(k_n, d_l)$
8: $dist(s_l, d_l) = dist(s_l, k_n) + dist(k_n, d_l)$
9: **end if**
10: **if** current path is the shortest **then**
11: Store (k_n) location to queue Q_k
12: **end if**
13: **end for**
14: **for** Q_k is not empty **do**
15: **if** current path is 0 **then**
16: Find 1NN (s_l, Q_k)
17: **else**
18: Find 1NN $(Q_k n-1, Q_k)$
19: **end if**
20: Find shortest path $\delta(s_l, Q_k n)$
21: Remove $Q_k n$ from queue Q_k
22: **end for**
23: Find shortest path $\delta(Q_k n, d_l)$

3.5.3 Ancestor Priority Approximation Search

In this study, we propose another approximation algorithm. This algorithm utilizes the common ancestor between the source and destination locations with the purpose of minimizing the tree traversal time. Sometimes when we are trying to find one or more keywords in the *IG-Tree*, we have to travel through most of the tree nodes even though the source and destination locations are on the same partition node. However in this algorithm, the idea is to traverse only on the branch of an ancestor node. This is basically to do early pruning through the common ancestor between source and destination locations in *IG-Tree*.

As an example, a user invokes a query with source location in vertex v_{10}, destination location in vertex v_7, and the preferred keyword is *book*. The query keyword inverted index in this case is 1000 for keyword *book*. Looking at the *IG-Tree* in Figure 3.8, the common ancestor between v_{10} and v_7 is G_1. Thus in this algorithm we are going to only focus on the branch under G_1, especially if the user given keyword is available in this branch. As the query inverted index is 1000 and the inverted index attached in G_1 is 1011, we can see

that the query keyword exists in G_1, so we can definitely focus on this node to find the best path from v_{10} to v_7 while passing by a keyword *book*.

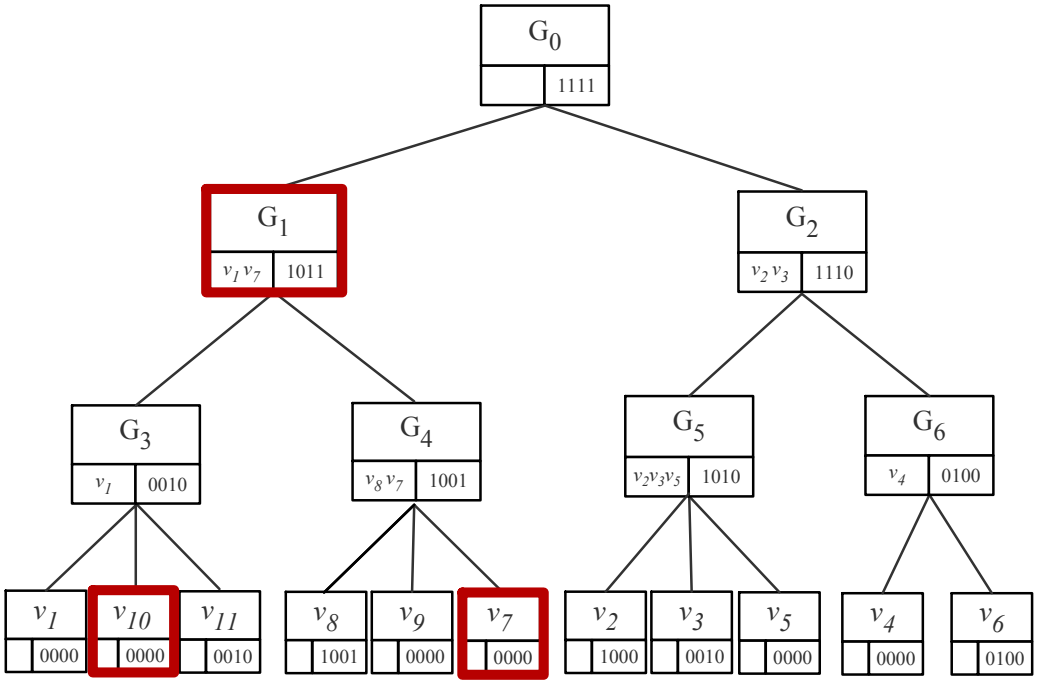


Figure 3.8: G_1 as the common ancestor of v_{10} and v_7

The way the Ancestor Priority Search algorithm works is similar to the Optimal Distance Approximation algorithm once we know which branch we need to work on. Firstly we have to compute the best path of each keyword, then recording the candidate vertices into a queue Q_k in order to find the final multiple keywords best path query. Continuing from the previous example, the focus now is only on the branch of G_1. So we do not need to travel to other branches outside partition G_1. In this case, we have to find the best path for each query keyword first. But since there is only one keyword, then we can find the Best Path directly. The way we find each keyword is through the inverted index attached in the *IG-Tree* and traverse down until we found which vertex has the keyword. We know that G_1 has 1000 so we have to check its immediate children. G_3 does not have 1000, while G_4 has 1000, thus we need to traverse down the partition of G_4 in order to find the keyword. The children of G_4 are v_8, v_9, and v_7. Only v_8 has 1000, therefore we have to find the best path from s_l to v_8 and then to v_7 as depicted in Figure 3.9 . Similar to the previous algorithm, we have to find the shortest path $\delta(v_{10}, v_8)$ and $\delta(v_8, v_7)$ to help finding the best path. The shortest path $\delta(v_{10}, v_8) = \{v_{10}, v_1, v_8\}$, while

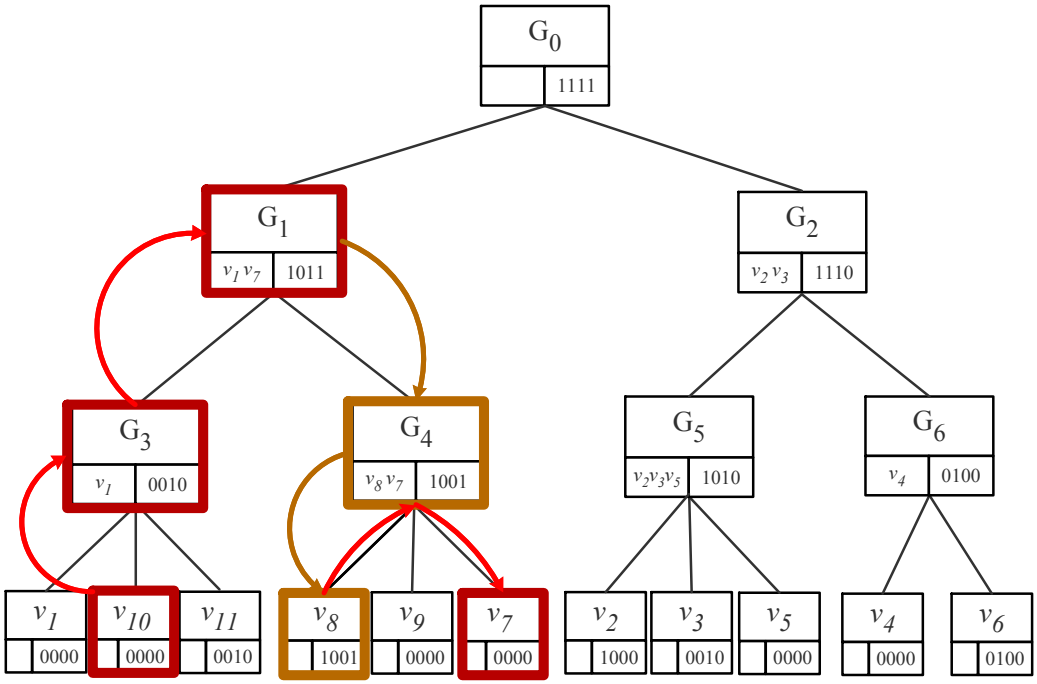


Figure 3.9: Path from v_{10} to v_7 passing through keyword *book*

the shortest path $\delta(v_8, v_7) = \{v_8, v_1, v_7\}$. As there is only one keyword, therefore the Best Path $BP(v_{10}, v_7, \{book\}) = \{v_{10}, v_1, v_8, v_1, v_7\}$.

The previous case however does not always happen because if the keyword does not exist in the current ancestor node, we have to go to its parent node and check whether the query keyword is available in the parent node. If it does not, then we have to keep going to the upper node until we can find the query keyword. Once the query keyword is found in the node, then we can continue the best path search. For example assume that a user wants to find the best path from v_{10} to v_7 while passing through a *cinema*. The inverted index for *cinema* is 0100, while the common ancestor of v_{10} and v_7 is G_1. The partition G_1 does not have *cinema* in it since its inverted index is 1011, thus we have to find out whether *cinema* is available in G_1's parent node. The parent node of G_1 is G_0 and its inverted index is 1111, which means that *cinema* exists in this partition. Therefore the best path search will cover the whole tree branches under G_0.

The main advantage of this algorithm is in the early pruning. There is no need to explore the whole tree as we only need to focus on one branch through the common ancestor between the source and destination. However the disadvantage of this algorithm is that it has even lower accuracy compared to the Optimal Distance Approximation Search algorithm. Traversing under one branch does not guarantee the shortest path

Algorithm 4 Ancestor-Priority Search

Input:$IG-Tree$, s_l, d_l, K in
Output:$BP(s_l, d_l, K)$ out
1: Index K to inverted file K_{if}
2: Find s_l on $IG-Tree$
3: Find d_l on $IG-Tree$
4: Find common ancestor A of s_l and d_l
5: Check if A contains all keywords in K_{if}
6: **while** $A \oplus K_{if}$ is 1 **do**
7: $A = A$'s parent node
8: **end while**
9: **for** each inverted keyword k_n in K_{if} **do**
10: **if** k_n+ **then**
11: Find shortest path (s_l, k_n)
12: Find shortest path (k_n, d_l)
13: **end if**
14: **if** current path is the shortest **then**
15: Store (k_n) location to queue Q_k
16: **end if**
17: **end for**
18: **for** Q_k is not empty **do**
19: **if** current path is 0 **then**
20: Find 1NN (s_l, Q_k)
21: **else**
22: Find 1NN $(Q_k n-1, Q_k)$
23: **end if**
24: Find shortest path $(s_l, Q_k n)$
25: Remove $Q_k n$ from queue Q_k
26: **end for**
27: Find shortest path $(Q_k n, d_l)$

distance for best path since some keywords with closer distances might be located in other partitions. But this algorithm tries to compromise this with lesser tree traversal cost. The pseudo-code of this algorithm is given in Algorithm 4.

3.5.4 Euclidean-based Approximation Search

We propose another approximation algorithm in this study. The idea behind this particular algorithm is to make use of the coordinate of each vertex and then find the best path through Euclidean distance before applying it into road network. This approximation algorithm is very fast compared to the previous algorithms since it is using Euclidean distance computation. However because of the usage of Euclidean distance on a road network data, the performance of this algorithm is quite low in terms of its accuracy.

The Euclidean-based Approximation has two main components, namely the Euclidean approximation (Algorithm 5 row 1-10) and the best road network path (Algorithm 5 row 11-20). In the Euclidean approximation part, we firstly need to find the Euclidean

Algorithm 5 Euclidean-based Approximation Search

Input:$IG-Tree$, s_l, d_l, K in
Output:$BP(s_l, d_l, K)$ out
1: Index K to inverted file K_{if}
2: Find s_l's Euclidean location on $IG-Tree$
3: Find d_l's Euclidean location on $IG-Tree$
4: **for** each inverted keyword k_n in K_{if} **do**
5: **if** k_n+ **then**
6: Find Euclidean shortest path (s_l, k_n)
7: Find Euclidean shortest path (k_n, d_l)
8: **end if**
9: Store (k_n) location to queue Q_k
10: **end for**
11: **for** Q_k is not empty **do**
12: **if** current path is 0 **then**
13: Find 1NN (s_l, Q_k)
14: **else**
15: Find 1NN $(Q_k n-1, Q_k)$
16: **end if**
17: Find Road Network shortest path $(s_l, Q_k n)$
18: Remove $Q_k n$ from queue Q_k
19: **end for**
20: Find Road Network shortest path $(Q_k n, d_l)$

locations of both the source location s_l and the destination location d_l. Based on these two Euclidean locations, we calculate the best path in Euclidean distance for each keyword k_n in K_{if}. The way we find the best path for each keyword k_n is similar to the Optimal Distance Approximation algorithm, where we have to get the optimum shortest path of $\delta(s_l, k_n) + \delta(k_n, d_l)$ in Euclidean distance and then store k_n into a candidate queue Q_k to help establishing the final best path. Once we have found the best path of each keyword k_n, we move to the second part, which is the road network path.

In the road network path component of Euclidean-based Approximation algorithm, the best path search is done in a similar fashion as the previous algorithms where we have to find the nearest candidate Q_k1 from s_l then establish the shortest path $\delta(s_l, Q_k1)$ between s_l and the candidate Q_k1 in road network distance instead of the Euclidean distance. After the shortest path $\delta(s_l, Q_k1)$ is established, we need to find the next nearest candidate $Q_k n$ from the Q_k1 and establish the shortest path $\delta(Q_k1, Q_k n)$. We repeat the same step until every single candidate in the queue Q_k has been visited. Then we can find the shortest path $\delta(Q_k n, d_l)$ to end the trip.

The Euclidean distance is merely to help deciding which vertices to be visited based on the keywords chosen by the user. But at the end the best path's result is in road network distance. The approximation in this algorithm is very low as it can over-approximate the

result up to 300% based on our experiment. However the running time of this algorithm is a lot faster compare to the other algorithms.

3.6 Experiment

In this section, we compare the efficiency and accuracy of the four Best Path query processing algorithms from Section 3.5: Baseline Algorithm (BruteForce), Optimal Distance Approximation Search (OptDist), Ancestor Priority Approximation Search (AncestorPriority), and Euclidean-based Approximation Search (Euclidean).

3.6.1 Settings

Environment

We perform our experiments on 2.5 GHz Intel Core i7-4870 CPU and 12 GB RAM running 64-bit Ubuntu. All of the algorithms were written in single-threaded C++.

Datasets

We use real datasets from 9th DIMACS Implementation Challenge - Shortest Paths [124] and [125] for the road network datasets. We select four datasets: California, New York City, Colorado, and Florida. Table 3.21 provides the details of the size of the real-world road network datasets.

Table 3.21: Road Network Datasets

Dataset	**Description**	**# Vertices**	**# Edges**
CAL	California	21,048	43,386
NY	New York City	264,346	733,846
COL	Colorado	435,666	1,057,066
FLA	Florida	1,070,376	2,712,798

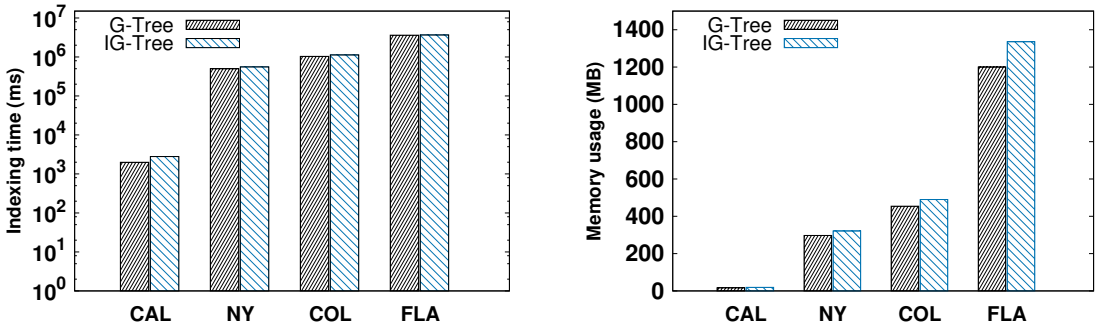
Meanwhile for the textual information we utilize keyword sets based on [125] and assign them into the vertices in the road network datasets. As the textual part needs to be able to detect whether the user gives one or more negative keywords, a number of negative sentiment analysis based words from [126, 127] are used in order to accommodate the negative keyword(s) in the user query.

Queries

In our experiments, we generate the keyword set K for the test queries with a random distribution from a keyword pool. The size of the keyword set in the test queries varies from 1 to 15 while the object density [1] of these keyword set varies from 1% to 30% of the whole road network datasets. We also evaluate the impact of varying the distance between s_l and d_l pairs, which are varied from 2% to 64% of the maximum distance between two vertices in the space.

3.6.2 Index Evaluation

This section evaluates the proposed *IG-Tree* index for planning Best Path queries on road networks in terms of index building time and space consumption, and index reconstruction time(s) for negative keywords in the tested queries. Figure 3.10 shows the index building times and space consumption of the proposed *IG-Tree* and the *G-Tree* indices for New York City dataset. We see that index building time of *IG-Tree* is comparable to that of *G-Tree* though *IG-Tree* combines IR^2-*Tree* with *G-Tree*. The index size of *IG-Tree* is slightly larger than that of *G-Tree* as inverted lists and Keyword Distance Matrices are maintained in *IG-Tree* in addition to Distance Matrices.



(a) Building time

(b) Index size

Figure 3.10: IG-Tree vs. G-Tree: index building time and size

Finally, the times required to reconstruct the *IG-Tree* index for negative keywords in the Best Path queries are quite durable as we observe from Figure 3.11. The *IG-Tree* takes only ~ 0.8 secs to reconstruct the index for up to 10 (negative) keywords. However, we observe only a few keywords in K including negative keywords in route planning

[1] object density: the quantity of keyword matched objects for each query keyword compared to the number of vertices in the road network.

queries, which is around 5-6 keywords, in our usual life. Therefore, we believe that the index reconstruction time *IG-Tree* for few negative keywords would be pretty durable in practical applications of Best Path queries.

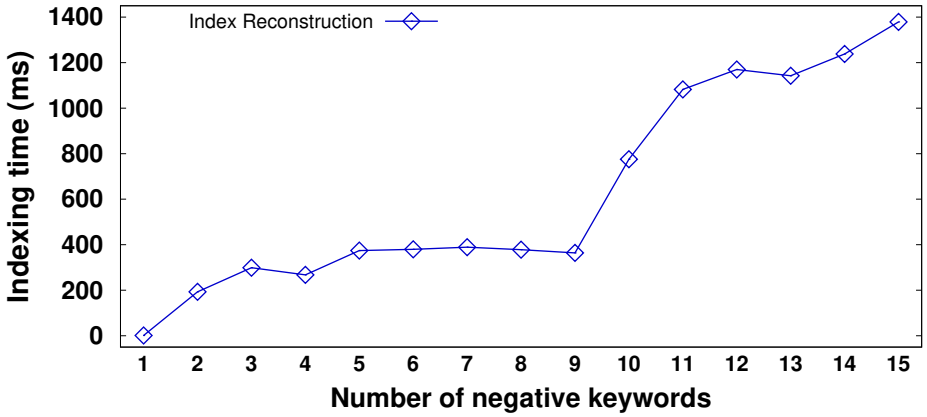


Figure 3.11: Index reconstruction times for varied number of negative keywords in K

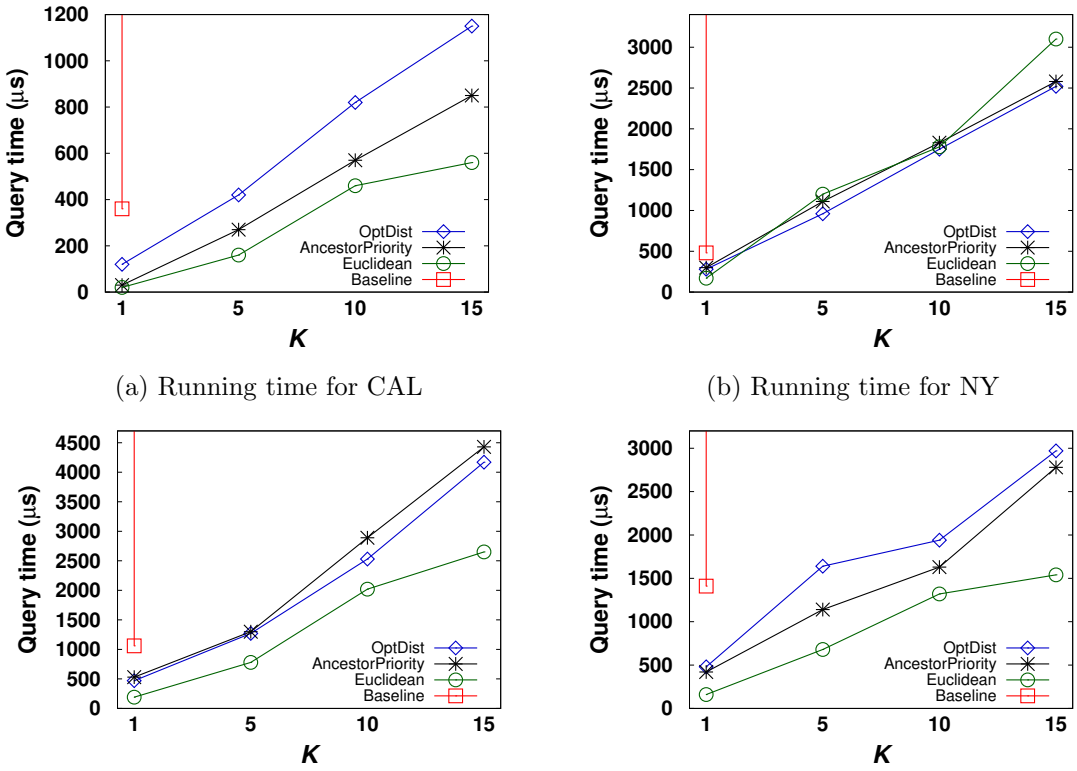
3.6.3 Performance Study

We evaluate our query processing algorithms on two metrics, specifically on the running time and approximation accuracy. The approximation accuracy shows the percentage of the result accuracy produced by each algorithm compared to the expected correct result obtained by running the baseline algorithm.

Effect of k^+

The positive keywords (k^+) given by the users take a very important role in Best Path Query. Each k^+ must be visited at least once, therefore the more k^+ to be visited, it is expected that the running time also increases for every algorithm. Figure 3.12 shows the query performance as the number of k^+ increases. In these experiments, we specified the query keywords K to be all positive, without any negative keywords. The experiments show that the running times for the approximation algorithms (OptDist, AncestorPriority, Euclidean) run a lot better compared to the baseline algorithm. The baseline algorithm has the worst running time among all as the time increases exponentially. According to our experiments, the average running time for baseline algorithm for $|K| = 1$ is 0.48 ms but it increases up to 21507.41 ms when $|K| = 5$. Even though baseline algorithm offers a precise solution, the amount of time taken to get the result is not suitable for daily usage.

Imagine when we want to plan a trip to a new country and it takes 17 hours for us to get the best path with $|K| = 10$. This is definitely impossible to be used in everyday life.



(a) Running time for CAL

(b) Running time for NY

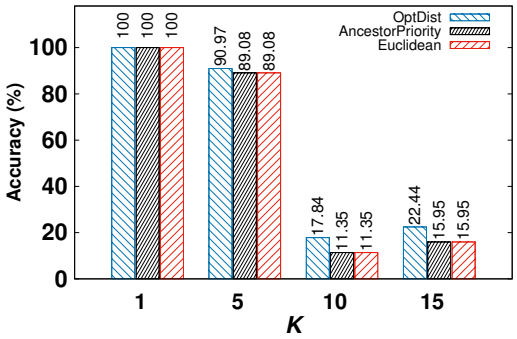
(c) Running time for COL

(d) Running time for FLA

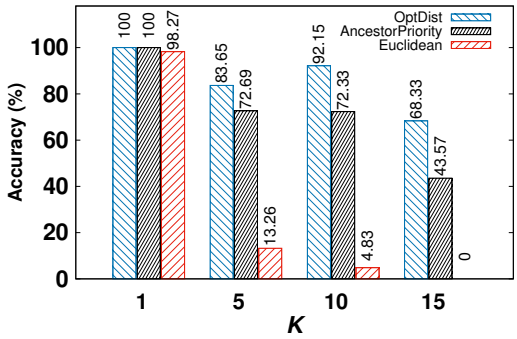
Figure 3.12: Query performance with all positive query keywords

As we proposed three approximation algorithms, we also evaluate the approximation accuracy for each algorithm. Figure 3.13 shows the percentage of accuracy of each approximation algorithm compared to the baseline algorithm. When K is only 1, all the three approximation algorithms have high accuracy. However when K increases, the accuracy decreases. The OptDist has the best approximation compared to the other two algorithms. Even though its approximation is not 100% accurate, the percentage of accuracy is still above 75%. It is very different from the Euclidean-based algorithm to which its approximation is very poor compared to others as the accuracy is only 6.83% when $K = 15$. The trend for Euclidean approximation algorithm is definitely the worst as the inaccuracy keeps escalating drastically.

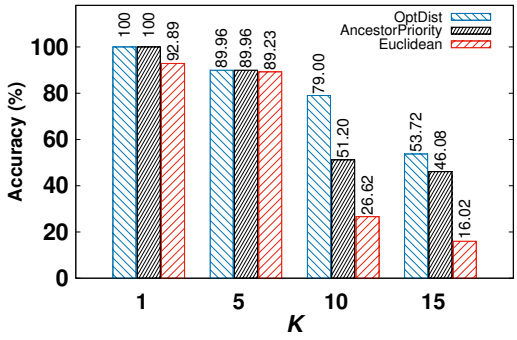
Based on this experiment, we can see that OptDist performs better than the other three algorithms in terms of the running time. It also performs better than the other two approximation algorithms in terms of the approximation accuracy. So we can conclude



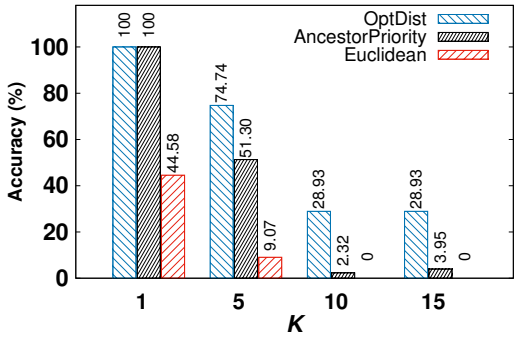
(a) Approximation accuracy for CAL



(b) Approximation accuracy for NY



(c) Approximation accuracy for COL



(d) Approximation accuracy for FLA

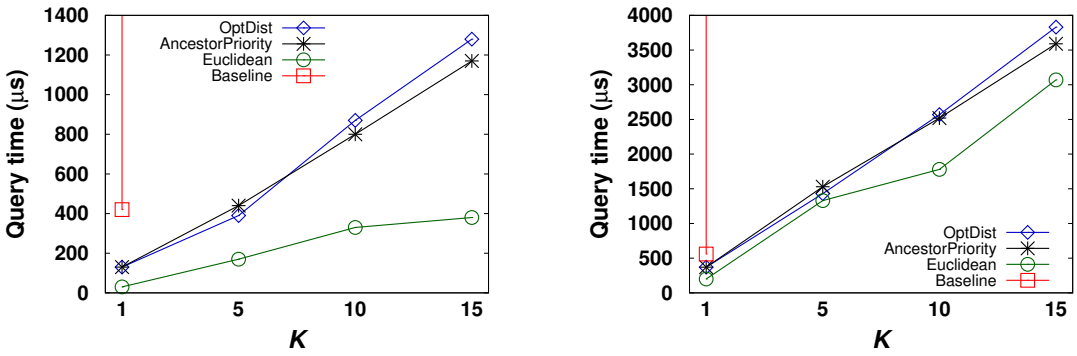
Figure 3.13: Approximation accuracy for all positive query keywords

that OptDist is the best choice when we want to invoke Best Path Query with various numbers of K.

Effect of k^-

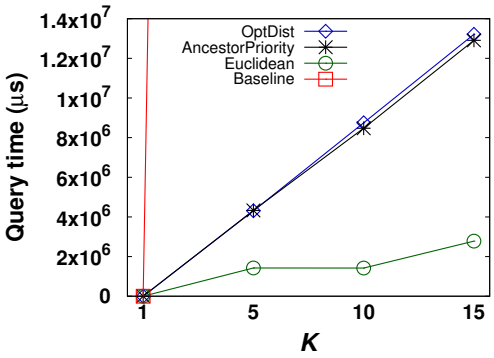
The k^- has a great impact to the Best Path Query. As previously discussed in Section 3.4, there are several cases on what would happen to the *IG-Tree* when we found a k^-. A lot of times when k^- is located on the border, the path cannot be retrieved at all. So the distribution of k^- is always kept to be lesser than k^+ in this particular experiment to ensure that we can retrieve some results. For this experiment, we set the query keywords K to have both positive and negative keywords.

According to our experiment result in Figure 3.14, the Euclidean-based algorithm always has a faster running time compared to the other three algorithms. The OptDist and AncestorPriority are actually almost the same in terms of their running time even though the AncestorPriority still seems to be a bit faster than OptDist. Meanwhile the baseline algorithm has the worst running time as expected. Having a negative keyword k^- definitely affects the running time of some queries as there might be path reconstruction

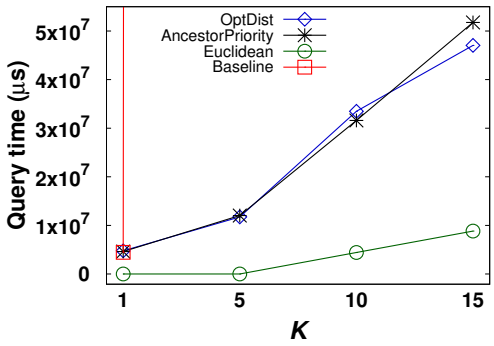


(a) Running time for CAL

(b) Running time for NY



(c) Running time for COL

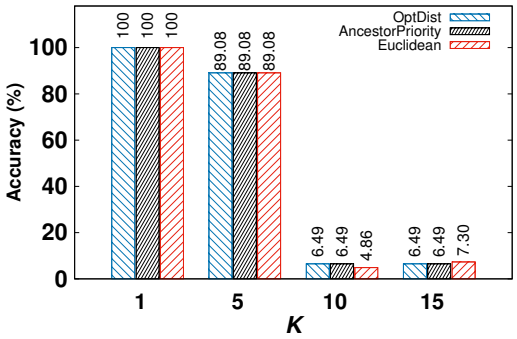


(d) Running time for FLA

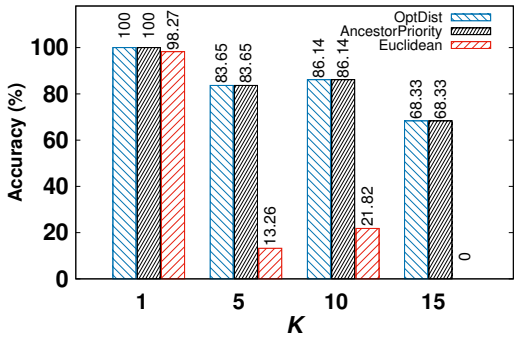
Figure 3.14: Query performance with combination of positive and negative query keywords

happening throughout the query processing. This also explains the difference between the time in Figure 3.12 and Figure 3.14, where the running time in Figure 3.12 with all positive keywords does not require any path reconstruction so it is faster than the experiment result in Figure 3.14.

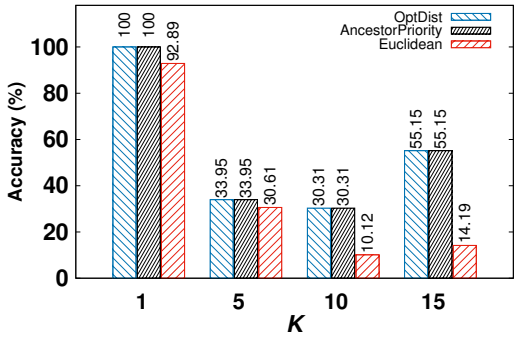
Another metric that we test is the accuracy of the approximation algorithms. Figure 3.15 shows the result of the approximation accuracy of each algorithm towards the result of baseline algorithm. The trend in Figure 3.15 is almost similar to the trend in Figure 3.13, which might indicate that the path reconstructions happening in these queries because of k^- does not truly have impact towards the accuracy. We can also see that the accuracy percentage of OptDist and AncestorPriority are the same for this case and both have better accuracy than the Euclidean-based algorithm. The Euclidean approximation is still the worst among the other algorithms with very low accuracy even though the running time is a bit faster than the others.



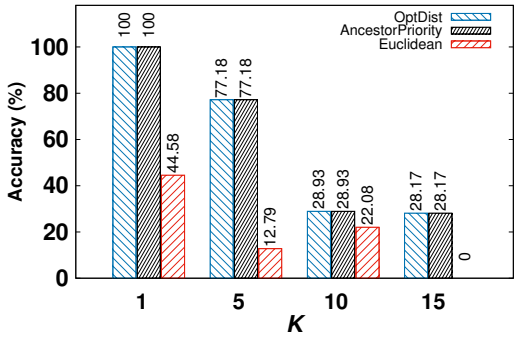
(a) Approximation accuracy for CAL



(b) Approximation accuracy for NY



(c) Approximation accuracy for COL



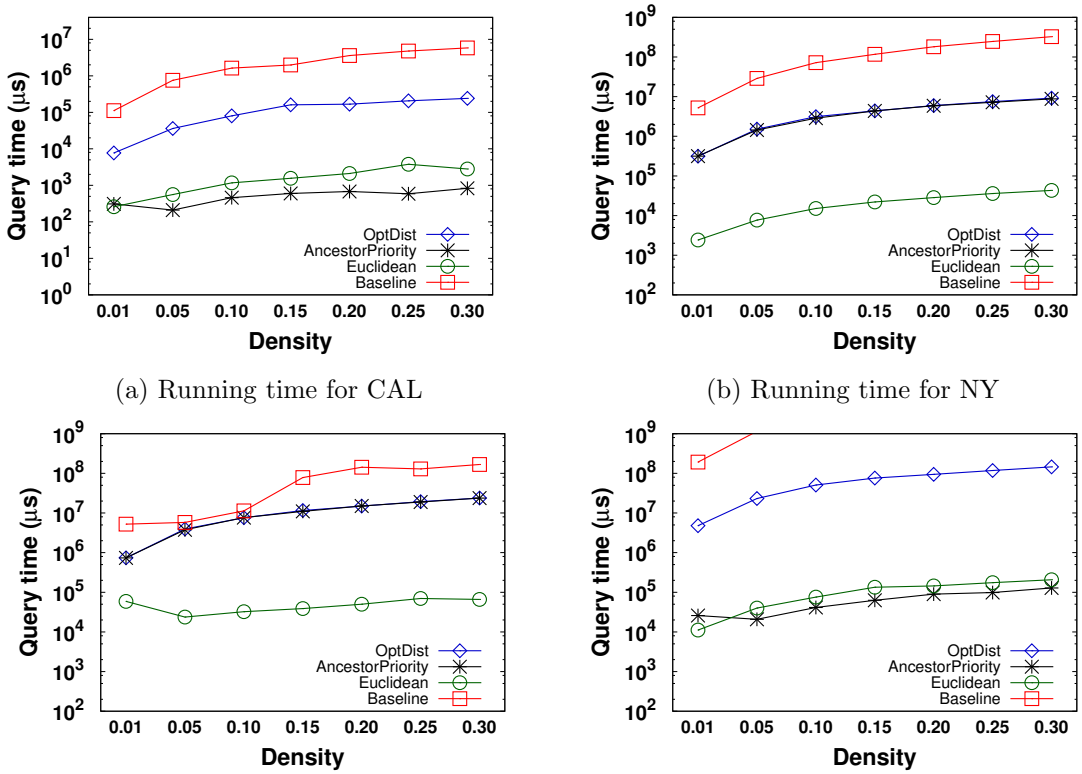
(d) Approximation accuracy for FLA

Figure 3.15: Approximation accuracy for datasets with negative keywords

Effect of Keyword Densities

In this experiment evaluation, we want to observe the query performance when we increase the keyword densities. Figure 3.16 shows the running time for query within the density of 0.01 to 0.30. The running time of the baseline algorithm increases drastically compared to the rest. Even in the lower density case, specifically on 0.01 density, the baseline algorithm takes about 17x more time than the OptDist algorithm. The Euclidean-based algorithm however performs in a constant manner even though the density increases. The OptDist and AncestorPriority on the other hand have similar running time.

We also run an experiment on the running time when K is varied from 1 to 15 while the density for each keyword is 0.05. Figure 3.17 shows the result of this specific experiment where it interestingly shows that the constant running time for Euclidean algorithm. The trend indicates that Euclidean-based algorithm has the most constant running time especially for denser datasets. This is totally different from the other three algorithms, which keep increasing with the increase in density. Even though Figure 3.12 and Figure



(a) Running time for CAL

(b) Running time for NY

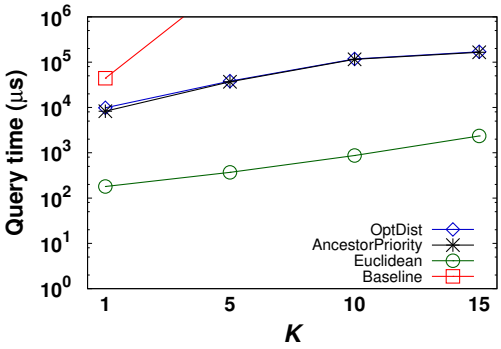
(c) Running time for COL

(d) Running time for FLA

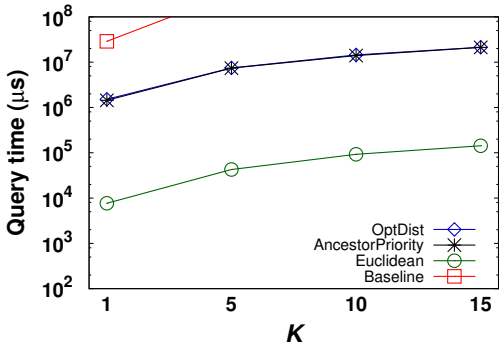
Figure 3.16: Query performance based on keyword density

3.14 run similar experiments, the results are different. In Figure 3.12 and Figure 3.14, the keyword density is randomized and usually below 0.05.

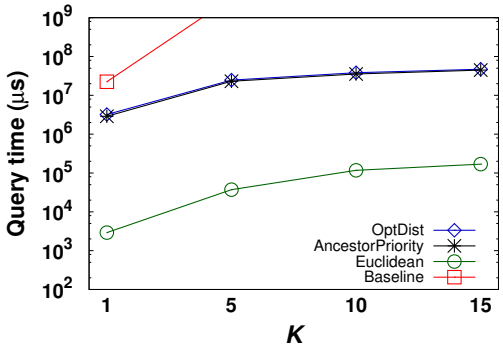
We conducted another experiment to test the running time of query with all negative keywords while increasing the density of the keywords from 0.01 to 0.055. We only increase until 0.055 by the reason of having higher density of negative keywords will return no path/no result at all. Figure 3.18 shows the result of this particular experiment. Surprisingly, the running time of Euclidean algorithm drastically increases almost in the same trend as the baseline algorithm. Meanwhile, the OptDist and AncestorPriority are running in constant time manner when the keywords are all negative.



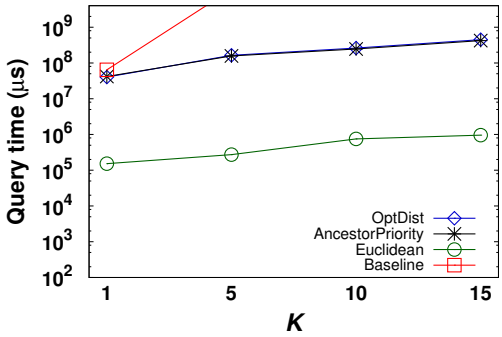
(a) Running time for CAL



(b) Running time for NY

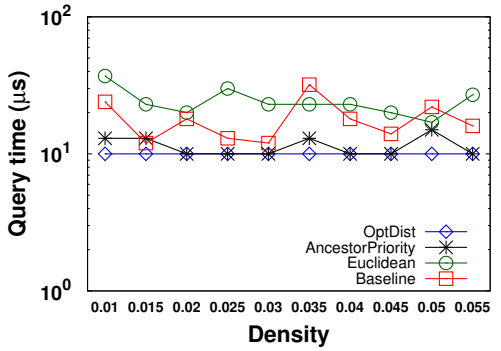


(c) Running time for COL

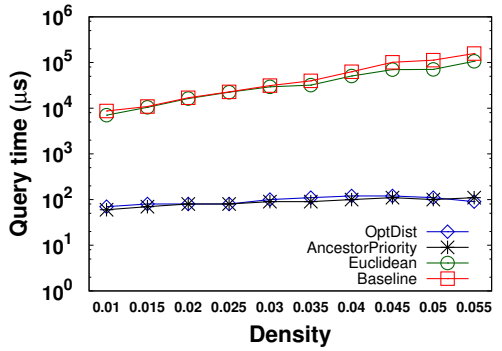


(d) Running time for FLA

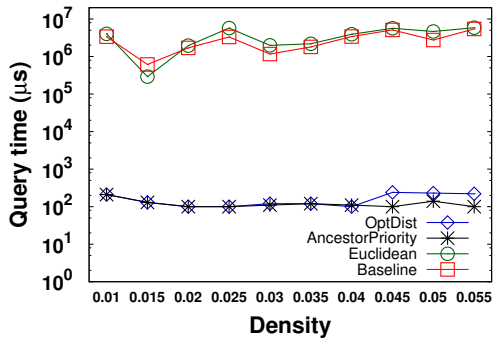
Figure 3.17: Query performance when K is varied (keyword density=0.05)



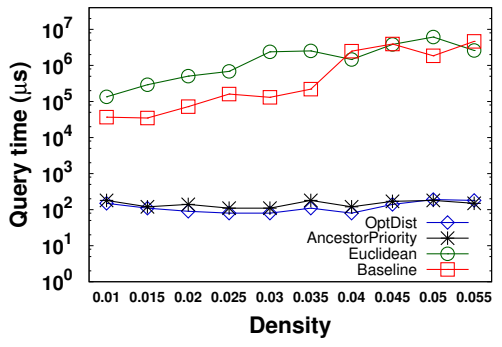
(a) Running time for CAL



(b) Running time for NY



(c) Running time for COL



(d) Running time for FLA

Figure 3.18: Query performance based on keyword density with all negative keywords

Effect of Positive and Negative Keywords Ratio

Figure 3.19 shows the running time on the positive and negative keywords ratio. The ratio is based on 0.01 density. In this experiment, we limit the ratio of the keywords (positive:negative) into 1:0, 0:1, 1:1, 5:0, 0:5, and 5:1. We exclude the ratio with negative keywords higher than the positive keywords as there is no path retrieved most of the time with this kind of query.

In Figure 3.19, we can see that the running time of OptDist algorithm increases if the number of query keywords increase (both positive and negative). The AncestorPriority has similar trend with OptDist with the increase of time towards the more keywords involved. The increase in trend also happens to the baseline algorithm but it increases exponentially as what we have expected from the previous experiments. Nevertheless, the increasing trend does not happen to the Euclidean algorithm as it is comparatively constant in running time even with more positive keywords added into the query.

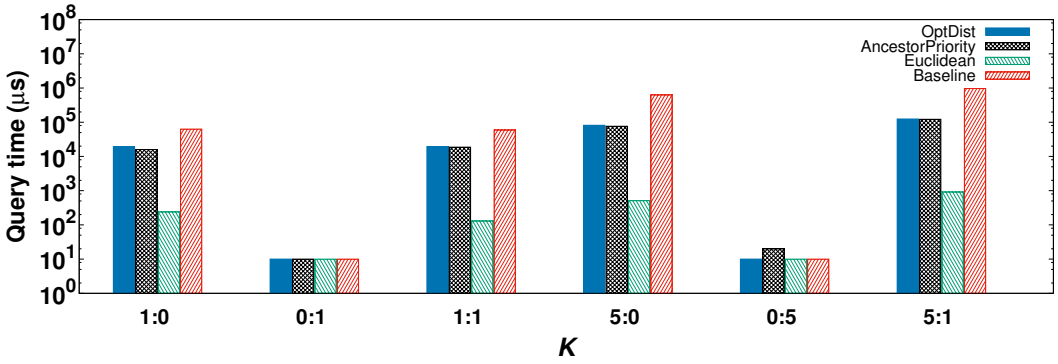
Effect of Distance Between s_l and d_l

We also evaluate the effect of varying the distance between s_l and d_l pairs. The distance between s_l and d_l are varied from 2%, 4%, 8%, 16%, 32%, up to 64% of the maximum distance between two points in the road network datasets. We set $|K| = 15$ by default and contain both positive and negative keywords. Figure 3.20 shows the experimental results of the source-destination distance. We do not include the baseline in Figure 3.20 since the runtime for 2% in NY dataset already reaches above 10,000 ms, which made a huge difference with the other three algorithms.

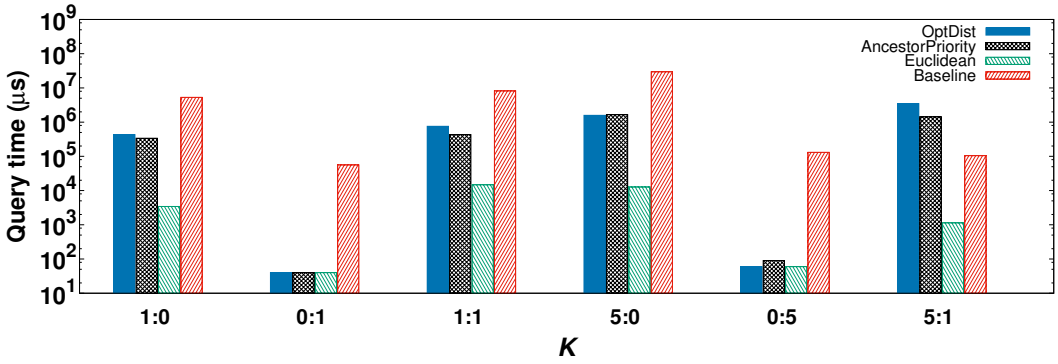
From the experimental result, we can see that the three algorithms are running in constant time even though the source-destination distance increases. Both OptDist and AncestorPriority have a similar trend, while Euclidean has better running time most of the time than the rest.

Summary

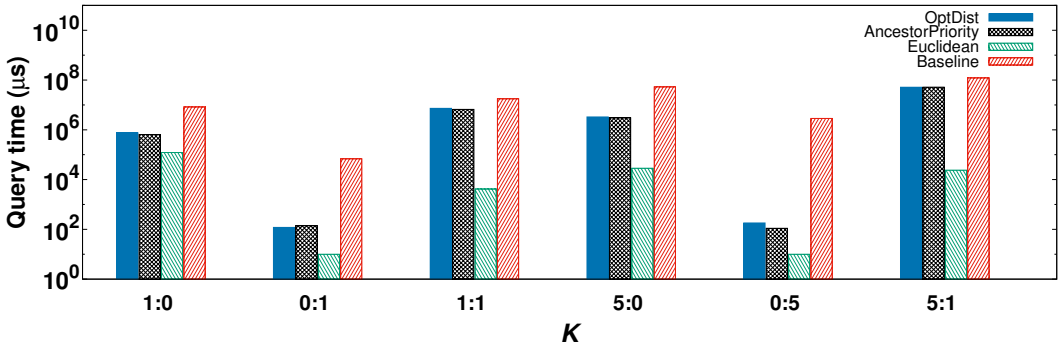
Based on our experimental study, each algorithm has its own strength. The baseline algorithm certainly offers accurate result, but it has the worst running time as it increases



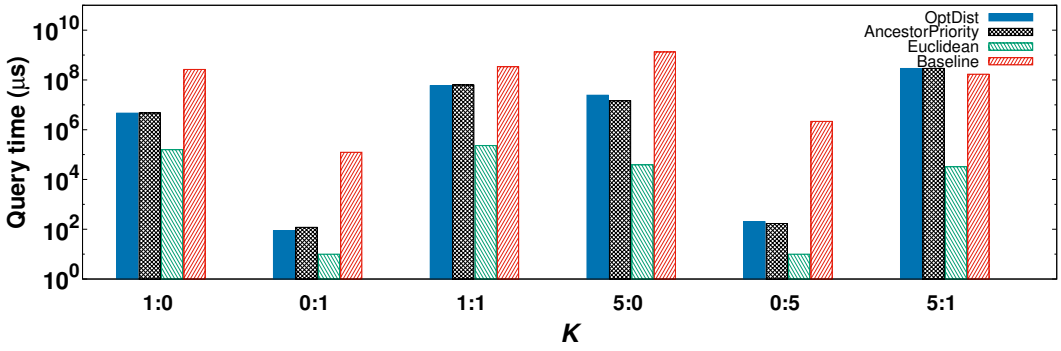
(a) Running time for CAL



(b) Running time for NY

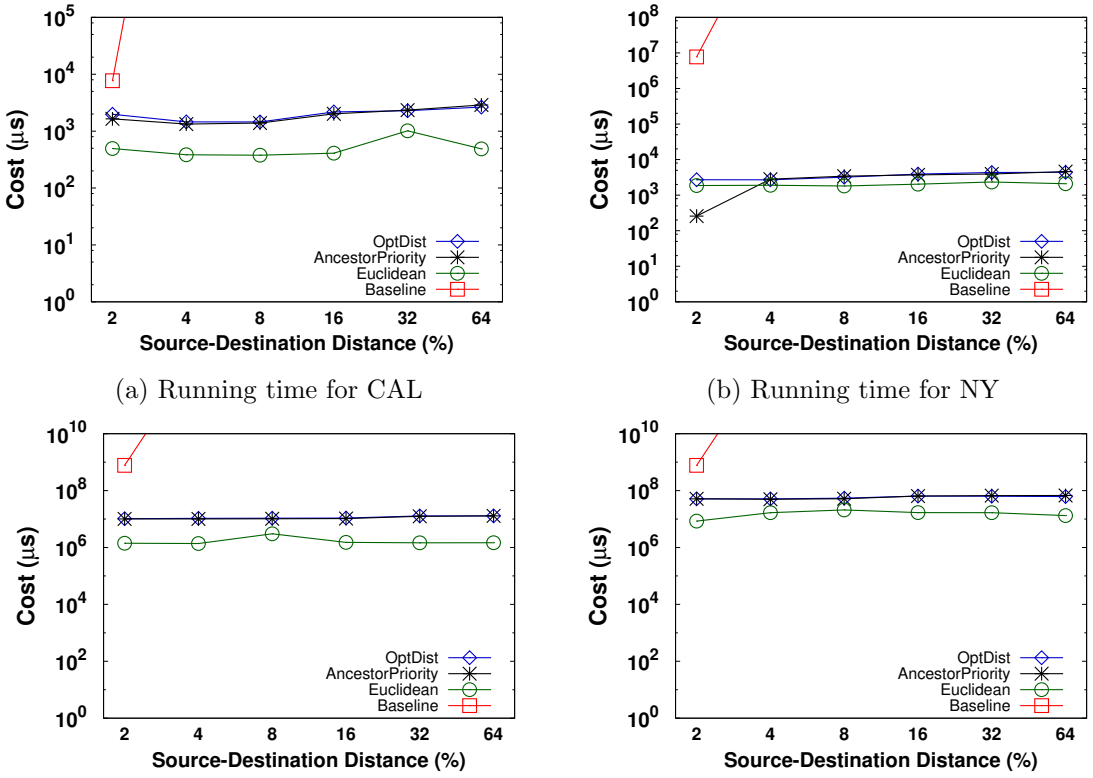


(c) Running time for COL



(d) Running time for FLA

Figure 3.19: Running time based on keyword ratio (positive:negative)



(a) Running time for CAL

(b) Running time for NY

(c) Running time for COL

(d) Running time for FLA

Figure 3.20: Effect of varying distance between s_l and d_l

exponentially when the queries and dataset increase. The OptDist itself has the best approximation compared to the other two approximation algorithms (AncestorPriority and Euclidean). Its running time is definitely a lot better than the baseline algorithm but the AncestorPriority and Euclidean-based algorithms still beats it by a few microseconds, especially when all the keywords in the user query are positive.

The AncestorPriority often follows the trend of OptDist on both the runnning time and accuracy. In terms of running time, AncestorPriority is frequently faster by only a few microseconds compared to OptDist. On the other hand, the accuracy of AncestorPriority is slightly lower than OptDist because of the early pruning. Compared to the Euclidean, AncestorPriority still has better accuracy even though its speed is still slower than the Euclidean.

The Euclidean-based algorithm's main strength is in its fast runtime. For a quick approximation, the Euclidean-based algorithm can be used but it has the lowest accuracy compared to the other algorithms. The Euclidean however does not perform well when the density of the negative keywords are high.

In general, OptDist offers the best solution in comparison with the other algorithms. Even though the Euclidean-based algorithm outperforms the running time of OptDist when the queries contain all positive keywords, OptDist still provide better approximation. OptDist is more stable in both its speed and approximation accuracy.

3.7 Conclusion

In this study, we introduce a new variant of Spatial Keywords Query on Road Networks, which is the Best Path Query. The Best Path Query is an *NP-Hard* problem as it can be reduced to the Travelling Salesman Problem (TSP). Throughout our study, we develop an indexing technique called the *IG-Tree* that can process both spatial and textual information on road networks environments. This indexing technique can be used on various types of Spatial Keywords Query, especially on the Best Path Query. Three algorithms to solve the Best Path Query are also proposed in this research, namely the Optimal Distance Approximation Search, Ancestor Priority Search, and the Euclidean-based Approximation solution. Each algorithm has its own strengths and weaknesses. The effectiveness and efficiency of the proposed algorithms are demonstrated through our extensive experiments.

Chapter 4

Best Path Queries on Weighted Regions

4.1 Overview

With the rapid growth in Location-Based Services (LBS) these days, the way people interact with LBS applications is becoming more advanced [128, 129]. The LBS applications must be able to cater user needs despite the varieties of user queries. Users are no longer asking for simple navigation search or finding specific spatial location of a Point of Interest (POI), but they now ask for more complicated queries like route planning [20]. On top of that, the user queries are not merely in a form of spatial information, but also involving other factors like textual information, which adds more challenges to the LBS applications [130]. Due to this reason, many researches have been conducted in order to improve the capability of LBS applications [131, 132, 133, 134, 135, 136].

One of the most popular study in this area is on route planning. The main objective of route planning is to help users efficiently plan their trip from a source location to several different locations based on their preferences and then ending the trip at a target location. The preferences in this case can be in a form of keywords. For example, a user wants to go home from her workplace and on her way home she has to stop by a supermarket to grocery shop, refilling car's fuel, and getting take away for dinner. Thus, the user will invoke a query that contains her source and destination locations and several keywords for her preferred places to visit like supermarket, fuel station, and restaurant. The input of this particular query involves both spatial and textual information, which is known as

Spatio-Textual Query, while the output will be a path with the most efficient cost (e.g. time, distance) taken. There are many variants of route planning query in Spatio-Textual area, and one of the recent variants is the Best Path Query [137].

In Best Path Query, given a source location s_l, a destination location d_l, and a set of keywords $K = \{k_1, k_2, ..., k_n\}$, where each k_i for $1 \leq i \leq n$ can be positive (denoted by k^+) or negative (denoted by k^-), find the Best Path from s_l to d_l, denoted by $BP(s_l, d_l, K)$, that passes through all k^+ and avoid all k^- with optimum cost. Based on this definition, the Best Path Query has the same goal with general route planning queries but it also considers negative keywords in the path searching process. The negative keywords here indicate places that users want to avoid, which it actually increases the complexity of the problem. The recent study of Best Path Query offers some solutions to the Best Path problem, including a new indexing technique that can be used to process both spatial and textual information. However, the current Best Path problem only focuses specifically on road networks environment.

Even though there is a significant number of solutions have been proposed to solve route planning problem, the focus of the existing works on route planning in Spatio-Textual area are always on either Euclidean space or Road Networks. Only recently that researchers start to expand to Indoor spaces [110]. Even so, there is no study yet on Weighted Regions environment. Consider someone with special needs, such as those with wheelchairs, who has to plan a trip in advanced as he has to consider the topology of the road he has to pass through. A path with staircases that gives the most efficient cost for general users would not work for these people as it is not easily accessible by them. So the existing works lack in the consideration of the different terrain that the user has to pass.

Moreover, the Euclidean distance-based approach is not always accurate as it does not reflect the real representation of the earth as the earth's surface is not constantly a flat concrete surface [25]. Due to this reason, Euclidean distance does not actually offer an optimal solution in real life. The same fact also applies to road networks. Real road networks only offer the distance of the most common passable roads. Unfortunately the distance is being generalised without considering the impact of the earth terrains. For example the amount of effort to pass a flat surface covered in grass is different from a flat concrete surface. The effort of passing through a slope is totally different from a flat surface. So not all roads are easily accessible. This is particularly crucial in concern of

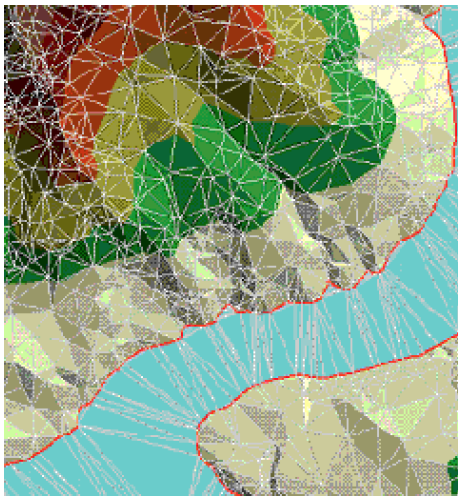


Figure 4.1: A map that incorporate various terrain [3]

various types of vehicles or even for wheelchair users who has to carry themselves through roads with different types of surface and sloped angles. Because of these reasons, we conduct our study to investigate the impact of the *Best Path Query on Weighted Regions* (wBestPath) and propose the best solution for route planning on a complex environment like this.

Similar to the Best Path Query on road networks, the wBestPath has to deal with both spatial and textual information. The query that the users give in wBestPath consists of source and destination locations, and a set of keywords that indicates the places the users want to visit or avoid throughout their trip before arriving to the destination. The keywords can be in a positive and negative conditions. Positive keyword means the place the user wants to visit, while the negative keyword means the place that the user wants to avoid. The main difference between the traditional Best Path [137] and the wBestPath is in the space model where wBestPath deals with more complex environment, which is the weighted regions. In this study, the weighted region is formalised to Delaunay Triangulation representation and each triangle is assigned with a weight. Figure 4.1 shows an example of a weighted region map that incorporate various terrains. This map is subdivided into polygonal regions and each region has a specific weight depending on the type of terrain. Hence, the definition of wBestPath is as follow:

Definition 4. ***(wBestPath)*** *Given a weighted regions space $DT(P)$, where each region dt_i is assigned with a certain weight w_{dt_i} ($w_{dt_i} \in [1, +\infty]$), a set of spatio-textual objects O, and a set of user queries that consist of a source location s_l, a destination location d_l,*

and a set of keywords $K = \{k_1, k_2, ..., k_n\}$*, where each* k_i *for* $1 \leq i \leq n$ *can be positive (denoted by* k^+*) or negative (denoted by* k^-*), find the Best Path from* s_l *to* d_l*, denoted by* $wBestPath(s_l, d_l, K)$*, that passes through all* k^+ *and avoid all* k^- *on* $DT(P)$ *with the least amount of travel cost.*

As an example of wBestPath query, assume that a user wants to plan a trip from his current location to his house while visiting a bookshop and a cafe in between his trip. So in this case, the query consists of a source location s_l and a destination location d_l, and also some keywords, which are *bookshop* and *cafe* ($K = \{bookshop, cafe\}$). In this query, we have to return the most optimum path that satisfies the user's preferred keyword conditions. Using Figure 4.2 as an illustration of this example, the weighted regions space is denoted by the triangulations while the region weights are indicated by the grey level of each triangle. The darker the grey level, the heavier the weight of the region. Looking at this illustration, the user's source and location are located on different area and there are also two cafes and a bookshop in the space. Hence there are several possibilities of route combination. The possible combinations consist of $\{source \rightarrow cafe1 \rightarrow bookshop \rightarrow destination\}$, $\{source \rightarrow cafe2 \rightarrow bookshop \rightarrow destination\}$, $\{source \rightarrow bookshop \rightarrow cafe1 \rightarrow destination\}$, and $\{source \rightarrow bookshop \rightarrow cafe2 \rightarrow destination\}$. In the Euclidean space model, the ideal solution is to have the route from $\{source \rightarrow cafe1 \rightarrow bookshop \rightarrow destination\}$. However, because of the weight of the

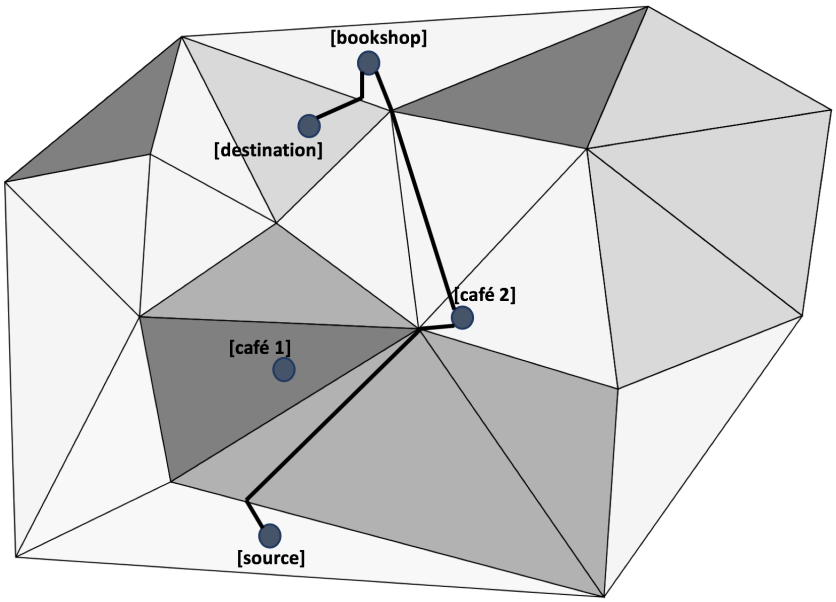


Figure 4.2: Illustration of route planning on weighted regions

regions, this is not the best solution as cafe 1 is located on a heavy weighted region. The path computation for wBestPath incorporates the weight of each region. Thus in this example, $\{source \rightarrow cafe2 \rightarrow bookshop \rightarrow destination\}$ gives a better solution because we also have to consider the effort of passing through each weighted region in the path computation. The problem may also elevate when we start adding negative keywords to the query, in which the regions where the negative keywords are located have to be avoided.

4.1.1 Challenges

A significant number of solutions to solve route planning in Spatio-Textual context have been proposed in the past years, even with the fact that route planning is an *NP-Hard* problem [23, 24]. Even though route planning problem has been widely studied, there is no existing solution to route planning on weighted regions. Current solutions are mostly focusing on simple Euclidean space model or Road Networks model, which unfortunately cannot be applied to weighted regions space model.

As previously mentioned, we formalise the weighted regions to Delaunay Triangulation as Delaunay Triangulation is commonly used for modelling terrain [138]. A matter that we need to consider when working with a terrain is that every surface is different. Figure 4.1 shows an example of terrain map. Each colour indicates different types of surface. When travelling through different surfaces, the effort taken will varied depending on the type of the surface. We can say that each surface has its own travel cost/weight that will affect our path planning. For instance when a vehicle is going from a concrete surface to a soil surface, the speed or even the direction of the vehicle might change because of the shift of terrain. Suppose that the vehicle maintains its speed when entering the new surface, then there must be a refraction on the path of the vehicle.

The weight of the terrain surface has a major impact on wBestPath. When transitioning between different terrain, the path will be affected. Therefore we utilize Snell's Law of Refraction to compute the impact [139]. This law is to identify the refracted angle to travel between different types of surface, which can be computed through the following ratio: $\frac{\sin\theta_1}{\sin\theta_2} = \frac{v_1}{v_2} = \frac{\lambda_1}{\lambda_2} = \frac{n_2}{n_1}$, where θ is the angle measured from the normal boundary, v is the velocity, λ is wavelength, and n is the refractive index (in our case it is the surface's weight).

The problem in this study is also intensified with the addition of negative keywords in the query as we have to ensure the route chosen will not pass through any of these negative keywords. Thus, with all of these challenges, we strive to provide some solutions to solve wBestPath problem.

4.1.2 Contributions

Our primary contributions in this chapter are summarised as follows.

- We formalise the wBestPath, a study on route planning on weighted regions. To the best of our knowledge, this research is the first to attempt on this kind of problem.
- We propose *wIG-Tree*, an indexing technique that incorporates spatio-textual attributes that can be used to solve Best Path problem on weighted regions.
- We propose two approximation algorithms with different trade-offs that utilise the *wIG-Tree* to process the wBestPath queries.
- We conduct a comprehensive set of experiments to demonstrate the effectiveness and efficiency of our algorithms.

4.1.3 Organisation

The remaining sections are organised as follows. Section 4.2 presents the preliminaries and the query model. Section 4.3 briefly discuss some basic concepts that are used in this study. Section 4.4 presents our proposed index structure, which is called the *wIG-Tree*. Section 4.5 discuss the impact of negative keywords to the wBestPath problem. Section 4.6 explains our proposed algorithms to solve wBestPath. Section 4.7 reports the experimental results of our proposed solution. Finally, we conclude this chapter in Section 4.8.

4.2 Problem Statement

In this section, we present the formal definition for wBestPath as well as the definition of weighted regions and data model.

Weighted Regions We consider weighted region as a polygonal subdivision that is denoted by Delaunay Triangulation $DT(P)$. The $DT(P)$ itself is specified by a set of points/vertices $P = \{v_1, v_2, ..., v_n\}$, a set of edges E, and a set of triangle faces $TF = \{dt_{v_1,v_2,v_3}, dt_{v_2,v_3,v_4}..., dt_{v_{n-2},v_{n-1},v_n}\} \subset DT(P)$. Each point v_n in P has a coordinate location $\{x, y\}$, where x and y are the longitute and latitude. Each edge $(v_i, v_j) \in E$ connects two adjacent points $v_i, v_j \in P$. Each dt_{v_i,v_j,v_k} has a weight $w_{dt_{v_i,v_j,v_k}} \in [1, +\infty]$ that denotes the cost to pass the region.

Figure 4.3 shows an example of our space model. The whole region is denoted by Delaunay Triangulation that consists of a set of vertices and each face has its own weight which is indicated by the grey level. In this case, the darker the colour of the triangle face, the more expensive the weight to travel through this particular region.

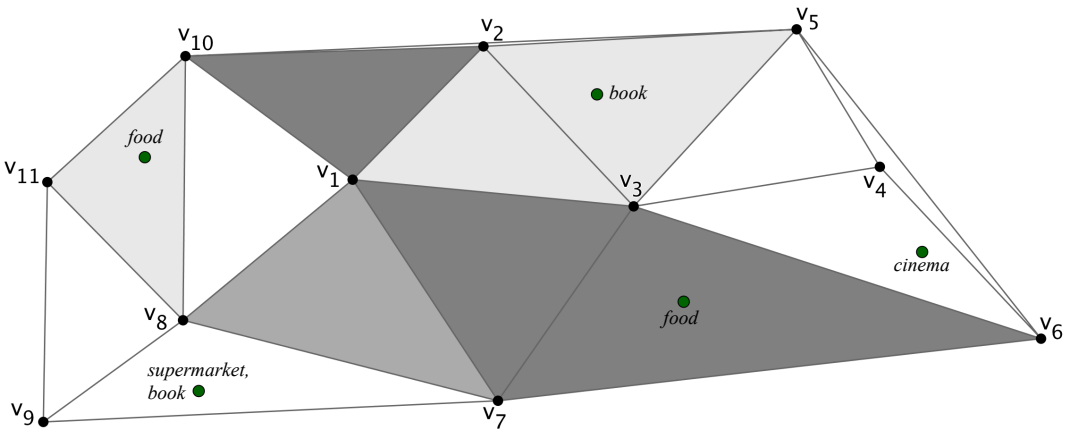


Figure 4.3: Weighted regions space model

Data Model A spatio-textual object o is an object with a spatial coordinate $\{x, y\}$ that contains a set of keywords from $T = \{t_1, t_2, ..., t_x\}$. We assume that spatio-textual objects are located at faces in TF. The $DT(P)$ in Figure 4.3 contains five spatio-textual objects that are spread throughout several faces. For example, object o_1 is located inside dt_{v_7,v_8,v_9} and it contains a set of keywords $\{book, supermarket\}$.

wBestPath Given a set of points $P = \{v_1, v_2, ..., v_n\}$ in R^2 space, Delaunay Triangulation $DT(P)$, a set of weights $W = w_{dt_{v_1,v_2,v_3}}, w_{dt_{v_2,v_3,v_4}}..., w_{dt_{v_{n-2},v_{n-1},v_n}}$, $dt \subset DT(P)$, a set of spatio-textual objects O and a set of user queries. The user queries consist of a source location s_l, destination location d_l, and preferable keywords $K = \{k_1, k_2, ..., k_z\}$,

where each k_i, $1 \leq i \leq z$, can be positive (k^+) or negative (k^-). A positive keyword k^+ means that the keyword satisfies what the user wants (e.g. the places user wishes to visit), while a negative keyword k^- means that part of the keyword expresses negative connotation which the user wants to avoid. Having all this information, we want to find the Best Path $wBP(s_l, d_l, K)$ that establishes a shortest path from s_l to d_l while passing through all k^+ and avoid k^- in a given weighted regions model.

Table 4.1 presents the list of mathematical notations used in this chapter.

Table 4.1: List of notations used in the chapter

Notation	**Definition**
$DT(P)$	Weighted regions
c_i	Coordinate location of vertex v_i
dt_{v_{n-2},v_{n-1},v_n}	Triangle face formed by v_{n-2}, v_{n-1}, v_n
$w_{dt_{v_{n-2},v_{n-1},v_n}}$	Weight of triangle face dt_{v_{n-2},v_{n-1},v_n}
$\delta(u, v)$	Shortest path between u and v
$dist(u, v)$	Distance between u and v
s_l, d_l	Source location, destination location
K	Query keywords given by user
k^+, k^-	Positive keyword, negative keyword
$wBP(s_l, d_l, K)$	The weighted Best Path from s_l to d_l that passes through all k^+ and avoids k^-

4.3 Basic Concepts

In this section, we provide some brief background on *IG-Tree* and *wNeighbor*, which inspire us to develop the proposed solution to wBestPath.

4.3.1 IG-Tree

IG-Tree is an index structure that can be used on various types of Spatial Keywords Query, especially on the Best Path Query on Road Networks environment [137]. It is capable to index both road networks and textual information of objects. The *IG-Tree* itself is a hybrid between IR^2*-Tree* [1] and *G-Tree* [68, 69], where the IR^2*-Tree* is a data structure for indexing Spatial Keywords Queries in Euclidean space, while the *G-Tree* is used for indexing Road Networks. Both of these index structures are combined in *IG-Tree* as they have their own benefits to solve the Best Path Query.

In *IG-Tree*, the road network is partitioned and transformed into a graph-based tree structure. Utilizing the multi-level partitioning algorithm [122], the road network (treated

as a graph) is divided into equal-sized subgraphs. Each partition consists of two or more vertices and makes up one node in the *IG-Tree*. After the graph partition process, vertices that connect one partition to another are marked as borders. Through these borders, the shortest paths between border-to-border are pre-computed and stored in Distance Matrices in order to speed up the path computation in the query processing stage.

Meanwhile the spatio-textual objects are transformed into bitmap inverted index. The two indexes of road network and spatio-textual objects are then combined by attaching the inverted index to the tree (as can be seen in Figure 4.4). The inverted index is attached to its corresponding vertices at the leaf nodes. For each parent node, its inverted index is calculated using logical OR of its child nodes. The Keyword Distance Matrices that computes the distance of the nearest keyword matching vertex from each border are also provided.

As the *IG-Tree* can efficiently index Spatio-Textual data and it can be used to process Best Path Query on road networks, this indexing technique unfortunately is not applicable to *w*BestPath as the space model is different and *w*BestPath has more complex environment to process.

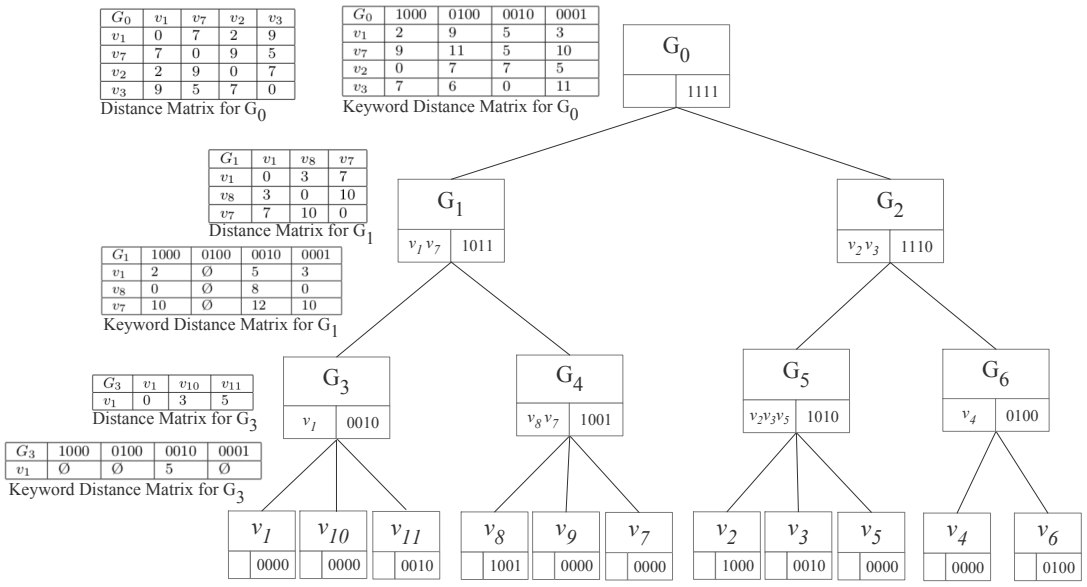


Figure 4.4: IG-Tree

4.3.2 wNeighbor

As previously discussed in Section 2.5, the first work to investigate the k Nearest Neighbour on weighted regions (WkNN) was done by Li *et al.* [121]. The WkNN query is defined as follow. Given a finite triangulation in the space, edge's weight w_e, face's weight w_f, a query point s, a destination point set D, and a value k for the number of objects to be retrieved. WkNN query finds a result set R of the top k nearest neighbours of point s in a destination point set D, that satisfies

$$\{d(x,q) < d(x',q) | x \in R, x' \in D - R, |R| = k\}$$

where $d(\cdot)$ indicates the weighted distance between two points.

In their work, Li *et al.* [121] introduced the concept of Combination Region (CR), which is a combination of region faces that share the same indexing data. The CR can be identified by an area that contains one block face or faces that are connected by combination angles. The block face and combination angles in this case are determined through the characteristics of shortest path in weighted regions, which are as follow.

- If $\cos\beta < 1 - 2/m^2$, then the shortest path between s and t must be through the edges of f. The angle β for this case is called the *block angle* as it can block the shortest path that try to cross the adjacent edges of β. If a face contains three block angles, then it is a *block face*.

- If $\cos\beta < 1 - 2/m^2$ is not satisfied, then there is a possibility that the shortest path between s and t is a direct line. When angle β is not a block angle, it is considered as *combination angle* the shortest path can pass through the adjacent edges of β.

Through the characterisations of shortest path in weighted regions, a new data structure called Weighted Indexing Map (WIM) is proposed by Li *et al.* [121]. This data structure stores $crList$, a list of CRs, and also $pList$, an altered R*-Tree that stores the location of vertices and destination points. An algorithm to answer the WkNN query that utilises WIM, namely wNeighbor algorithm, is also proposed. The wNeighbor adopts both Dijkstra's algorithm [63] and Mitchell's algorithm [116] in its query processing phase. Based on the characterisation of shortest path, if given a query point s in R_s and a destination point d in R_d, R_s is disconnected from R_d, then the shortest path utilises Dijkstra's

algorithm. But if R_s and R_d are adjacent to each other, then the shortest path is calculated using Mitchell's algorithm. Based on the shortest paths between the query point to the destination points, we can easily identify the WkNN result.

The wNeighbor offers outstanding solution to k Nearest Neighbour queries on weighted regions. The solution provided are not fully applicable to wBest Path as wBest Path is dealing with Spatio-Textual queries. However in this study, we adopt the characterisation of shortest path in wNeighbor in developing our techniques.

4.4 Data Index

This section presents the *wIG-Tree*, an indexing technique for planning the Best Path on Weighted Regions (wBestPath).

4.4.1 wIG-Tree

In this study of wBestPath, we devise a new indexing technique called *wIG-Tree* that can be used to process spatial and textual information on weighted regions. The index construction of *wIG-Tree* consists of several components: (1) tree construction, (2) distance matrices computation, (3) face-buckets, (4) keyword index, and (5) keywords distance matrices computation. Some methods from *IG-Tree* are adopted in the creation of *wIG-Tree* (e.g. the tree construction and distance matrices computation), but some modifications are done in order to accommodate the weighted regions environment.

Firstly, the vertices P and edges E in the weighted regions that is denoted by Delaunay Triangulation $DT(P)$ are indexed in a similar manner as the *IG-Tree*. Using the multi-level partitioning algorithm [122], the vertices P and edges E of the weighted regions $DT(P)$ are partitioned into equal-sized subgraphs. Each subgraph partition consists of two or more vertices. Figure 4.5 shows the example of the region partition of the weighted regions given in Figure 4.3. The tree construction is similar to the *IG-Tree* in which every node of the tree is based on the subgraph in the graph partition.

In the example in Figure 4.5, the weighted regions, determined by the Delaunay Triangulation with different coloured-weights, is divided into several partitions. The first partition of G_0 consists of equal-sized subgraph partitions of G_1 and G_2. G_1 and G_2 are also partitioned until the subgraph only consists of at least two vertices. In this case, the

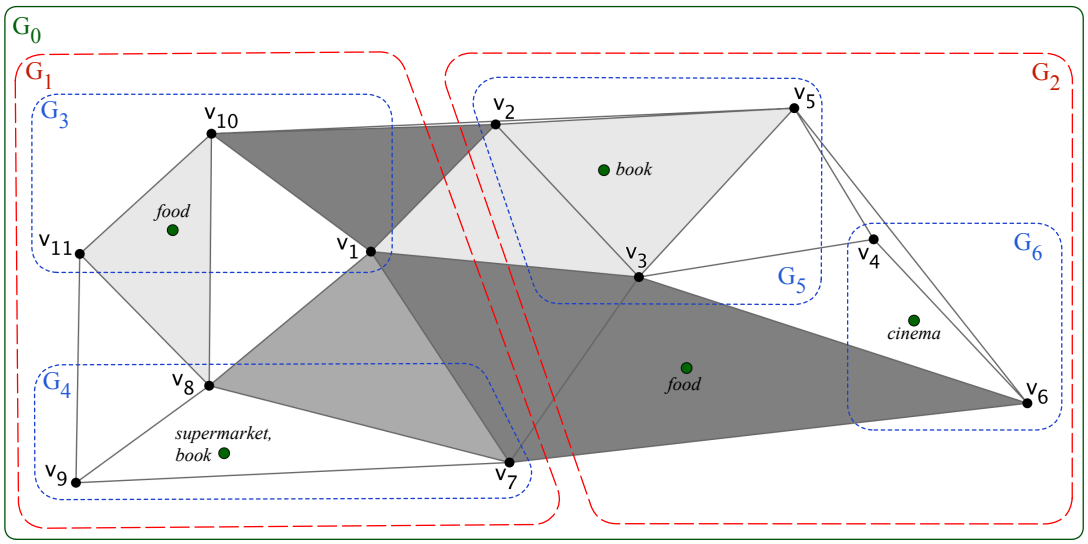


Figure 4.5: Partition

smallest partitions are G_3, G_4, G_5, and G_6. The partition G_3 contains vertices v_1, v_{10} and v_{11}; subgraph partition G_4 contains vertices v_7, v_8 and v_9; subgraph partition G_5 contains vertices v_2, v_3 and v_5; and finally, subgraph partition G_6 contains vertices v_4 and v_6. Each subgraph partition composes a node on the *wIG-Tree*, as shown in Figure 4.6. G_0 is the root and each partition of G_0 are its children nodes, while the vertices are in the leaf nodes.

We also have to maintain the connection between partitions in the tree. The connection is identified by the vertices in a partition whose edges connect to other partition. For example v_{11} in partition G_3 connects the subgraph in partition G_3 to the subgraph in partition G_4 through v_8 and v_9. Thus, v_{11} can be considered as one of the borders of G_3. The other vertices in G_3, specifically v_1 and v_{10} are also borders as they connect subgraph partition G_3 to other borders. So each partition can have more than one border. In this case, the borders of G_3 are $\{v_1, v_{10}, v_{11}\}$; the borders of G_4 are $\{v_7, v_8, v_9\}$; the borders of G_5 are $\{v_2, v_3, v_5\}$; the borders of G_6 are $\{v_4, v_6\}$; the borders of G_1 are $\{v_1, v_7, v_{10}\}$; and the borders of G_5 are $\{v_2, v_3, v_5, v_6\}$. These borders are also recorded in the *wIG-Tree* based on the nodes they belong, which can be seen in Figure 4.6.

Distance Matrices between borders are also pre-computed at this stage. The usage of distance matrices is to store the shortest paths between border-to-border in order to speed up the path computation later in the query processing stage. The calculation of distance on weighted regions is slightly different from the normal road networks or other

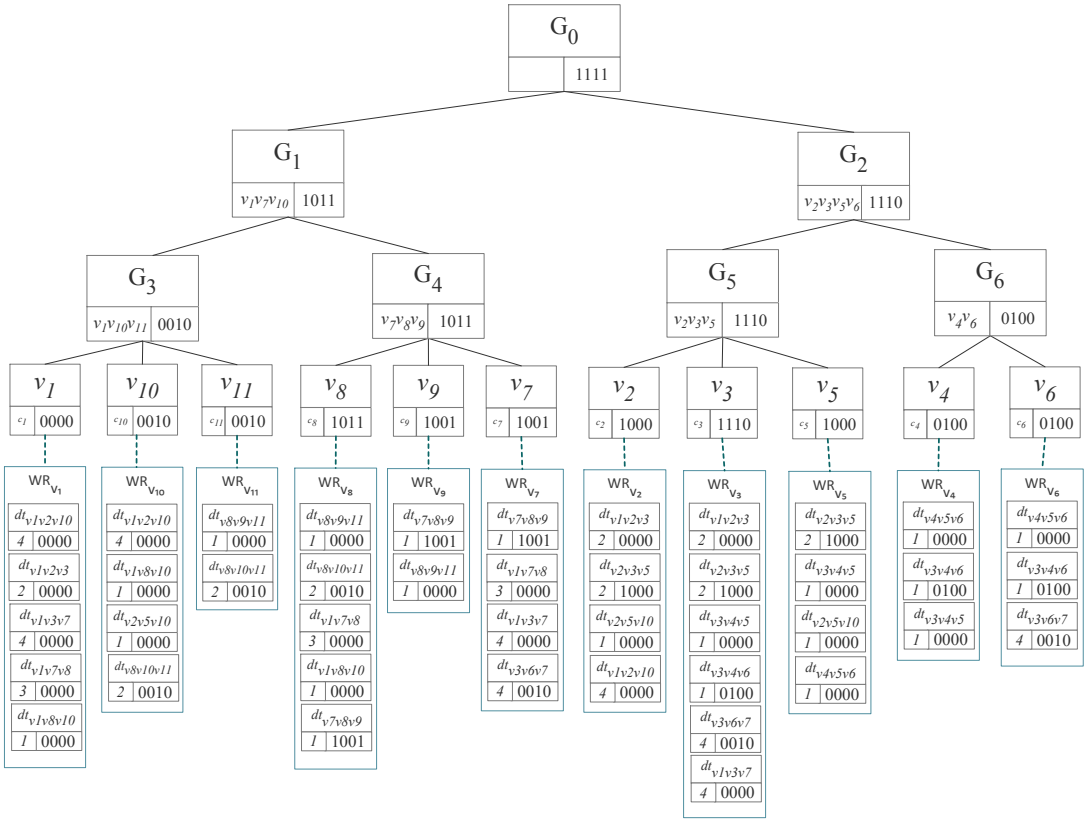


Figure 4.6: wIG-Tree

simple environment model due to the impact of the weight of each region. Following the property of edge weight in [121], the distance indexed in the *wIG-Tree* is based on the product of distance of an edge (x, y) and the minimum weight of the corresponding edge $min\{w_{dt_{x,y,z}}, w_{dt_{x,y,w}}\}$. For example, assume that the distance from v_1 to v_2 is 1, the weight $w_{dt_{v_1,v_2,v_3}}$ is 2, and the weight $w_{dt_{v_1,v_2,v_{10}}}$ is 3. So the distance that will be stored in the distance matrices is the product of $(v_1, v_2) = 1$ and $min\{w_{dt_{v_1,v_2,v_3}}, w_{dt_{v_1,v_2,v_{10}}}\} = 2$, which is $(distance \cdot weight) = 2$. Table 4.2 - Table 4.8 present the distance matrices for each node in *IG-Tree*.

Another important component of weighted region is its faces that hold a certain weight. Moreover in our data model, the spatio-textual objects are located in a triangle face. Thus this information is crucial to be stored inside the *wIG-Tree*. So for each vertex, we identify its corresponding faces as each vertex is a part of at least one triangle face. Note that according to the properties of Delaunay Triangulation, each vertex has an average of six surrounding triangles. So in the *wIG-Tree*, the faces that belong to

Table 4.2: Distance Matrix for G_0

G_0	v_1	v_2	v_3	v_5	v_6	v_7	v_{10}
v_1	0	2	4	4	6	6	1
v_2	2	0	2	2	4	6	8
v_3	2	2	0	1	3	8	10
v_5	4	2	1	0	2	9	4
v_6	6	4	3	2	0	5	6
v_7	6	6	8	9	5	0	7
v_{10}	1	8	10	4	6	7	0

Table 4.3: Distance Matrix for G_1

G_1	v_1	v_7	v_8	v_9	v_{10}	v_{11}
v_1	0	6	1	2	1	3
v_7	6	0	2	3	7	3
v_{10}	1	7	2	3	0	2

Table 4.4: Distance Matrix for G_2

G_2	v_2	v_3	v_4	v_5	v_6
v_2	0	2	3	2	4
v_3	2	0	2	1	3
v_5	2	1	1	0	2
v_6	4	3	1	2	0

Table 4.5: Distance Matrix for G_3

G_3	v_1	v_{10}	v_{11}
v_1	0	1	3
v_{10}	1	0	2
v_{11}	3	2	0

Table 4.6: Distance Matrix for G_4

G_4	v_7	v_8	v_9
v_7	0	2	3
v_8	2	0	1
v_9	3	1	0

Table 4.7: Distance Matrix for G_5

G_5	v_2	v_3	v_5
v_2	0	2	2
v_3	2	0	1
v_5	2	1	0

Table 4.8: Distance Matrix for G_6

G_6	v_4	v_6
v_4	0	1
v_6	1	0

a certain vertex is stored inside a *face-bucket* and it is attached into their corresponding vertices on the leaf node of *wIG-Tree*, together with the information of its weight and which keywords lie inside the triangle face. Figure 4.6 shows an illustration of the face-buckets that holds the weighted regions information and they are attached to its corressponding vertex in the *wIG-Tree*. Vertex v_1 makes up five triangle faces, which are $dt_{v_1,v_2,v_3}, dt_{v_1,v_2,v_{10}}, dt_{v_1,v_3,v_7}, dt_{v_1,v_7,v_8}, dt_{v_1,v_8,v_{10}}$. All these faces are combined into a face-bucket WR_{v_1} and is attached into vertex v_1 in *wIG-Tree*.

As we are also dealing with spatio-textual objects in wBestPath, this information is also kept in the *wIG-Tree* in a bitmap invertex index form. First of all, all the keywords in the space are sorted and assigned a binary value based on its location in each region face. Using the case in Figure 4.5 as an example, there are four keywords available, which are *book*, *cinema*, $food$, and *supermarket*. These keywords are then sorted and we assign a binary value to all the triangle faces based on the availability of the keyword in the face. The keywords are indexed into bitmap inverted index as in Table 4.9. For instance, face dt_{v_7,v_8,v_9} contains keywords book and supermarket, so it is assigned a binary value of 1001. All these inverted indexes are then stored at the face-bucket of *wIG-Tree*. We

also attach the inverted index to the vertices at the leaf nodes and the parent nodes. The inverted index however is calculated using logical OR of its child nodes as to indicate the combination of keywords that can be accessed through those vertices or subgraphs. For example the inverted index of v_8 is calculated using logical OR of the face-bucket WR_{v_8} that contains 0010 at $dt_{v_8,v_{10},v_{11}}$ and 1001 at dt_{v_7,v_8,v_9}, which in this case it produce inverted index of 1011 for vertex v_8. The same calculation is applied to every node in the tree. The index at the root node is usually made up of all 1s.

Table 4.9: Keyword index

Keyword / *Face*	*Book*	*Cinema*	*Food*	*Supermarket*
$dt_{v_8,v_9,v_{11}}$	0	0	0	0
dt_{v_7,v_8,v_9}	1	0	0	1
$dt_{v_8,v_{10},v_{11}}$	0	0	1	0
$dt_{v_1,v_8,v_{10}}$	0	0	0	0
dt_{v_1,v_7,v_8}	0	0	0	0
$dt_{v_1,v_2,v_{10}}$	0	0	0	0
dt_{v_1,v_3,v_7}	0	0	0	0
dt_{v_1,v_2,v_3}	0	0	0	0
$dt_{v_2,v_5,v_{10}}$	0	0	0	0
dt_{v_2,v_3,v_5}	1	0	0	0
dt_{v_3,v_6,v_7}	0	0	1	0
dt_{v_3,v_4,v_6}	0	1	0	0
dt_{v_3,v_4,v_5}	0	0	0	0
dt_{v_4,v_5,v_6}	0	0	0	0

The last component of *wIG-Tree* is the Keyword Distance Matrices, which its purpose is to pre-compute the shortest path from certain vertices to the nearest keywords. The way the Keyword Distance Matrix is stored and calculated in *wIG-Tree* is very different from *IG-Tree*. In *wIG-Tree*, we only store the shortest paths between the keyword with its surrounding vertices, instead of borders. The surrounding vertices in this case are identified through the Combination Region (CR) method that was discussed in Section 4.3.2. So for each keyword matched spatio-textual object, we find its CR. For each CR, we collect the vertices that makes up the corresponding CR. We call all these vertices as the *access points* to the keyword matched spatio-textual object. Through all these access points, we can calculate the shortest paths from each access point to the keyword matched objects that are inside the CR. The shortest path here is calculated using Mitchell's Algorithm [116]. Each shortest path distance is then recorded in the Keywords Distance Matrix.

Different from *IG-Tree* where it keeps the Keyword Distance Matrices in every node, the Keyword Distance Matrices in *wIG-Tree* are only available for the face-buckets.

For example, suppose that we want to build the Keyword Distance Matrix of face-bucket WR_{v_2}. Keyword book, whose inverted index is 1000, is available inside WR_{v_2}. There are four faces inside this face-bucket. However, only one of them contains a spatio-textual object. Therefore we only focus on finding the CR of dt_{v_2,v_3,v_5} where the keyword is located. Assume that the CR for this case comprises faces dt_{v_2,v_3,v_5} and dt_{v_1,v_2,v_3}. Thus, the access points for this CR are v_1, v_2, v_3, v_5. We calculate the shortest paths from v_1, v_2, v_3, v_5 to the keyword inside this CR, which then stored in the distance matrix as shown in Table 4.10.

Table 4.10: Keyword Distance Matrix of face WR_{v_2}

	1000
v_1	3.82
v_2	1.31
v_3	1.88
v_5	2.98

Based on the discussion above, *wIG-Tree* consists of two tiers: *the tree* and *the face-buckets*. The root and children nodes of the tree contain the subgraph partition, borders, and keywords related to the partition. The leaf nodes contain the vertices, together with their coordinate location, and the keywords that can be access through the vertices. The face-buckets maintain the information of the faces related to each vertex. There are also two types of matrices recorded in *wIG-Tree*: *distance matrices* and *keywords distance matrices*. The distance matrices are used to store the shortest distance between borders in the tree tier, while the keywords distance matrices are for storing the shortest distance between access points to the keyword matched spatio-textual objects.

4.5 Negative Query Keywords

The *wIG-Tree* data structure holds a number of information that can help us in answering route planning query on weighted regions. However in the *w*BestPath query, the users are allowed to provide some keywords as input and the keywords given can be both in positive and negative values. The positive keywords indicate the spatio-textual objects that the user wants to visit, while negative keywords indicate the spatio-textual objects

that the user wants to avoid. Based on textual relevancy of the query keywords and the keywords of spatio-textual objects, we can determine which objects we can include to our wBestPath result and which path have to be avoided throughout our route in finding the wBestPath. In the case of finding path with positive keyword objects, the *wIG-Tree* can be utilised and produce significant result to the query. For instance when we want to find the nearest *book* from v_2, we can check the location of v_2 in the *wIG-Tree* and see whether the inverted index of keyword *book* is also located at this node. When the keyword is found in the tree, we can proceed to the keywords distance matrix of WR_{v_2} to get the distance from v_2 to the keyword. However, when it comes to finding a path with negative keywords, the complexity increases as there are paths that we have to avoid.

In the case of negative keywords are detected, the current *wIG-Tree* requires an update. As we do not want to pass any negative keywords, the spatio-textual object that contains the negative keyword is regarded as an obstacle. The region where it is located is also regarded as unpassable area. The unpassable area means that the weight of the area is ∞. Therefore we have to update the *wIG-Tree*, specifically on the distance matrices and keywords distance matrices, to indicate the change in weight of the region face that contains the negative keyword. The procedure of updating the *wIG-Tree* is as follow:

1. If a k^- is detected at one of the nodes of *wIG-Tree*, then trace which region face the k^- is located. So we need to traverse the tree until the face-bucket in order to get the exact location of the k^-.

2. Once the region face dt is located, get the access points (AP) of the spatio-textual object that contains k^-.

3. Update the keywords distance matrix for region face dt to ∞, making it as unpassable area.

4. Check if the other region faces in the same face-bucket of dt has any overlapping access points. For those overlapping access points, they have to be excluded from the list so that we can avoid the possibility of passing through the unpassable area of dt.

5. Traverse back to the tree tier to update the distance matrices because of the unpassable area. Path reconstruction is required for all the tree nodes where the access points AP are part of the border.

The update, however, does not always affect the whole *wIG-Tree*. As the identification of negative keywords only happens in the query processing stage, it depends on the moment when we encounter a negative keyword in a node of the tree. If we encounter one while traversing through the *wIG-Tree*, then:

- If node G_x contains k^- and no k^+ available in this node, then we can block the whole branch of node G_x. By doing this, we have pruned any unnecessary subtree.

- If node G_x contains both k^+ and k^-, then traverse through the children nodes until k^+ and k^- are on different nodes.

  - If k^+ and k^- are on different nodes, the node with k^- can be blocked and its subtree can be pruned.

  - If k^+ and k^- are still in the same node until the leaf node, then we have to check the face-bucket to locate the exact location of k^+ and k^-.

    * If k^+ and k^- are in the same region face, then the weight of the region face has to be updated, which we have to follow the procedure of updating *wIG-Tree* that was explained before. Furthermore, even though there is k^+ in this particular region face, we have to give up the k^+ as it is inside an unpassable area. We have to find other spatio-textual object with k^+ in another area, rather than the one with k^-.

    * If k^+ and k^- are at different faces, then the face where k^- is located is blocked by updating its weight (follow the update procedure). Meanwhile for k^+, some of the access points of the face where k^+ is located might be removed because they overlap with the access point list of k^- (procedure step 4). k^+ however is still accessible through the remaining access points.

4.6 Query Processing

In this section, we discuss our proposed algorithms to solve wBestPath. The algorithms, namely the Minimum Distance Approximation (MDA) and Minimum Path Approximation (MPA), utilises the *wIG-Tree*. A baseline algorithm is also discussed in this section.

4.6.1 Baseline Algorithm

A naive approach to solve the wBestPath is by finding all possible combinations of routes. In this case, we get the permutation of all combinations of positive keywords and then at the end find the route with the least cost among all. This approach will give a precise solution, but the performance is very low. Algorithm 6 shows the pseudo-code of the baseline algorithm.

At the first stage, the query keywords given by the user is indexed to bitmap inverted file K_{if} and also locating where the source location s_l is. If the source location is inside a triangulation face, not at a certain vertex, then we have to find its nearest border v_n and compute the shortest path from s_l to the nearest border v_n by using Mitchell's algorithm. Otherwise if s_l is at a vertex, then we only have to locate the leaf node of the vertex v_n (lines 2-8). The purpose of finding v_n is to locate our starting leaf node in the tree in order to find the wBestPath.

The next step is to scan through all of the spatio-textual objects' access points through the leaf node of *wIG-Tree* to find a set of vertices V_k with keywords that match to k^+ in K_{if} (lines 9-11). The purpose of collecting these access points is to help us in identifying the entrance to the region where our keyword is located. Based on the keyword matched vertices V_k, we can start assembling the possible path combinations through cartesian product of V_k. So for each combination of path, we also have to get the permutation to help computing the path with the least distance (lines 12-13). For example a user wants to find a path with keywords *cinema* and *supermarket*. The inverted index of *cinema* is 0100 and the inverted index of *supermarket* is 0001. While scanning through the *wIG-Tree*, we can find that the access points of 0100 are v_3, v_4, v_6 and the access points of 0001 are v_7, v_8, v_9. So based on these access points, we can find the possible path combinations that can contribute to the result of wBestPath. The possible paths will involve the combinations of $\{v_3, v_7\}$, $\{v_3, v_8\}$, $\{v_3, v_9\}$, $\{v_4, v_7\}$, $\{v_4, v_8\}$, $\{v_4, v_9\}$,

$\{v_6, v_7\}$, $\{v_6, v_8\}$, and $\{v_6, v_9\}$. After we have found all possible combinations through the cartesian product of the keyword matched vertices, we calculate the distance from s_l to the permutation of the possible paths and then at the end finishing at the destination location d_l (lines 14-27). While calculating the paths, we also have to keep track the path with the least cost in order to get the best solution for wBestPath (lines 28-30).

As mentioned earlier, this algorithm can give precise solution. However, this algorithm runs more than 10 hours just to compute wBestPath with 5 query keywords due to the fact that it is an NP-Hard problem. Taking more than 10 hours just to plan one trip is definitely not applicable in our daily life. Therefore in the next two subsections, we discuss our propose solutions to this problem.

Algorithm 6 The Baseline

Input: $wIG\text{-}Tree$, s_l, d_l, K in
Output: $wBestPath(s_l, d_l, K)$ out
1: Index K to bitmap inverted file K_{if};
2: Locate s_l;
3: **if** s_l is inside a face **then**
4: $v_n \leftarrow$ nearest border from s_l;
5: Calculate $\delta(s_l, v_n)$ by Mitchell's algorithm;
6: **else**
7: $v_n \leftarrow leaf(s_l)$;
8: **end if**
9: **for** each $o_i \in O$ **do**
10: $V_k = KeywordMatch(o_i, k^+)$;
11: **end for**
12: $cartesian\_product(V_k)$;
13: **for** each $cart\_product(V_k)$ result cp **do**
14: $Permutation(cp)$;
15: **for** each permutation $perm_x \in Permutation(cp)$ **do**
16: $v_{start} \leftarrow \emptyset$;
17: **for** each vertex v in $perm_x$ **do**
18: **if** $v_{start} = \emptyset$ **then**
19: Find $\delta(v_n, v)$;
20: Find $\delta(v, k_v^+)$;
21: **else**
22: Find $\delta(v_{start}, v)$;
23: Find $\delta(v, k_v^+)$;
24: **end if**
25: $v_{start} = v$;
26: **end for**
27: Find $\delta(v_{start}, d_l)$ by Mitchell's algorithm;
28: **if** current path is the shortest **then**
29: $wBestPath(s_l, d_l, K)$;
30: **end if**
31: **end for**
32: **end for**
33: return $wBestPath(s_l, d_l, K)$;

Algorithm 7 Shortest Path Search $\delta(s_l, d_l)$ on *wIG-Tree*

Input: *wIG-Tree*, s_l, d_l, *inverted file* K_{if} in
Output:$\delta(s_l, d_l)$ out

1: Index K to inverted file K_{if};
2: **if** s_l is inside a face **then**
3: $v_{s_l} \leftarrow$ nearest border from s_l;
4: Calculate $\delta(s_l, v_n)$ by Mitchell's algorithm;
5: **else**
6: $v_{s_l} \leftarrow leaf(s_l)$;
7: **end if**
8: $G_{s_l} = parent(v_{s_l})$;
9: **if** d_l is inside a face **then**
10: $v_{d_l} \leftarrow$ nearest border from d_l;
11: Calculate $\delta(d_l, v_n)$ by Mitchell's algorithm;
12: **else**
13: $v_{d_l} \leftarrow leaf(d_l)$;
14: **end if**
15: $G_{d_l} = parent(v_{d_l})$;
16: **while** $G_{s_l} \neq G_{d_l}$ **do**
17: $b_{s_l} \leftarrow$ nearest border from v_{s_l}
18: **if** G_{s_l} contains k^- **then**
19: Path reconstruction for partition G_{s_l};
20: **end if**
21: $dist(d_l, b_{s_l}) += dist(v_{s_l}, b_{s_l})$;
22: $v_{s_l} = b_{s_l}$
23: $b_{d_l} \leftarrow$ nearest border from v_{d_l}
24: **if** G_{d_l} contains k^- **then**
25: Path reconstruction for partition G_{d_l};
26: **end if**
27: $dist(d_l, b_{d_l}) += dist(v_{d_l}, b_{d_l})$;
28: $v_{d_l} = b_{d_l}$;
29: $G_{s_l} = parent(v_{s_l})$;
30: $G_{d_l} = parent(v_{d_l})$;
31: **end while**
32: **if** $G_{s_l} == G_{d_l}$ **then**
33: $dist(s_l, d_l) = dist(s_l, b_{s_l}) + dist(d_l, b_{d_l}) + dist(v_{s_l}, v_{d_l})$;
34: return $\delta(s_l, d_l)$;
35: **end if**

4.6.2 Minimum Distance Approximation (MDA) Algorithm

Due to the low performance of the Baseline Algorithm, which runs in non-polynomial time, we propose an approximate solution that runs faster than the baseline, which we call Minimum Distance Approximation (MDA) Algorithm. The MDA algorithm is divided into three phases. The first phase is the query input initialisation. The second phase is the process of collecting candidate objects that match with the query keywords. Then the third phase is the best path construction based on the candidate objects from the second phase. Algorithm 8 describes the pseudo-code of our MDA algorithm.

At the beginning of the first phase, all of the query input, which consists of a source location s_l, a destination location d_l, and a set of query keywords K, are initialised in

order to prepare them for the path construction process. The query keywords K are transformed into bitmap inverted file K_{if} (line 1). We also have to locate the position of s_l and d_l (lines 2-13). If both s_l and d_l are inside a region face, not at a vertex, then we have to find the nearest border from s_l and d_l. The nearest border from s_l is marked as v_{s_l} (line 3), while the nearest border from d_l is marked as v_{d_l} (line 9). Otherwise if s_l or d_l is located at a vertex, that particular vertex will be our starting point v_{s_l} (for s_l) or destination point v_{d_l} (for d_l).

Once all of the query inputs are processed, then we can move to the next phase to start collecting the candidate objects. Candidate objects are spatio-textual objects which contain the keywords that match with one of the keywords in K and has the best location from s_l and d_l. The best location here means that among all spatio-textual objects with a particular keyword k_z, the spatio-textual object o_n has the most optimum cost when it is paired up with s_l and d_l. So the cost of constructing a path $s_l \rightarrow o_n \rightarrow d_l$ is the shortest compare to the other objects. The idea behind this algorithm comes from the fact that route planning with only one category can be solved in polynomial time.

Lines 14-24 of Algorithm 8 shows the second phase of MDA algorithm. So for each keyword k_n in the query keyword set K, we find the best path between $s_l \rightarrow k_n \rightarrow d_l$. The computation to find the best path is through the calculation of shortest path from s_l to k_n and then from k_n to d_l. The shortest path algorithm that we use in *wIG-Tree* is shown in Algorithm 7. From all the possible paths for each keyword, we find the one with the minimum distance, which then the spatio-textual object that contains k_n becomes a candidate point. This candidate point is then stored into a queue Q_k. Once we have found the best path for all of the keywords in K, we have all the candidate objects locations that will help us establish the best path on weighted regions.

In the last phase of MDA algorithm (lines 25-35), our method is to progressively adding the next nearest neighbour of the last point we visit from a list of candidate objects. In this case, we start the path construction from s_l and then find the nearest candidate object that we previously stored in Q_k (line 27). After the first nearest neighbour Q_kn is found, we establish the shortest path between s_l to Q_kn (line 28) and then dequeue Q_kn from the candidate object list Q_k (line 33). The Q_kn then become our new starting point where we find the next nearest neighbour of Q_kn from the candidate list Q_k and repeat the same process until all of the candidate objects have been visited (Q_k is empty). At the end, we

have a path from s_l to all of the spatio-textual objects that contains the query keywords in K. Then from this point, we only have to find the path from the last visited object to the destination point d_l, which complete the wBestPath search.

The MDA algorithm performs better than the baseline algorithm in terms of the running time. However the downside for this algorithm is on the accuracy in which the accuracy is compromised in order to optimise the runtime. The result is not always 100% accurate, especially when the query involved is large.

Algorithm 8 MDA Algorithm

Input: $wIG\text{-}Tree$, s_l, d_l, K in
Output: $wBestPath(s_l, d_l, K)$ out
1: Index K to inverted file K_{if};
2: **if** s_l is inside a face **then**
3: $v_{s_l} \leftarrow$ nearest border from s_l;
4: Calculate $\delta(s_l, v_n)$ by Mitchell's algorithm;
5: **else**
6: $v_{s_l} \leftarrow leaf(s_l)$;
7: **end if**
8: **if** d_l is inside a face **then**
9: $v_{d_l} \leftarrow$ nearest border from d_l;
10: Calculate $\delta(d_l, v_n)$ by Mitchell's algorithm;
11: **else**
12: $v_{d_l} \leftarrow leaf(d_l)$;
13: **end if**
14: **for** each inverted keyword k_n in K_{if} **do**
15: **if** k_n+ **then**
16: $\delta(v_{s_l}, k_n)$;
17: $\delta(k_n, v_{d_l})$;
18: $dist(s_l, d_l) = dist(s_l, v_{s_l}) + dist(v_{s_l}, k_n) + dist(k_n, v_{d_l}) + dist(k_n, d_l)$;
19: **end if**
20: **if** $dist(s_l, d_l) < minDist$ **then**
21: Store (k_n) location to queue Q_k;
22: $minDist = dist(s_l, d_l)$;
23: **end if**
24: **end for**
25: **while** $Q_k \neq \emptyset$ **do**
26: **if** current path is 0 **then**
27: Find NN(s_l, Q_k);
28: Find shortest path $\delta(s_l, Q_k n)$;
29: **else**
30: Find NN$(Q_k n - 1, Q_k)$;
31: Find shortest path $\delta(Q_k n - 1, Q_k n)$;
32: **end if**
33: Dequeue $Q_k n$ from Q_k;
34: **end while**
35: Find shortest path $\delta(Q_k n, d_l)$;

4.6.3 Minimum Path Approximation (MPA) Algorithm

Another approximation algorithm that we proposed in this study is Minimum Path Approximation (MPA) Algorithm. The idea behind this algorithm is to minimize the path traversal by doing expansion to the spatio-textual objects while computing the path between source and destination points. Algorithm 9 presents the pseudo-code of the MPA algorithm. The MPA algorithm consists of three phases. The first phase is the initialisation of query input (lines 1-13). The second phase is to find the common ancestor of the source point s_l and destination point d_l and also checking the ancestor node whose all keywords match with the query keywords K (lines 14-18). The last phase is the best path formation and expansion (lines 19-32).

The first phase for this algorithm (lines 1-13) is similar to the MDA algorithm where we want to locate the position of the source point s_l and destination point d_l. If s_l and d_l are inside a region face, then we have to find their nearest borders (v_{s_l} and v_{d_l}) and calculate the shortest path of $\delta(s_l, v_{s_l})$ and $\delta(d_l, v_{d_l})$ by using Mitchell's algorithm. If s_l and d_l are at certain vertices, then these vertices are the starting point v_{s_l} for s_l and destination point v_{d_l} for d_l.

In the second phase, we are looking for an ancestor node in *wIG-Tree* that is the common ancestor node between s_l and d_l and also contains all relevant keywords of the query input. Thus in line 14 of Algorithm 9, the algorithm attempts to find the common ancestor node G_a between s_l and d_l. Then the next step is to check whether all query keywords K_{if} are available in the node G_a. If some of the keywords in K_{if} are not available in G_a, then we have to check the parent node of G_a. We keep repeating this step until all keywords are found (lines 15-18). At the end of this phase, we will have an ancestor node that contains all necessary keywords for the query and we can also establish a path between s_l and d_l. The reason of getting this ancestor node is to limit our traversal space to the subtree/branch of this ancestor node, which means that the nodes outside this ancestor node's subtree are pruned.

The last phase of MPA algorithm is where we construct the wBestPath. In this phase, we start our route construction from s_l and finish at d_l while expanding the path in between these two points in order to find the query keywords. So in lines 19-20 of Algorithm 9, we initialise the current point (v_{curr}) that we are visiting, which starts at s_l, and also the parent node (G_{s_l}) of our current point. Starting our route search, we have to check

whether G_{s_l} contains any keyword from K_{if} (line 22). If there is matching keyword k_n in G_{s_l}, then we have to traverse the children nodes of G_{s_l} to where k_n is located and find the shortest path from s_l to k_n (lines 23-25). Once the shortest path between s_l to k_n is found, the keyword k_n is removed from K_{if} (line 27). So now our currently visited point is the location of k_n (line 26). We repeat the same step where we have to check whether the current visited node G_{s_l} contains any keyword in K_{if}. If there is no matching keyword, then we move to the parent node of G_{s_l} in order to find any matching keyword (line 29). We keep traversing the tree until K_{if} is empty (all query keywords are found). At the end when the path from s_l to all the location of query keywords in K_{if} is establish, we can finish the path from the last visited keyword to destination point d_l (line 32).

Algorithm 9 MPA Algorithm

Input: $wIG\text{-}Tree$, s_l, d_l, K in
Output: $wBestPath(s_l, d_l, K)$ out
1: Index K to inverted file K_{if};
2: **if** s_l is inside a face **then**
3: $v_{s_l} \leftarrow$ nearest border from s_l;
4: Calculate $\delta(s_l, v_n)$ by Mitchell's algorithm;
5: **else**
6: $v_{s_l} \leftarrow leaf(s_l)$;
7: **end if**
8: **if** d_l is inside a face **then**
9: $v_{d_l} \leftarrow$ nearest border from d_l;
10: Calculate $\delta(d_l, v_n)$ by Mitchell's algorithm;
11: **else**
12: $v_{d_l} \leftarrow leaf(d_l)$;
13: **end if**
14: $G_a \leftarrow commonAncestor(s_l, d_l)$;
15: Check if G_a contains all keywords in K_{if};
16: **while** $G_a \oplus K_{if} = 1$ **do**
17: $G_a = parent(G_a)$;
18: **end while**
19: $v_{curr} \leftarrow s_l$;
20: $G_{s_l} \leftarrow parent(s_l)$;
21: **while** $G_{s_l} \neq parent(G_a)$ and $K_{if} \neq \emptyset$ **do**
22: **if** $G_{s_l} \oplus K_{if} = 0$ **then**
23: $k_n \leftarrow$ matched keyword;
24: $v_{k_n} \leftarrow leaf(k_n)$;
25: $\delta(v_{curr}, v_{k_n})$;
26: $v_{curr} = v_{k_n}$;
27: $K_{if} = K_{if} \setminus k_n$;
28: **else**
29: $G_{s_l} = parent(G_{s_l})$
30: **end if**
31: **end while**
32: Find shortest path $\delta(v_{curr}, d_l)$;

The MPA algorithm offers some advantages and also drawbacks. The number of visited vertices are generally lesser than MDA but it does not guarantee the minimum distance. The accuracy of MPA itself can be lower than MDA. The main advantage of MPA algorithm, however, is in the early pruning of nodes, which narrow down the path traversal to a smaller subtree and avoiding unnecessary traversal to other tree branches that can cause longer running time.

4.7 Experiment

This section provides the detailed experimental evaluations of our proposed solutions to the *w*BestPath. We evaluate the performance of the *wIG-Tree* indexing scheme and we also compare the efficiency and accuracy of the algorithms from Section 4.6, which are the Baseline Algorithm, Minimum Distance Approximation (MDA) Algorithm, and Minimum Path Approximation (MPA) Algorithm.

4.7.1 Settings

Environment The experiments are conducted on 2.5 GHz Intel Core i7-4870HQ CPU and 12 GB RAM running 64-bit Ubuntu. All of the algorithms were written in single-threaded C++.

Datasets We use three datasets of Computational Fluid Dynamics (CFD) Dataset from [140, 141, 142]. The details of these CFD datasets are provided in Table 4.11. The weights of the regions are synthetically generated. The keyword information is produced using the keyword sets from [125]. Then each keyword object is mapped into a certain region by randomly assigning the x and y coordinate of the object inside the weighted regions. In order to detect the negative keywords given by the users, we utilize some negative sentiment analysis based on words from [126, 127] to help us in identifying certain words that can be considered as negative words.

Table 4.11: CFD Datasets

Dataset	**# Vertices**	**# Edges**	**# Triangles**
CFD-1	5,088	58,554	9,759
CFD-2	52,510	621,984	103,664
CFD-3	208,688	2,487,936	414,656

Queries For the experiments, we generate the keyword set K for the test queries with a random distribution from a keyword pool. We vary the size of K in the test queries from 1 to 15. We also set the size of object density of K from 1% to 30% of the whole datasets. The object density here means the number of spatio-textual objects for each query keyword in comparison to the number of vertices in the space. Furthermore, we also evaluate the impact of varying the distance between s_l and d_l. In this experiment, the distance is varied from 2% up to 64% of the maximum distance between s_l and d_l.

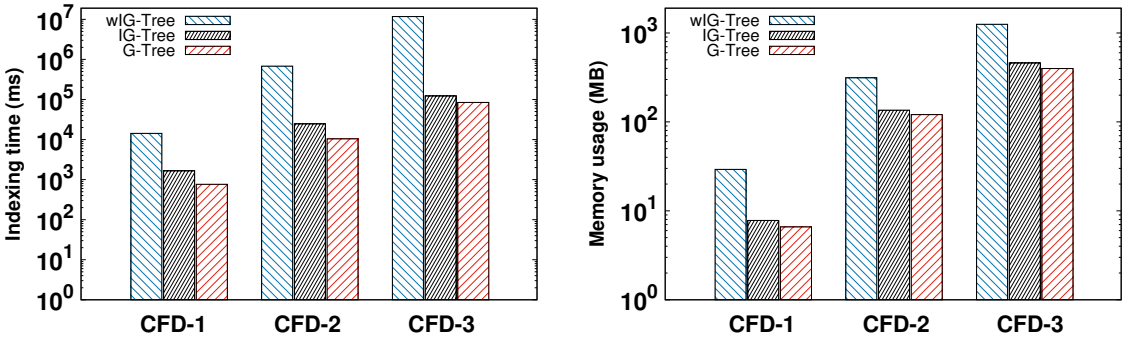
4.7.2 Index Evaluation

We evaluate our proposed indexing technique, *wIG-Tree*, in terms of index building time and index size. As the *wIG-Tree* has no competitor yet, we compare *wIG-Tree* with *IG-Tree* and *G-Tree*, which are the predecessors of *wIG-Tree*. Even though *IG-Tree* and *G-Tree* are the predecessors of *wIG-Tree*, there are some information that *wIG-Tree* has but both *IG-Tree* and *G-Tree* do not have.

Figure 4.7a presents the index running time for each method to build the indices for each given dataset. This figure shows that *wIG-Tree* takes the longest time to be built in comparison to the *IG-Tree* and *G-Tree*. It does make sense as the *wIG-Tree* has more data to be indexed (e.g. weight of the region) and also involves different computation in terms of the path construction when we compare this index compared to the other two indexes. The *wIG-Tree* may look a lot slower when it is compared to the *G-Tree*, but *G-Tree* unfortunately does not index any spatio-textual objects.

The same trend also applies for the index size, which is shown in Figure 4.7b. The *wIG-Tree* has higher space consumption due to the fact that we need to store more information. The *wIG-Tree* involves indexing weighted regions in which the information of each region has to be stored inside additional face-buckets to the tree, where the *IG-Tree* and *G-Tree* do not have this kind of information.

Based on these evaluations, the *wIG-Tree* itself is still comparable with the other two predecessors, especially for the fact that *wIG-Tree* incorporate more data of weighted regions.



(a) Building time

(b) Index size

Figure 4.7: Index building time and size on CFD datasets

4.7.3 Performance Study

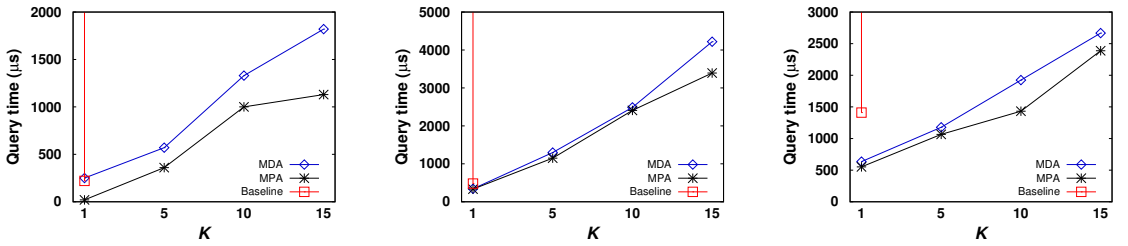
Here, we present the efficiency study results of our query processing algorithms on two metrics, which are on the running time and approximation accuracy. The approximation accuracy presents the percentage of the result accuracy produced by each algorithm in comparison to the expected correct result that is obtained by running the baseline algorithm.

Effect of k^+

This section investigates the effect of the positive keywords (k^+) that are given as the query input by users. As can be seen in Figure 4.8, the MPA algorithm has better running time compared to both MDA and the baseline algorithms. Even though we increase the number of k^+, MPA still performs the best. The baseline algorithm itself performs the worst compared to the other two algorithms due to the fact that it has to compute all possible combination of routes. When the number of K increases to 5, it takes more than 10 hours to finish the baseline algorithm, which is far from acceptable.

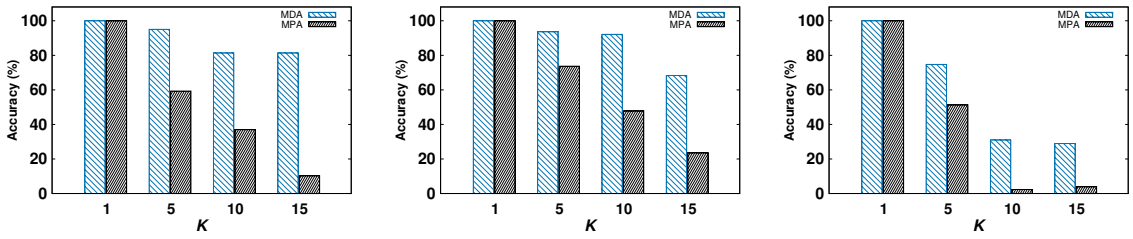
In terms of accuracy, MDA performs better than MPA, as shown in Figure 4.9. The reason for this is because MDA finds the best location for each keyword in order to minimize the total distance. However, MPA focuses on finding lesser route and limits its working space to a smaller subtree, which impacted on the accuracy. As can be seen in Figure 4.9, the higher the number of K and also the bigger the datasets, the lower the accuracy of MPA algorithm. So even though MPA performs well in terms of its running time, it has a drawback in its accuracy.

Based on the discussion above, we can say that MDA has better overall performance compared to the baseline algorithm and MPA algorithm. Even though MPA beats MDA for running time, MDA is still comparable to MPA as the time gap is not that much and it actually gives more accurate results than MPA. But if we are just dealing with smaller data and query sizes, MPA may still offer good results to the wBestPath. So depending on our goal, both MDA and MPA have their own advantages.



(a) Running time for CFD-1 (b) Running time for CFD-2 (c) Running time for CFD-3

Figure 4.8: Query performance with all positive query keywords



(a) Approximation accuracy for CFD-1 (b) Approximation accuracy for CFD-2 (c) Approximation accuracy for CFD-3

Figure 4.9: Approximation accuracy for all positive query keywords

Effect of k^-

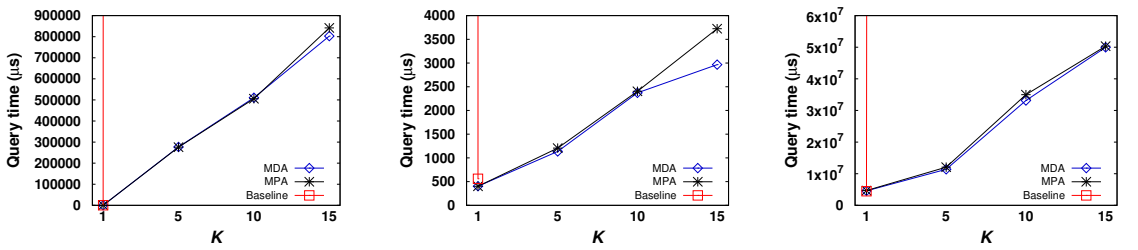
In our experiment, we want to evaluate the impact of having some negative keywords (k^-) as part of our query. The (k^-) itself has a significant impact to our result as it may also forbid us to retrieve any results. So in this case, the distribution of k^- in our test queries is always maintained to be lesser than the number of k^+ for this particular experiment in order to guarantee that the results can be retrieved.

Figure 4.10 shows the running time of each algorithm when the test query contains some negative keywords. The running time for MPA is surprisingly similar to MDA and sometimes MPA is even slower than MDA, which is very different from the previous case

where MPA always outperforms MDA in terms of their query processing running time. The baseline algorithm is always the worst compared to the MDA and MPA.

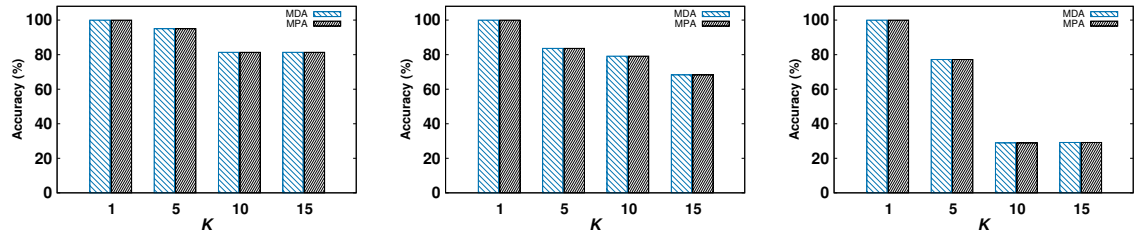
The accuracy evaluation on queries that involve k^- is presented in Figure 4.11. Based on our experiments, the accuracy for algorithms MDA and MPA are the same despite the big accuracy gap in the previous case (Figure 4.9). The result is quite surprising but it is however understandable because of the path blockage when the trip reaches any negative keywords. The path blockage may have caused similar route to be chosen for these two algorithms.

Overall, both MDA and MPA algorithms offer similar performance in terms of their running time and accuracy when we are dealing with queries that contain some negative keywords (k^-).



(a) Running time for CFD-1 (b) Running time for CFD-2 (c) Running time for CFD-3

Figure 4.10: Query performance with combination of positive and negative query keywords



(a) Approximation accuracy for CFD-1 (b) Approximation accuracy for CFD-2 (c) Approximation accuracy for CFD-3

Figure 4.11: Approximation accuracy for datasets with negative keywords

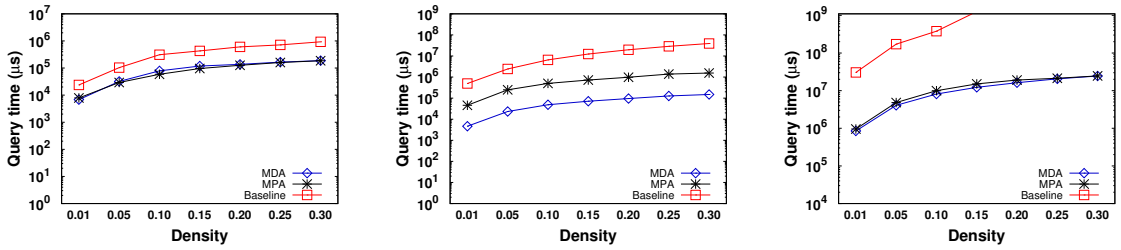
Effect of keyword densities

As previously mentioned in our experiment settings (Section 4.7.1), we want to observe the query performance based on the keyword densities. The density here means for the total number of vertices in the space, the number of spatio-textual objects for each query keyword in the space is based on the percentage of the total number of vertices. For

example, the density of spatio-textual objects for 100 vertices is 10%, which means that there are 10 objects that match each query keyword in the space.

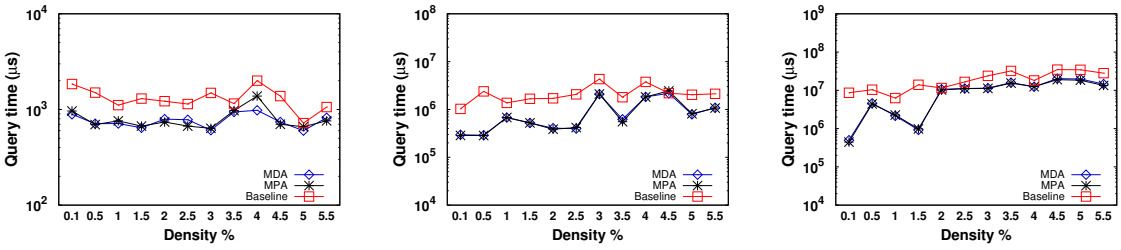
Figure 4.12 shows the query performance of the spatio-textual objects density for all positive keywords. We set the density for this particular experiment from 1% to 30%. As shown in Figure 4.12, the running time of MPA algorithm is usually around 1 order of magnitude compared to the baseline. The running time MDA algorithm itself can outperform both MPA and baseline, especially shown in Figure 4.12b.

Meanwhile, Figure 4.13 shows the query performance of the spatio-textual objects density for all negative keywords. We only set the density up to 5.5% by the reason of having higher density of negative keywords will return no path/no result at all. Based on the result in Figure 4.13, the running time trend of both MDA and MPA algorithms are similar. Compared to the baseline algorithm, the MDA and MPA are still better in performance even though some cases the baseline seems to have similar runtime as seen in Figure 4.13c.



(a) Running time for CFD-1 (b) Running time for CFD-2 (c) Running time for CFD-3

Figure 4.12: Query performance based on keyword density



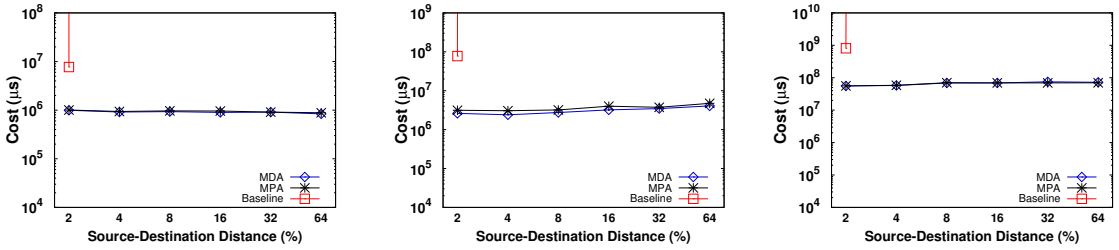
(a) Running time for CFD-1 (b) Running time for CFD-2 (c) Running time for CFD-3

Figure 4.13: Query performance based on keyword density with all negative keywords

Effect of distance between s_l and d_l

Here, we study the effect of varying the distance between source location s_l and destination location d_l pairs. The distance between s_l and d_l are varied from 2%, 4%, 8%, 16%, 32%,

up to 64% of the maximum distance between the two points in each dataset. We set $|K| = 15$ by default and K contains both positive and negative keywords (k^+, k^-). Figure 4.14 shows the experimental results of the source-destination pair distance. The results for both MDA and MPA algorithms run in constant time despite the increase in the distance range. The baseline algorithm however increases significantly, which is why we only include the baseline results for 2% in Figure 4.14.



(a) Running time for CFD-1 (b) Running time for CFD-2 (c) Running time for CFD-3

Figure 4.14: Effect of varying distance between s_l and d_l

Summary

In this experimental study, we evaluated the baseline algorithm, MDA algorithm, and MPA algorithm. Each algorithm is evaluated through different cases and from these evaluations we can see that each algorithm has its own strength and weaknesses. For the baseline algorithm, it offers precise solution but the main problem is in its runtime. The running time exponentially increases in general when the queries and datasets increase. The MDA algorithm is generally better than the baseline algorithm in terms of query processing time, but the accuracy is compromised in order to boost the runtime. Meanwhile for the MPA algorithm, there are times when it excels both MDA and baseline algorithms in terms of running time, such as in the case where we evaluate the effect of k^+. But there are also times where MPA algorithm follows similar trend to MDA. In terms of accuracy, MPA does not offer the best accuracy compared to MDA. Even though there are some cases where the baseline algorithm can compete with the runtime of both MDA and MPA, the baseline algorithm is still not the best solution to wBestPath. So in general, the MDA algorithm offers the best solution based on the overall query processing runtime and accuracy.

4.8 Conclusion

In this chapter, we study a new Spatio-Textual Query on a whole different environment than the traditional spatial queries, which is the wBestPath. The wBestPath is a variant of route planning query that returns a shortest path from a source point to a destination point while visiting a number of positive keywords and avoiding negative keywords in a weighted regions space model. We proposed a new indexing technique for Spatio-Textual queries on weighted regions called *wIG-Tree*. We also proposed two approximation algorithms with different trade-offs that exploit the *wIG-Tree* in order to solve the wBestPath problem, namely the Minimum Distance Approximation (MDA) Algorithm and Minimum Path Approximation (MPA) Algorithm. The results of our extensive experiments demonstrate the effectiveness and efficiency of our proposed solutions.

Chapter 5

Final Remarks

5.1 Overview

This chapter provides an overall conclusion of our research. Section 5.2 presents the highlights of the research contributions in this PhD thesis. Some possible future works on this research topic is discussed in Section 5.3.

5.2 Conclusion

In this thesis, we conduct a research on route planning in Spatio-Textual context. We propose a new Spatial Keywords Query, namely the **Best Path Query**. The Best Path Query is a variant of route planning query that also handles negative keywords which these negative keywords can be considered as obstacles. Our research derived from the fact that there are very limited studies on route planning in Spatio-Textual area and no existing work in this area that considers any negative keywords as part of the query. With the rise of demand in Geographical Information System (GIS), we realise that we have to be able to handle not only complex user queries, but also different types of environment that the user uses. The existing works also are more centred to Euclidean space. Therefore we conducted our study on two different environments, which are *Road Networks* and *Weighted Regions*.

In Chapter 2, we explore some existing queries in Spatial Databases area, which are the basis of our research, and we provide some discussions on the limitations of the current works. Through our exploration, we learn that Spatial Queries can be categorised according to the working space models. The existing research on the Spatial Databases

area are mostly focus on the Euclidean space model and Road Networks model. Each space model can affect the way we solve a certain spatial query, hence we also discuss various techniques that have been developed for spatial queries in each space model. Not only the basic queries, we also analyse the different types of Spatial Keywords Queries and current indexing techniques in this area of study. In this Chapter too, some reviews on the Route Planning Queries are provided as our main research focus is on the combination of both Spatial Keywords and Route Planning Queries. Furthermore, as we mentioned earlier, the main focus on the existing studies are mostly centred on Euclidean space and Road Networks. Therefore, we explore another space model that represents the real earth surface, which is the Weighted Regions. The existing studies on this particular space model is typically on shortest path query. There is only one existing study on spatial query in a weighted regions model, which is the wNeighbor. We review both of these types of weighted regions query and their solutions in this sub-chapter, in which help us in developing our research.

In Chapter 3, we focus our study on Best Path on Road Networks. In this case, given a source location s_l, a destination location d_l, and a set of keywords $K = \{k_1, k_2, ..., k_n\}$, where each k_i for $1 \leq i \leq n$ can be positive (denoted by k^+) or negative (denoted by k^-), in a road network G, find the Best Path from s_l to d_l, denoted by $BP(s_l, d_l, K)$, that passes through all k^+ and avoid all k^- with the most optimum cost. In this chapter, we showed that our Best Path Query is an *NP-Hard* problem. Our solution towards the Best Path Query consists of three approximation algorithms, namely the Optimal Distance Approximation Search, Ancestor Priority Search, and the Euclidean-based Approximation solution. Each of these algorithms has its own strengths and weaknesses. These algorithms also utilise our newly proposed index structure, called the *IG-Tree*. Our experimental evaluation shows that the *IG-Tree* is comparable to its predecessor technique, which is the *G-Tree*. We also evaluated the effectiveness and efficiency of each of our proposed algorithm in this chapter.

In Chapter 4, we extend our research on Best Path problem to a more complex space model, which is the Weighted Regions. We conducted some studies on the existing works on the weighted regions and despite the limitation of current works in Spatial Queries on Weighted Regions, we successfully designed a new indexing technique and algorithms to answer the Best Path Query. To the best of our knowledge, we are the first to propose a

solution on spatio-textual route planning query in weighted regions model. The definition of Best Path problem on Weighted Regions is as follow. Given a weighted regions space $DT(P)$, where each region dt_i is assigned with a certain weight w_{dt_i} ($w_{dt_i} \in [1, +\infty]$), a set of spatio-textual objects O, and a set of user queries that consist of a source location s_l, a destination location d_l, and a set of keywords $K = \{k_1, k_2, ..., k_n\}$, where each k_i for $1 \leq i \leq n$ can be positive (denoted by k^+) or negative (denoted by k^-), find the Best Path from s_l to d_l, denoted by $wBestPath(s_l, d_l, K)$, that passes through all k^+ and avoid all k^- on $DT(P)$ with the least amount of travel cost. So in this chapter, we propose a new indexing technique called the *wIG-Tree*, which incorporates the weighted regions with spatio-textual objects. And utilising this indexing scheme, we develop two algorithms to answer the Best Path Query on weighted regions, which are the Minimum Distance Approximation (MDA) Algorithm and Minimum Path Approximation (MPA) Algorithm. Some extensive experiments to evaluate the accuracy and efficiency of our proposed solutions are also included in this chapter.

5.3 Limitations and Future Research

There are still some improvements that can be done to our current work. Thus, some possible directions for future works are as follow:

- Our main focus right now is on finding the best route with all possible combinations of keywords. However, having all possible combinations increases the processing time of our query. An interesting idea is to find a clustered area that consist of the objects with the given positive keywords. It may have longer distance in total compared to the current solution, but the distance for all the objects might be shorter as all are clustered together in one area.

- Currently, our solution to the Best Path Query is limited to exact keyword matching. There are a lot of times that we are unable to retrieve any Best Path if the keywords that the user provide are not available in the index. For future work, a Query Expansion for the keywords will be needed. The Query Expansion here is to expand the keywords given by the user to other related keywords. For example when we give a keyword *burger* but there is no such keyword on the space, then we can find

other related keyword to *burger*, such as *fast-food*. Thus our query will be expanded and will return a result.

In this Query Expansion we have to consider the synonyms and other related concepts of each word. A special index for the keywords must be generated in the beginning to help the searching. The index will be in a hierarchical form where each word will be grouped into their own category. It will consider the hyponymy and hypernymy of each keyword.

- In our second objective, the weighted regions are simplified due to the complexity of the environment. The method for calculating the best path right now is through the adaptation of both Euclidean and road networks. It can still be optimised in order to increase the path planning efficiency and accuracy. The indexing technique is also able to be optimised by combining similar regions together to decrease the amount of information that we record.

- The current *IG-Tree* and *wIG-Tree* techniques utilises bitmap inverted file to index the textual information of the spatio-textual objects in the space. However, the bitmap inverted index only allows exact match to the keywords given. Thus, we can further improve the *IG-Tree* and *wIG-Tree* indices to include keyword scoring function in order to increase the keyword search accuracy.

References

[1] I. De Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pp. 656–665, IEEE, 2008.

[2] J. B. Rocha-Junior and K. Nørvåg, "Top-k spatial keyword queries on road networks," in *Proceedings of the 15th international conference on extending database technology*, pp. 168–179, ACM, 2012.

[3] esri, "Arcgis desktop." `http://desktop.arcgis.com/en/`, accessed 2019-02-18.

[4] R. Ramakrishnan and J. Gehrke, *Database Management Systems*. New York, NY, USA: McGraw-Hill, Inc., 2nd ed., 2000.

[5] R. H. Güting, "An introduction to spatial database systems," *The VLDB Journal*, vol. 3, pp. 357–399, Oct. 1994.

[6] K. M. Adhinugraha, D. Taniar, and M. Indrawan, "Finding reverse nearest neighbors by region," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 5, pp. 1142–1156, 2014.

[7] K. Hwang and S. Cho, "A lifelog browser for visualization and search of mobile everyday-life," *Mobile Information Systems*, vol. 10, no. 3, pp. 243–258, 2014.

[8] A. B. Waluyo, B. Srinivasan, and D. Taniar, "Research in mobile database query optimization and processing," *Mob. Inf. Syst.*, vol. 1, pp. 225–252, Dec. 2005.

[9] A. B. Waluyo, D. Taniar, W. Rahayu, and B. Srinivasan, "Mobile service oriented architectures for nn-queries," *J. Netw. Comput. Appl.*, vol. 32, pp. 434–447, Mar. 2009.

[10] I. Yairi and S. Igi, "Mobility support gis with universal-designed data of barrier/barrier-free terrains and facilities for all pedestrians including the elderly and the disabled," in *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, vol. 4, pp. 2909–2914, Oct 2006.

[11] R. Zhong, J. Fan, G. Li, K.-L. Tan, and L. Zhou, "Location-aware instant search," in *ACM CIKM*, pp. 385–394, 2012.

[12] Y. Li, D. Wu, J. Xu, B. Choi, and W. Su, "Spatial-aware interest group queries in location-based social networks," *Data & Knowledge Engineering*, vol. 92, no. Supplement C, pp. 20 – 38, 2014.

[13] S. B. R. Institute, "Mobile browser vs application preferences." `http://www.statisticbrain.com/mobile-browser-vs-application-preferences/`, accessed 2019-02-18.

[14] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems," in *Scientific and Statistical Database Management, 2007. SSBDM '07. 19th International Conference on*, pp. 16–16, July 2007.

[15] Y.-Y. Chen, T. Suel, and A. Markowetz, "Efficient query processing in geographic web search engines," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, (New York, NY, USA), pp. 277–288, ACM, 2006.

[16] M. Mari, "Top global smartphone apps, whos in the top 10." `http://blog.globalwebindex.net/chart-of-the-day/top-global-smartphone-apps-who-s-in-the-top-10/`, accessed 2019-02-18.

[17] D. Zhang, Y. M. Chee, A. Mondal, A. K. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pp. 688–699, IEEE, 2009.

[18] C. B. Jones, A. I. Abdelmoty, D. Finch, G. Fu, and S. Vaid, *Geographic Information Science: Third International Conference, GIScience 2004, Adelphi, MD, USA,*

October 20-23, 2004. Proceedings, ch. The SPIRIT Spatial Search Engine: Architecture, Ontologies and Spatial Indexing, pp. 125–139. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[19] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," in *Proceedings of the VLDB Endowment*, vol. 6, pp. 217–228, 2013.

[20] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng, "On trip planning queries in spatial databases," in *Proceedings of the 9th International Conference on Advances in Spatial and Temporal Databases*, SSTD'05, (Berlin, Heidelberg), pp. 273–290, Springer-Verlag, 2005.

[21] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi, "The optimal sequenced route query," *The VLDB Journal*, vol. 17, pp. 765–787, July 2008.

[22] X. Cao, L. Chen, G. Cong, J. Guan, N. T. Phan, and X. Xiao, "Kors: Keyword-aware optimal route search system," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pp. 1340–1343, April 2013.

[23] S. Arora, "Approximation schemes for np-hard geometric optimization problems: a survey," *Mathematical Programming*, vol. 97, no. 1, pp. 43–69.

[24] S. Arora, "Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems," *J. ACM*, vol. 45, pp. 753–782, Sept. 1998.

[25] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh, "A road network embedding technique for k-nearest neighbor search in moving object databases," in *Proceedings of the 10th ACM International Symposium on Advances in Geographic Information Systems*, GIS '02, (New York, NY, USA), pp. 94–100, ACM, 2002.

[26] Y. Saab and M. VanPutte, "Shortest path planning on topographical maps," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 29, no. 1, pp. 139–150, 1999.

[27] Z. Sun and J. H. Reif, "On finding approximate optimal paths in weighted regions," *Journal of Algorithms*, vol. 58, no. 1, pp. 1 – 32, 2006.

[28] C. Li, Y. Gu, G. Yu, and F. Li, *wNeighbors: A Method for Finding k Nearest Neighbors in Weighted Regions*. 2011.

[29] "Elementary linear algebra: Howard anton, (wiley, new york, 1991) 526 pages," *Discrete Applied Mathematics*, vol. 36, no. 1, p. 95, 1992.

[30] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *Proceedings 2003 {VLDB} Conference* (J.-C. Freytag, P. Lockemann, S. Abiteboul, M. Carey, P. Selinger, and A. Heuer, eds.), pp. 802 – 813, San Francisco: Morgan Kaufmann, 2003.

[31] A. N. Papadopoulos and Y. Manolopoulos, "Multiple range query optimization in spatial databases," in *Advances in Databases and Information Systems* (W. Litwin, T. Morzy, and G. Vossen, eds.), (Berlin, Heidelberg), pp. 71–82, Springer Berlin Heidelberg, 1998.

[32] J. Shan, D. Zhang, and B. Salzberg, "On spatial-range closest-pair query," in *Advances in Spatial and Temporal Databases* (T. Hadzilacos, Y. Manolopoulos, J. Roddick, and Y. Theodoridis, eds.), (Berlin, Heidelberg), pp. 252–269, Springer Berlin Heidelberg, 2003.

[33] H. K. Ng and H. W. Leong, "Path-based range query processing using sorted path and rectangle intersection approach," in *Database Systems for Advanced Applications* (Y. Lee, J. Li, K.-Y. Whang, and D. Lee, eds.), (Berlin, Heidelberg), pp. 184–189, Springer Berlin Heidelberg, 2004.

[34] D. Li, J. Cao, X. Lu, and K. C. C. Chen, "Efficient range query processing in peer-to-peer systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, pp. 78–91, Jan 2009.

[35] M. A. Cheema, L. Brankovic, X. Lin, W. Zhang, and W. Wang, "Continuous monitoring of distance-based range queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, pp. 1182–1199, Aug 2011.

[36] S. Ilarri, E. Mena, and A. Illarramendi, "Location-dependent query processing: Where we are and where we are heading," *ACM Comput. Surv.*, vol. 42, pp. 12:1–12:73, Mar. 2010.

[37] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," *SIGMOD Rec.*, vol. 24, pp. 71–79, May 1995.

[38] T. Seidl and H.-P. Kriegel, "Optimal multi-step k-nearest neighbor search," *SIGMOD Rec.*, vol. 27, pp. 154–165, June 1998.

[39] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM Trans. Database Syst.*, vol. 24, pp. 265–318, June 1999.

[40] D. Taniar and W. Rahayu, "A taxonomy for nearest neighbour queries in spatial databases," *Journal of Computer and System Sciences*, vol. 79, no. 7, pp. 1017 – 1039, 2013.

[41] H. Cho, S. J. Kwon, and T. Chung, "A safe exit algorithm for continuous nearest neighbor monitoring in road networks," *Mobile Information Systems*, vol. 9, no. 1, pp. 37–53, 2013.

[42] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *SIGMOD Rec.*, vol. 14, pp. 47–57, June 1984.

[43] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: An efficient and robust access method for points and rectangles," *SIGMOD Rec.*, vol. 19, pp. 322–331, May 1990.

[44] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The r+-tree: A dynamic index for multi-dimensional objects," pp. 507–518, 1987.

[45] I. Kamel and C. Faloutsos, "Hilbert r-tree: An improved r-tree using fractals," in *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, (San Francisco, CA, USA), pp. 500–509, Morgan Kaufmann Publishers Inc., 1994.

[46] S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The x-tree: An index structure for high-dimensional data," in *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, (San Francisco, CA, USA), pp. 28–39, Morgan Kaufmann Publishers Inc., 1996.

[47] C. Yang and K.-I. Lin, "An index structure for efficient reverse nearest neighbor queries," in *Data Engineering, 2001. Proceedings. 17th International Conference on*, pp. 485–492, 2001.

[48] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, (New York, NY, USA), pp. 201–212, ACM, 2000.

[49] T. Nghiem, K. Maulana, A. Waluyo, D. Green, and D. Taniar, "Bichromatic reverse nearest neighbors in mobile peer-to-peer networks," in *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, pp. 160–165, March 2013.

[50] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *Data Engineering, 2004. Proceedings. 20th International Conference on*, pp. 301–312, March 2004.

[51] T. Nghiem, D. Green, and D. Taniar, "Peer-to-peer group k-nearest neighbours in mobile ad-hoc networks," in *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, pp. 166–173, Dec 2013.

[52] D. W. Choi and C. W. Chung, "Nearest neighborhood search in spatial databases," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pp. 699–710, April 2015.

[53] R. Benetis, S. Jensen, G. Karciauskas, and S. Saltenis, "Nearest and reverse nearest neighbor queries for moving objects," *The VLDB Journal*, vol. 15, pp. 229–249, Sept. 2006.

[54] A. Singh, H. Ferhatosmanoglu, and A. c. Tosun, "High dimensional reverse nearest neighbor queries," in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, (New York, NY, USA), pp. 91–98, ACM, 2003.

[55] E. Achtert, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle, "Reverse k-nearest neighbor search in dynamic and general metric databases," in *Proceedings of the 12th*

International Conference on Extending Database Technology: Advances in Database Technology, EDBT '09, (New York, NY, USA), pp. 886–897, ACM, 2009.

[56] L. Hu, W.-S. Ku, S. Bakiras, and C. Shahabi, "Verifying spatial queries using voronoi neighbors," in *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, (New York, NY, USA), pp. 350–359, ACM, 2010.

[57] D. Taniar, M. Safar, Q. T. Tran, W. Rahayu, and J. H. Park, "Spatial network rnn queries in gis," *The Computer Journal*, vol. 54, no. 4, pp. 617–627, 2011.

[58] K. Xuan, G. Zhao, D. Taniar, M. Safar, and B. Srinivasan, "Constrained range search query processing on road networks," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 5, pp. 491–504, 2011.

[59] M. Cheema, X. Lin, W. Zhang, and Y. Zhang, "Influence zone: Efficiently processing reverse k nearest neighbors queries," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pp. 577–588, April 2011.

[60] K. Xuan, G. Zhao, D. Taniar, W. Rahayu, M. Safar, and B. Srinivasan, "Voronoi-based range and continuous range query processing in mobile databases," *J. Comput. Syst. Sci.*, vol. 77, pp. 637–651, July 2011.

[61] K. Xuan, G. Zhao, D. Taniar, M. Safar, and B. Srinivasan, "Voronoi-based multi-level range search in mobile navigation," *Multimedia Tools Appl.*, vol. 53, pp. 459–479, June 2011.

[62] M. Safar, D. Ibrahimi, and D. Taniar, "Voronoi-based reverse nearest neighbor query processing on spatial networks," *Multimedia Systems*, vol. 15, no. 5, pp. 295–308, 2009.

[63] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, pp. 269–271, Dec. 1959.

[64] D. R. Lanning, G. K. Harrell, and J. Wang, "Dijkstra's algorithm and google maps," in *Proceedings of the 2014 ACM Southeast Regional Conference*, ACM SE '14, (New York, NY, USA), pp. 30:1–30:3, ACM, 2014.

[65] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: the case for a*," *International Journal of Geographical Information Science*, vol. 23, no. 4, pp. 531–543, 2009.

[66] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, July 1968.

[67] J. Sankaranarayanan, H. Alborzi, and H. Samet, "Efficient query processing on spatial networks," in *Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems*, GIS '05, (New York, NY, USA), pp. 200–209, ACM, 2005.

[68] R. Zhong, G. Li, K.-L. Tan, and L. Zhou, "G-tree: An efficient index for knn search on road networks," in *ACM CIKM*, pp. 39–48, 2013.

[69] R. Zhong, G. Li, K. L. Tan, L. Zhou, and Z. Gong, "G-tree: An efficient and scalable index for spatial search on road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2175–2189, 2015.

[70] K. C. K. Lee, W. Lee, B. Zheng, and Y. Tian, "Road: A new spatial object search framework for road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, pp. 547–560, March 2012.

[71] C. S. Jensen, J. Kolářvr, T. B. Pedersen, and I. Timko, "Nearest neighbor queries in road networks," in *Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems*, GIS '03, (New York, NY, USA), pp. 1–8, ACM, 2003.

[72] M. Kolahdouzan and C. Shahabi, "Voronoi-based k nearest neighbor search for spatial network databases," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pp. 840–851, VLDB Endowment, 2004.

[73] M. R. Kolahdouzan and C. Shahabi, "Continuous k-nearest neighbor queries in spatial network databases," in *STDBM*, 2004.

[74] H. Hu, D. L. Lee, and J. Xu, "Fast nearest neighbor search on road networks," in *Advances in Database Technology - EDBT 2006* (Y. Ioannidis, M. H. Scholl, J. W. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, eds.), (Berlin, Heidelberg), pp. 186–203, Springer Berlin Heidelberg, 2006.

[75] K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis, "Continuous nearest neighbor monitoring in road networks," in *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pp. 43–54, VLDB Endowment, 2006.

[76] H.-J. Cho and C.-W. Chung, "An efficient and scalable approach to cnn queries in a road network," in *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pp. 865–876, VLDB Endowment, 2005.

[77] K. Xuan, G. Zhao, D. Taniar, B. Srinivasan, M. Safar, and M. Gavrilova, "Network voronoi diagram based range search," in *Advanced Information Networking and Applications, 2009. AINA '09. International Conference on*, pp. 741–748, May 2009.

[78] Q. T. Tran, D. Taniar, and M. Safar, "Bichromatic reverse nearest-neighbor search in mobile systems," *Systems Journal, IEEE*, vol. 4, pp. 230–242, June 2010.

[79] S.-H. Shin, S.-C. Lee, S.-W. Kim, J. Lee, and E. G. Im, "Efficient shortest path finding of k-nearest neighbor objects in road network databases," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, (New York, NY, USA), pp. 1661–1665, ACM, 2010.

[80] M. Safar, "Enhanced continuous knn queries using pine on road networks," in *Digital Information Management, 2006 1st International Conference on*, pp. 248–256, Dec 2007.

[81] A. Cary, O. Wolfson, and N. Rishe, "Efficient and scalable method for processing top-k spatial boolean queries," in *Scientific and Statistical Database Management* (M. Gertz and B. Ludäscher, eds.), (Berlin, Heidelberg), pp. 87–95, Springer Berlin Heidelberg, 2010.

[82] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu, "Spatial keyword querying," in *Conceptual Modeling*, pp. 16–29, 2012.

[83] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems," in *Proceedings of the 19th International Conference on Scientific and Statistical Database Management*, SSDBM '07, (Washington, DC, USA), pp. 16–, IEEE Computer Society, 2007.

[84] Z. Li, K. C. K. Lee, B. Zheng, W. C. Lee, D. Lee, and X. Wang, "Ir-tree: An efficient index for geographic document search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, pp. 585–599, April 2011.

[85] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proc. VLDB Endow.*, vol. 2, pp. 337–348, Aug. 2009.

[86] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson, "Spatio-textual indexing for geographical search on the web," in *Advances in Spatial and Temporal Databases* (C. Bauzer Medeiros, M. J. Egenhofer, and E. Bertino, eds.), (Berlin, Heidelberg), pp. 218–235, Springer Berlin Heidelberg, 2005.

[87] A. Khodaei, C. Shahabi, and C. Li, "Hybrid indexing and seamless ranking of spatial and textual features of web documents," in *Database and Expert Systems Applications* (P. G. Bringas, A. Hameurlain, and G. Quirchmayr, eds.), (Berlin, Heidelberg), pp. 450–466, Springer Berlin Heidelberg, 2010.

[88] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma, "Hybrid index structures for location-based web search," in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, (New York, NY, USA), pp. 155–162, ACM, 2005.

[89] X. Cao, G. Cong, and C. S. Jensen, "Retrieving top-k prestige-based relevant spatial web objects," *Proc. VLDB Endow.*, vol. 3, pp. 373–384, Sept. 2010.

[90] J. Xu and H. Lu, "Efficiently answer top-k queries on typed intervals," *Information Systems*, vol. 71, no. Supplement C, pp. 164 – 181, 2017.

[91] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen, "Joint top-k spatial keyword query processing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 10, pp. 1889–1903, 2012.

[92] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pp. 541–552, IEEE, 2011.

[93] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pp. 373–384, ACM, 2011.

[94] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu, "Collective spatial keyword queries: a distance owner-driven approach," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 689–700, ACM, 2013.

[95] L. Guo, J. Shao, H. Aung, and K.-L. Tan, "Efficient continuous top-k spatial keyword queries on road networks," *GeoInformatica*, vol. 19, no. 1, pp. 29–60, 2015.

[96] J. Lu, Y. Lu, and G. Cong, "Reverse spatial and textual k nearest neighbor search," in *ACM SIGMOD*, pp. 349–360, 2011.

[97] Y. Gao, X. Qin, B. Zheng, and G. Chen, "Efficient reverse top-k boolean spatial keyword queries on road networks," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 27, pp. 1205–1218, May 2015.

[98] H. Hu, G. Li, Z. Bao, J. Feng, Y. Wu, Z. Gong, and Y. Xu, "Top-k spatio-textual similarity join," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 2, pp. 551–565, 2016.

[99] K. Zheng, H. Su, B. Zheng, S. Shang, J. Xu, J. Liu, and X. Zhou, "Interactive top-k spatial keyword queries," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pp. 423–434, April 2015.

[100] P. Zhang, H. Lin, B. Yao, and D. Lu, "Level-aware collective spatial keyword queries," *Information Sciences*, vol. 378, no. Supplement C, pp. 194 – 214, 2017.

[101] C. Zhang, Y. Zhang, W. Zhang, X. Lin, M. A. Cheema, and X. Wang, "Diversified spatial keyword search on road networks," in *EDBT*, pp. 367–378, 2014.

[102] X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu, "Retrieving regions of interest for user exploration," *Proc. VLDB Endow.*, vol. 7, pp. 733–744, May 2014.

[103] Y. Xu, J. Guan, F. Li, and S. Zhou, "Scalable continual top-k keyword search in relational databases," *Data & Knowledge Engineering*, vol. 86, pp. 206–223, 2013.

[104] C. Luo, L. Junlin, G. Li, W. Wei, Y. Li, and J. Li, "Efficient reverse spatial and textual k nearest neighbor queries on road networks," *Knowledge-Based Systems*, vol. 93, no. Supplement C, pp. 121 – 134, 2016.

[105] D.-W. Choi and C.-W. Chung, "A k-partitioning algorithm for clustering large-scale spatio-textual data," *Information Systems*, vol. 64, no. Supplement C, pp. 1 – 11, 2017.

[106] S. Alsubaiee and C. Li, "Fuzzy keyword search on spatial data," in *Database Systems for Advanced Applications*, pp. 464–467, Springer, 2010.

[107] D. Zhang, B. C. Ooi, and A. K. H. Tung, "Locating mapped resources in web 2.0," in *IEEE ICDE*, pp. 521–532, 2010.

[108] S. Luo, Y. Luo, S. Zhou, G. Cong, J. Guan, and Z. Yong, "Distributed spatial keyword querying on road networks.," in *EDBT*, pp. 235–246, 2014.

[109] T. Hashem, T. Hashem, M. E. Ali, and L. Kulik, "Group trip planning queries in spatial databases," in *Proceedings of the 13th International Conference on Advances in Spatial and Temporal Databases*, SSTD'13, (Berlin, Heidelberg), pp. 259–276, Springer-Verlag, 2013.

[110] D. Taniar, M. A. Cheema, and Z. Shao, "Trip Planning Queries in Indoor Venues," *The Computer Journal*, vol. 61, pp. 409–426, 11 2017.

[111] H. Chen, W.-S. Ku, M.-T. Sun, and R. Zimmermann, "The multi-rule partial sequenced route query," in *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '08, (New York, NY, USA), pp. 10:1–10:10, ACM, 2008.

[112] C. Salgado, "Keyword-aware skyline routes search in indoor venues," in *Proceedings of the 9th ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness*, ISA'18, (New York, NY, USA), pp. 25–31, ACM, 2018.

[113] J. Dai, C. Liu, J. Xu, and Z. Ding, "On personalized and sequenced route planning," *World Wide Web*, vol. 19, pp. 679–705, Jul 2016.

[114] S. C. Soma, T. Hashem, M. A. Cheema, and S. Samrose, "Trip planning queries with location privacy in spatial databases," *World Wide Web*, vol. 20, no. 2, pp. 205–236, 2017.

[115] S. Du, H. Zhang, H. Xu, J. Yang, and O. Tu, "To make the travel healthier: a new tourism personalized route recommendation algorithm," *Journal of Ambient Intelligence and Humanized Computing*, Oct 2018.

[116] J. S. B. Mitchell and C. H. Papadimitriou, "The weighted region problem: Finding shortest paths through a weighted planar subdivision," *J. ACM*, vol. 38, pp. 18–73, Jan. 1991.

[117] S.-W. Cheng, J. Jin, A. Vigneron, and Y. Wang, "Approximate shortest homotopic paths in weighted regions," in *Algorithms and Computation* (O. Cheong, K.-Y. Chwa, and K. Park, eds.), (Berlin, Heidelberg), pp. 109–120, Springer Berlin Heidelberg, 2010.

[118] L. Aleksandrov, A. Maheshwari, and J.-R. Sack, "Determining approximate shortest paths on weighted polyhedral surfaces," *J. ACM*, vol. 52, pp. 25–53, Jan. 2005.

[119] M. Lanthier, A. Maheshwari, and J.-R. Sack, "Shortest anisotropic paths on terrains," in *Automata, Languages and Programming* (J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds.), (Berlin, Heidelberg), pp. 524–533, Springer Berlin Heidelberg, 1999.

[120] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J. R. Sack, "An ϵ — approximation algorithm for weighted shortest paths on polyhedral surfaces," in *Algorithm Theory — SWAT'98* (S. Arnborg and L. Ivansson, eds.), (Berlin, Heidelberg), pp. 11–22, Springer Berlin Heidelberg, 1998.

[121] C. Li, Y. Gu, G. Yu, and F. Li, "wneighbors: A method for finding k nearest neighbors in weighted regions," in *Database Systems for Advanced Applications* (J. X. Yu, M. H. Kim, and R. Unland, eds.), (Berlin, Heidelberg), pp. 134–148, Springer Berlin Heidelberg, 2011.

[122] G. Karypis and V. Kumar, "Analysis of multilevel graph partitioning," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, 1995.

[123] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: An efficient and robust access method for points and rectangles," in *ACM SIGMOD*, pp. 322–331, 1990.

[124] C. Demetrescu, "9th dimacs implementation challenge - shortest paths." `http://www.dis.uniroma1.it/challenge9/download.shtml`, accessed 2019-02-18.

[125] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng, "Real datasets for spatial databases: Road networks and points of interest." `http://www.cs.utah.edu/~lifeifei/SpatialDataset.htm`, accessed 2019-02-18.

[126] P. Stone, "General inquirer." `http://www.wjh.harvard.edu/~inquirer/No.html`, accessed 2019-02-18.

[127] J. Breen, "twitter-sentiment-analysis-tutorial-201107." `https://github.com/jeffreybreen/twitter-sentiment-analysis-tutorial-201107/blob/master/data/opinion-lexicon-English/negative-words.txt`, accessed 2019-02-18.

[128] H. Schrom-Feiertag, P. Luley, and L. Paletta, *A Mobile LBS for Geo-Content Generation Facilitating Users to Share, Rate and Access Information in a Novel Manner*, pp. 55–75. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[129] N. Allheeib, M. S. Islam, D. Taniar, Z. Shao, and M. A. Cheema, "Density-based reverse nearest neighbourhood search in spatial databases," *Journal of Ambient Intelligence and Humanized Computing*, Oct 2018.

[130] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *2009 IEEE 25th International Conference on Data Engineering*, pp. 688–699, March 2009.

[131] I. K. Adusei, K. Kyamakya, and F. Erbas, "Location-based services: advances and challenges," in *Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat. No.04CH37513)*, vol. 1, pp. 1–7 Vol.1, May 2004.

[132] K. Virrantaus, J. Markkula, A. Garmash, V. Terziyan, J. Veijalainen, A. Katanosov, and H. Tirri, "Developing gis-supported location-based services," in *Proceedings of the Second International Conference on Web Information Systems Engineering*, vol. 2, pp. 66–75 vol.2, Dec 2001.

[133] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, MobiSys '03, (New York, NY, USA), pp. 31–42, ACM, 2003.

[134] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, (New York, NY, USA), pp. 443–454, ACM, 2003.

[135] J. Xu, L. Guo, Z. Ding, X. Sun, and C. Liu, "Traffic aware route planning in dynamic road networks," in *Database Systems for Advanced Applications* (S.-g. Lee, Z. Peng, X. Zhou, Y.-S. Moon, R. Unland, and J. Yoo, eds.), (Berlin, Heidelberg), pp. 576–591, Springer Berlin Heidelberg, 2012.

[136] L. Wu, W. Ma, Y. Yang, and K. Wang, "A competitive analysis approach for route choice with uncertain travel times and blocked nodes," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, pp. 345–355, Jan 2019.

[137] A. A. Haryanto, M. S. Islam, D. Taniar, and M. A. Cheema, "Ig-tree: an efficient spatial keyword index for planning best path queries on road networks," *World Wide Web*, Nov 2018.

[138] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Santa Clara, CA, USA: Springer-Verlag TELOS, 3rd ed. ed., 2008.

[139] N. A. Papadakis and A. N. Perakis, "Deterministic minimal time vessel routing," *Operations Research*, vol. 38, no. 3, pp. 426–438, 1990.

[140] S. T. Leutenegger, "Multi dimensional data sets." `https://www.cs.du.edu/~leut/MultiDimData.html`, accessed 2019-02-18.

[141] S. T. Leutenegger, M. A. Lopez, and J. Edgington, "Str: a simple and efficient algorithm for r-tree packing," in *Proceedings 13th International Conference on Data Engineering*, pp. 497–506, April 1997.

[142] D. J. Mavriplis, "An advancing front delaunay triangulation algorithm designed for robustness," *Journal of Computational Physics*, vol. 117, no. 1, pp. 90 – 101, 1995.

Appendix A

Best Path on Road Networks

This appendix section provides some sample datasets that we use in our experiments and also the raw evaluation results of our experiments on Best Path in Road Networks setting.

A.1 Sample Datasets

In this section, we provide some sample datasets that we use in our experiments. Each dataset consists of vertices data, edges data, and keywords data. We also provide a snippet of the *IG-Tree* data structure after all these datasets have been indexed.

A.1.1 Sample data for vertices

Below we present sample vertices data. The first line indicates the number of vertices in the dataset, the second line and forward consists of v to indicate that it is a vertex, an ID of the vertex, the longitude and latitude of the corresponding vertex.

```
264346
v 1 -73530767 41085396
v 2 -73530538 41086098
v 3 -73519366 41048796
v 4 -73519377 41048654
v 5 -73524567 41093796
v 6 -73525490 41093834
v 7 -73531927 41110484
v 8 -73530106 41110611
```

```
v 9 -73529341 41125895
v 10 -73529746 41127369
v 11 -73530583 41125051
v 12 -73529763 41085358
v 13 -73529834 41086062
v 14 -73613384 41065086
v 15 -73615019 41065131
v 16 -73693133 41058075
v 17 -73694273 41059296
v 18 -73512230 41287431
v 19 -73511896 41286556
v 20 -73501634 41286067
v 21 -73501424 41284975
v 22 -73500122 41286141
v 23 -73554632 41132182
v 24 -73554531 41129747
v 25 -73663848 41025429
          ⋮
v 264346 -73917690 41291980
```

A.1.2 Sample data for edges

Below we present sample edges data. The first line indicates the number of vertices and edges in the dataset. The second line and forward consists of a to indicate that it is an edge, the vertex where the edge starts, the vertex where the edge ends, and the edge's weight.

```
264346 733846
a 1 2 803
a 2 1 803
a 3 4 158
a 4 3 158
a 5 6 774
a 6 5 774
```

```
a 7 8 1531
a 8 7 1531
a 9 10 1673
a 10 9 1673
a 9 11 1400
a 11 9 1400
a 1 12 842
a 12 1 842
a 2 13 591
a 13 2 591
a 14 15 1371
a 15 14 1371
a 16 17 1659
a 17 16 1659
a 18 19 1012
a 19 18 1012
a 20 21 1226
a 21 20 1226
a 20 22 1265
          ⋮
a 259707 261228 389
```

A.1.3 Sample data for keywords

The keywords dataset presented below consists of a keyword and its coordinate location in the graph.

```
airport -124.18944 41.93778
airport -124.21056 40.78028
airport -124.21222 40.78083
airport -124.23472 41.77917
airport -124.23639 41.78
          ⋮
arch -115.74778 35.23944
```

```
arch -116.09972 34.88194
arch -116.76444 36.285
arch -116.84194 34.95111
arch -117.2725 32.85111
          ⋮
area -114.5675 32.76083
area -115.07722 34.14111
area -115.25194 35.20139
area -115.37722 34.34833
area -115.51333 33.07556
          ⋮
woods -124.08444 41.78889
woods -124.08667 41.78611
woods -124.10861 41.76806
woods -124.12889 41.76639
woods -124.13056 41.80639
```

A.1.4 A snippet of *IG-Tree* data structure

```
264346 733846 733847
264346 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150
151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181
182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212
213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243
244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274
275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336
337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367
368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398
399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429
430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460
461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491
492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553
554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584
585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615
616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646
647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677
678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708
709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739
740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770
771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801
802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832
833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863
864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894
895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925
926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956
957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987
988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014
1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038
1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063
1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088
1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113
1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138
1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163
1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188
```

Figure A.1: A snippet of the indexed vertices and edges data to *IG-Tree*

A.2 Evaluation Data

This section presents the evaluation results of our experiments in Chapter 3.

A.2.1 IG-Tree

Table A.1: Building Time (ms)

Dataset	IG-Tree	G-Tree
CAL	2790.23	1992.50
NY	554465.00	498784.15
COL	1127079.26	1035827.45
FLA	3658657.04	3612154.00

Table A.2: Index Size (MB)

Dataset	IG-Tree	G-Tree
CAL	18.94	16.80
NY	321.40	297.40
COL	489.80	454.10
FLA	1335.70	1200.00

Table A.3: Index Reconstruction Time (ms)

No. of k^-	Runtime
1	0.92
2	192.732
3	298.844
4	267.300
5	374.010
6	379.524
7	389.116
8	377.966
9	364.102
10	775.528
11	1082.498
12	1169.898
13	1142.446
14	1238.278
15	1378.931

A.2.2 Query Performance with all positive query keywords

Table A.4: Query performance (in μs) with all positive query keywords for CAL

K	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
1	118	30	20	360
5	420	270	164	2025160
10	822	572	456	∞
15	1148	854	562	∞

Table A.5: Query performance (in μs) with all positive query keywords for NY

K	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
1	280	300	170	480
5	960	1110	1200	21507410
10	1750	1830	1780	∞
15	2520	2580	3100	∞

Table A.6: Query performance (in μs) with all positive query keywords for COL

K	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
1	472	530	186	1060
5	1272	1296	776	11917850
10	2528	2886	2022.5	∞
15	4170	4430	2650	∞

Table A.7: Query performance (in μs) with all positive query keywords for FLA

K	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
1	484	420	160	1410
5	1644	1136	677.5	22731410
10	1940	1630	1322.5	∞
15	2966	2780	1540	∞

A.2.3 Approximation accuracy for all positive query keywords

Table A.8: Approximation accuracy for CAL

K	**OptDist**	**Ancestor**	**Euclidean**
1	100.00%	100.00%	100.00%
5	90.97%	89.08%	89.08%
10	17.84%	11.35%	11.35%
15	22.44%	15.95%	15.95%

Table A.9: Approximation accuracy for NY

K	**OptDist**	**Ancestor**	**Euclidean**
1	100.00%	100.00%	98.27%
5	83.65%	72.69%	13.26%
10	92.15%	72.33%	4.83%
15	68.33%	43.57%	0.00%

Table A.10: Approximation accuracy for COL

K	**OptDist**	**Ancestor**	**Euclidean**
1	100.00%	100.00%	92.89%
5	89.96%	89.96%	89.23%
10	79.00%	51.20%	26.62%
15	53.72%	46.08%	16.02%

Table A.11: Approximation accuracy for FLA

K	**OptDist**	**Ancestor**	**Euclidean**
1	100.00%	100.00%	44.58%
5	74.74%	51.30%	9.07%
10	28.93%	2.32%	0.00%
15	28.93%	3.95%	0.00%

A.2.4 Query performance with combination of positive and negative query keywords

Table A.12: Query performance (in μs) for CAL

K	OptDist	Ancestor	Euclidean	Baseline
1	126	130	28	420
5	392	440	174	2070810
10	870	800	328	∞
15	1278	1170	376	∞

Table A.13: Query performance (in μs) for NY

K	OptDist	Ancestor	Euclidean	Baseline
1	370	370	200	560
5	1430	1530	1330	15867510
10	2570	2520	1780	∞
15	3830	3590	3070	∞

Table A.14: Query performance (in μs) for COL

K	OptDist	Ancestor	Euclidean	Baseline
1	508	486	190	1210
5	4311358	4330274	1425448	317350200
10	8750282	8473730	1420520	∞
15	13212202	12908344	2777676	∞

Table A.15: Query performance (in μs) for FLA

K	OptDist	Ancestor	Euclidean	Baseline
1	4771740	4627490	240	4455530
5	11746080	12028610	890	∞
10	33483730	31591060	4431390	∞
15	47044680	51762480	8832640	∞

A.2.5 Approximation accuracy for datasets with negative keywords

Table A.16: Approximation accuracy for CAL

K	**OptDist**	**Ancestor**	**Euclidean**
1	100.00%	100.00%	100.00%
5	89.08%	89.08%	89.08%
10	6.49%	6.49%	4.86%
15	6.49%	6.49%	7.30%

Table A.17: Approximation accuracy for NY

K	**OptDist**	**Ancestor**	**Euclidean**
1	100.00%	100.00%	98.27%
5	83.65%	83.65%	13.26%
10	86.14%	86.14%	21.82%
15	68.33%	68.33%	0.00%

Table A.18: Approximation accuracy for COL

K	**OptDist**	**Ancestor**	**Euclidean**
1	100.00%	100.00%	92.89%
5	33.95%	33.95%	30.61%
10	30.31%	30.31%	10.12%
15	55.15%	55.15%	14.19%

Table A.19: Approximation accuracy for FLA

K	**OptDist**	**Ancestor**	**Euclidean**
1	100.00%	100.00%	44.58%
5	77.18%	77.18%	12.79%
10	28.93%	28.93%	22.08%
15	28.17%	28.17%	0.00%

A.2.6 Query performance based on keyword density

Table A.20: Running time for CAL (μs)

Density	OptDist	Ancestor	Euclidean	Baseline
0.01	7730	310	260	111480
0.05	36160	210	560	754650
0.10	80150	460	1170	1632060
0.15	160820	600	1560	1988160
0.20	167140	680	2110	3601800
0.25	206610	590	3780	4790410
0.30	241010	830	2800	5824460

Table A.21: Running time for NY (μs)

Density	OptDist	Ancestor	Euclidean	Baseline
0.01	314880	316110	2420	5205130
0.05	1514410	1437980	7680	28806980
0.10	3132610	2866290	15120	72088560
0.15	4383280	4356070	22100	117026160
0.20	5958840	5881730	28440	181371380
0.25	7492410	7210280	36040	246477170
0.30	9170350	8789430	43050	325406060

Table A.22: Running time for COL (μs)

Density	OptDist	Ancestor	Euclidean	Baseline
0.01	740457	736650	58900	5233520
0.05	3942953	3769780	23670	5777270
0.10	7610313	7646680	32410	11319830
0.15	11546237	10990880	38770	78222030
0.20	14963597	15037360	49960	142473920
0.25	19289803	18938100	69610	29134130
0.30	23850140	23798180	66170	167117250

Table A.23: Running time for FLA (μs)

Density	OptDist	Ancestor	Euclidean	Baseline
0.01	4781260	25870	11160	192074020
0.05	23185060	20830	39940	1149258000
0.10	50665380	41420	75510	∞
0.15	76096870	62720	133880	∞
0.20	94145250	89870	144610	∞
0.25	117999220	98620	175000	∞
0.30	145745630	128560	207170	∞

A.2.7 Query performance when K is varied (keyword density=0.05)

Table A.24: Query performance (in μs) when K is varied for CAL

K	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
1	9850	8280	180	44230
5	38100	36930	370	6421390
10	117810	115470	870	∞
15	169650	165590	2350	∞

Table A.25: Query performance (in μs) when K is varied for NY

K	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
1	1514410	1437980	7680	28806980
5	7476930	7388430	42720	325406060
10	14470460	14024720	92570	∞
15	21355820	21221190	142780	∞

Table A.26: Query performance (in μs) when K is varied for COL

K	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
1	3212160	2889780	2920	22378240
5	24689770	23079440	37275	∞
10	38218740	35421150	117195	∞
15	47106560	44772080	168980	∞

Table A.27: Query performance (in μs) when K is varied for FLA

K	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
1	40302960	41841250	152840	64213900
5	163691360	157232920	273540	∞
10	261409640	245686190	752300	∞
15	448582940	421862600	957670	∞

A.2.8 Query performance based on keyword density with all negative keywords

Table A.28: Running time for CAL (μs)

Density	OptDist	Ancestor	Euclidean	Baseline
0.01	10.00	13.50	37.33	24.00
0.015	10.00	12.50	23.33	12.00
0.02	10.00	10.00	20.00	18.00
0.025	10.00	10.00	30.00	13.00
0.03	10.00	10.00	23.33	12.00
0.035	10.00	12.50	23.33	32.00
0.04	10.00	10.00	23.33	18.00
0.045	10.00	10.00	20.00	14.00
0.05	10.00	15.00	16.67	22.00
0.055	10.00	10.00	26.67	16.00

Table A.29: Running time for NY (μs)

Density	OptDist	Ancestor	Euclidean	Baseline
0.01	70.00	60.00	7030.00	8700.00
0.015	80.00	70.00	10480.00	10890.00
0.02	80.00	80.00	16270.00	16940.00
0.025	80.00	80.00	22400.00	22790.00
0.03	100.00	90.00	29360.00	31270.00
0.035	110.00	90.00	32010.00	39810.00
0.04	120.00	100.00	50760.00	62910.00
0.045	120.00	110.00	70020.00	101380.00
0.05	110.00	100.00	70730.00	112950.00
0.055	90.00	110.00	106520.00	157560.00

Table A.30: Running time for COL (μs)

Density	OptDist	Ancestor	Euclidean	Baseline
0.01	210	210	4018410	3368290
0.015	130	130	29100	61390
0.02	100	100	1944510	1694840
0.025	100	100	5715200	3318000
0.03	120	110	198620	116290
0.035	120	120	2181470	1797700
0.04	100	110	3914810	3410070
0.045	240	100	5646470	5114500
0.05	230	140	4637690	2746790
0.055	220	100	5856380	5425160

Table A.31: Running time for FLA (μs)

Density	OptDist	Ancestor	Euclidean	Baseline
0.01	150	180	134700	36676
0.015	110	120	291410	34790
0.02	90	140	505340	71830
0.025	80	110	683270	160700
0.03	80	110	2379900	130240
0.035	110	180	2530570	220020
0.04	80	120	1458890	2486840
0.045	140	170	3818020	3933990
0.05	190	180	6122890	1851210
0.055	180	150	2594470	4640500

A.2.9 Running time based on keyword ratio (positive:negative)

Table A.32: Running time for CAL (μs)

Ratio ($k^+ : k^-$)	OptDist	Ancestor	Euclidean	Baseline
1:0	19350	15960	240	63140
0:1	10	10	10	10
1:1	19590	18670	130	59060
5:0	81540	76240	510	628240
0:5	10	20	10	10
5:1	124580	121570	910	965350

Table A.33: Running time for NY (μs)

Ratio ($k^+ : k^-$)	OptDist	Ancestor	Euclidean	Baseline
1:0	432675	338290	3430	5301010
0:1	40	40	40	56250
1:1	755130	431200	14750	8175750
5:0	1578150	1648927	12740	29581170
0:5	60	90	60	131250
5:1	3484590	1428350	1140	104590

Table A.34: Running time for COL (μs)

Ratio ($k^+ : k^-$)	OptDist	Ancestor	Euclidean	Baseline
1:0	754100	646270	122040	8297350
0:1	120	140	10	68130
1:1	7275250	6606650	4160	17669590
5:0	3275750	3016690	28700	53244890
0:5	180	110	10	2869860
5:1	51253040	51583770	23950	122071220

Table A.35: Running time for FLA (μs)

Ratio ($k^+ : k^-$)	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
1:0	4641073	4695650	157815	264780230
0:1	85	120	10	124475
1:1	58663630	62213150	229835	340855640
5:0	24277520	14570340	38985	1329239960
0:5	200	170	10	2155695
5:1	288156000	291612900	32670	170465957

A.2.10 Effect of varying distance between s_l and d_l

Table A.36: Running time for CAL (μs)

s_l-d_l **Distance**	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
2%	1982	1644	495	7703
4%	1456	1334	385	63029220012
8%	1452	1388	378	∞
16%	2174	2016	410	∞
32%	2284	2342	1015	∞
64%	2650	2886	488	∞

Table A.37: Running time for NY (μs)

s_l-d_l **Distance**	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
2%	2713	257	1860	7214300
4%	2695	2815	1900	∞
8%	3200	3413	1810	∞
16%	3919	3721	2040	∞
32%	4345	3928	2350	∞
64%	4391	4588	2100	∞

Table A.38: Running time for COL (μs)

s_l-d_l **Distance**	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
2%	10286004	10100062	1415092	763418401
4%	10597290	10121202	1375676	∞
8%	10721350	10229815	3026498	∞
16%	10812266	10411977.14	1513884	∞
32%	12872446	12450766	1464288	∞
64%	12908952	12889676	1467690	∞

Table A.39: Running time for FLA (μs)

s_l-d_l **Distance**	**OptDist**	**Ancestor**	**Euclidean**	**Baseline**
2%	51416950	51420868	8464684	7134293200
4%	51226042	49723128	16750744	∞
8%	54082062	51796420	20881688	∞
16%	64164566	64205782	16848088	∞
32%	63393884	66546390	16776136	∞
64%	61654196	67316516	13249918	∞

Appendix B

Best Path on Weighted Regions

This appendix section provides some sample datasets that we use in our experiments and also the raw evaluation results of our experiments on Best Path in Weighted Regions setting.

B.1 Sample Datasets

In this section, we provide some sample datasets that we use in our experiments. Each weighted regions dataset consists of vertices data, edges data, faces data, and keywords data. We also provide a snippet of the *wIG-Tree* data structure after all these datasets have been indexed.

B.1.1 Sample data for vertices

Below we present sample vertices data. The first line indicates the number of vertices in the dataset, the second line and forward consists of v to indicate that it is a vertex, an ID of the vertex, the longitude and latitude of the corresponding vertex.

```
5088
v 1 -0.0000000E+02 -0.0000000E+02
v 2 -0.7196484E+01 -0.10000000E+0
v 3 -0.4912166E+01 0.10000000E+02
v 4 -0.1590125E+01 0.10000000E+02
v 5 0.1216096E+01 .10000000E+02
v 6 0.4028150E+01 .10000000E+02
```

```
v 7 0.6832086E+01 .10000000E+02
v 8 0.10000000E+02 0.10000000E+02
v 9 0.10000000E+02 0.72046522E+01
v 10 0.000000E+02 44394928E+01
v 11 0.10000000E+02 0.16758174E+01
v 12 0.10000000E+02 0.10975847E+01
v 13 0.10000000E+02 0.38789688E+01
v 14 0.10000000E+02 0.66524759E+01
v 15 0.10000000E+02 0.10000000E+02
v 16 0.72485234E+01 0.10000000E+02
v 17 0.44925888E+01 0.10000000E+02
v 18 0.17219966E+01 0.10000000E+02
v 19 -0.10595662E+01 0.10000000E+02
v 20 -0.38374964E+01 0.10000000E+02
v 21 -0.66067043E+01 0.10000000E+02
v 22 -0.10000000E+02 0.10000000E+02
v 23 -0.10000000E+02 0.72126789E+01
v 24 -0.10000000E+02 0.44215686E+01
v 25 -0.10000000E+02 0.16171541E+01
                ⋮
v 5088 -0.73502172E+01 -0.63912125E+01
```

B.1.2 Sample data for edges

Below we present sample edges data. The first line indicates the number of vertices and edges in the dataset. The second line and forward consists of a to indicate that it is an edge, the vertex where the edge starts, the vertex where the edge ends, and the edge's weight.

```
5088 58554
a 5045 5066 1.33275693481564
a 844 295 0.00440306118276318
a 551 466 0.00140518531945969
a 9 8 2.7953478
```

```
a 4952 4972 0.702577814698251
a 4988 5027 1.19710684530158
a 728 795 0.00486440527707684
a 439 517 0.00113300306857157
a 298 297 0.00439908271034647
a 5079 14 2.37968692327731
a 4951 4957 0.841799039091048
a 5030 5017 1.23130491867307
a 696 90 0.00501435348120274
a 24 23 2.7911103
a 22 21 3.3932957
a 454 457 0.00126814174177649
a 2 1 7.19717875019483
a 457 513 0.00106442981296512
a 876 877 0.00514234148352673
a 5068 3 17.3618169663472
a 5060 5062 2.00429877030636
a 429 474 0.000766538897405759
a 918 258 0.00492902253747427
a 282 281 0.00447362678231983
a 988 358 0.00707860329115861
          ⋮
a 2 5068 7.60800162786464
```

B.1.3 Sample data for triangle faces

Below we present sample faces data. The first line indicates the number of faces in the dataset. The second line and forward consists of t to indicate that it is a face, vertices $\{v_i, v_j, v_k\}$ that make up the triangle face, and the face's weight.

```
9759
t 5045 5066 5007 20
t 844 295 846 17
t 551 466 516 10
```

```
t 9 8 7 15
t 4952 4972 4956 5
t 4988 5027 5016 5
t 728 795 794 9
t 439 517 451 7
t 298 297 873 7
t 5079 14 5065 18
t 4951 4957 4996 20
t 5030 5017 5013 5
t 696 90 1007 5
t 24 23 5082 13
t 22 21 5081 1
t 454 457 440 3
t 2 1 8 5
t 457 513 481 16
t 876 877 889 19
t 5068 3 2 8
t 5060 5062 5061 16
t 429 474 476 4
t 918 258 1010 7
t 282 281 917 15
t 988 358 1132 17
          ⋮
a 2 5068 7.60800162786464
```

B.1.4 Sample data for keywords

Below we present some sample keywords that we use in our experiments of wBestPath. In our program, we use a random generator to place these keywords randomly in the weighted regions.

```
airport
arch
area
```

arroyo

bar

basin

bay

beach

bench

bend

bridge

building

canal

cape

cemetery

channel

church

civil

cliff

crater

crossing

dam

falls

flat

forest

⋮

woods

B.1.5 Sample Data Generator

Figure B.1: Sample program to generate Delaunay Triangulation datasets

B.1.6 A snippet of *wIG-Tree* data structure

```
5088 58554 58555
5088 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150
151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181
182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212
213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243
244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274
275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336
337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367
368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398
399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429
430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460
461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491
492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553
554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584
585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615
616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646
647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677
678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708
709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739
740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770
771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801
802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832
833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863
864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894
895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925
926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956
957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987
988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014
1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039
1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064
1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089
1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114
1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139
1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164
1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189
```

Figure B.2: A snippet of the indexed vertices, edges, and faces data to *wIG-Tree*

B.2 Evaluation Data of Best Path on Weighted Regions

This section presents the evaluation results of our experiments in Chapter 4.

B.2.1 *wIG-Tree*

Table B.1: Building Time (ms)

Dataset	wIG-Tree	IG-Tree	G-Tree
CFD-1	14282.933	1660.15	763.61
CFD-2	680163.79	24598.38	10405.35
CFD-3	11725780.75	122522.72	84217.70

Table B.2: Index Size (MB)

Dataset	wIG-Tree	IG-Tree	G-Tree
CFD-1	29.279	7.8	6.6
CFD-2	313.5	135.5	120.9
CFD-3	1251.6	460.2	398.5

B.2.2 Query Performance with all positive query keywords

Table B.3: Query performance (in μs) with all positive query keywords for CFD-1

K	MDA	MPA	Baseline
1	250	20	220
5	570	360	6621240
10	1330	1000	∞
15	1820	1130	∞

Table B.4: Query performance (in μs) with all positive query keywords for CFD-2

K	MDA	MPA	Baseline
1	333	342	480
5	1150	1298	21507410
10	2408	2488	∞
15	3392	4218	∞

Table B.5: Query performance (in μs) with all positive query keywords for CFD-3

K	MDA	MPA	Baseline
1	635	552	1410
5	1178	1066	22731410
10	1924	1432	∞
15	2667	2388	∞

B.2.3 Approximation accuracy for all positive query keywords

Table B.6: Approximation accuracy for CFD-1

K	**MDA**	**MPA**
1	100.00%	100.00%
5	95.04%	59.29%
10	81.37%	37.01%
15	81.37%	10.29%

Table B.7: Approximation accuracy for CFD-2

K	**MDA**	**MPA**
1	100.00%	100.00%
5	93.65%	73.69%
10	92.15%	47.79%
15	68.33%	23.57%

Table B.8: Approximation accuracy for CFD-3

K	**MDA**	**MPA**
1	100.00%	100.00%
5	74.74%	51.30%
10	31.00%	2.32%
15	28.93%	3.95%

B.2.4 Query performance with combination of positive and negative query keywords

Table B.9: Query performance (in μs) for CFD-1

K	**MDA**	**MPA**	**Baseline**
1	194	205	517
5	276798	275825	1219522590
10	510492	505584	∞
15	802734	841557	∞

Table B.10: Query performance (in μs) for CFD-2

K	**MDA**	**MPA**	**Baseline**
1	403	402	560
5	1136	1206	15867510
10	2372	2404	∞
15	2966	3724	∞

Table B.11: Query performance (in μs) for CFD-3

K	**MDA**	**MPA**	**Baseline**
1	4577428	4680524	4455530
5	11396902	12076554	∞
10	33139018	34992438	∞
15	49971682	50326990	∞

B.2.5 Approximation accuracy for datasets with negative keywords

Table B.12: Approximation accuracy for CFD-1

K	**MDA**	**MPA**
1	100.00%	100.00%
5	95.04%	95.04%
10	81.37%	81.37%
15	81.37%	81.37%

Table B.13: Approximation accuracy for CFD-2

K	**MDA**	**MPA**
1	100.00%	100.00%
5	83.65%	83.65%
10	79.14%	79.14%
15	68.33%	68.33%

Table B.14: Approximation accuracy for CFD-3

K	MDA	MPA
1	100.00%	100.00%
5	77.18%	77.18%
10	28.93%	28.93%
15	29.17%	29.17%

B.2.6 Query performance based on keyword density

Table B.15: Running time for CFD-1 (μs)

Density	MDA	MPA	Baseline
0.01	6830	7890	23800
0.05	32060	29270	103420
0.10	78870	58900	309430
0.15	118520	96020	428710
0.20	136880	126200	604100
0.25	165360	159540	715300
0.30	187800	185900	927930

Table B.16: Running time for CFD-2 (μs)

Density	MDA	MPA	Baseline
0.01	4648	45790	496150
0.05	23325	248790	2423280
0.10	48784	499890	6553570
0.15	71647	723670	12422250
0.20	95635	982680	19761120
0.25	127055	1387240	28752420
0.30	149762	1551030	39041870

Table B.17: Running time for CFD-3 (μs)

Density	MDA	MPA	Baseline
0.01	848570	956870	30486030
0.05	4073110	4832290	172278150
0.10	8127180	9981500	379953640
0.15	12239270	15110690	∞
0.20	16254880	19180300	∞
0.25	20620330	21393360	∞
0.30	24515200	24641760	∞

B.2.7 Query performance based on keyword density with all negative keywords

Table B.18: Running time for CFD-1 (μs)

Density (%)	MDA	MPA	Baseline
0.1	890	963	1845
0.5	710	695	1500
1	713	760	1110
1.5	643	670	1300
2	793	740	1220
2.5	780	670	1140
3	607	635	1490
3.5	942	975	1152
4	980	1380	2000
4.5	740	700	1380
5	600	660	720
5.5	820	760	1060

Table B.19: Running time for CFD-2 (μs)

Density (%)	MDA	MPA	Baseline
0.1	296000	286000	1027000
0.5	289000	285000	2390000
1	678740	680000	1373930
1.5	521000	524000	1674000
2	407000	386000	1709000
2.5	397000	419000	2070000
3	2090330	2100970	4322110
3.5	622000	555000	1804000
4	1831690	1826220	3773060
4.5	2178450	2441407	2192468
5	791000	813000	2033000
5.5	1086740	1069770	2124880

Table B.20: Running time for CFD-3 (μs)

Density (%)	MDA	MPA	Baseline
0.1	497000	444000	8631900
0.5	4585450	4405320	10384980
1	2156150	2263580	6275550
1.5	923000	983700	13982900
2	10643470	10244370	11498040
2.5	11121190	11076300	16605270
3	11409230	11227130	23908850
3.5	15894050	15436930	32179790
4	12406210	12473930	18118060
4.5	20262750	19020530	34702960
5	19735120	18381600	34399800
5.5	14569600	13386450	28133340

B.2.8 Effect of varying distance between s_l and d_l

Table B.21: Running time for CFD-1 (μs)

Density (%)	MDA	MPA	Baseline
2%	994865	1005517	7703000
4%	916310	939057	∞
8%	934115	967367	∞
16%	898915	959023	∞
32%	912095	912200	∞
64%	840020	884157	∞

Table B.22: Running time for CFD-2 (μs)

Density (%)	MDA	MPA	Baseline
2%	2620000	3135000	77723012
4%	2405000	3055000	∞
8%	2750000	3200000	∞
16%	3210000	3995000	∞
32%	3470000	3745000	∞
64%	4060000	4740000	∞

Table B.23: Running time for CFD-3 (μs)

Density (%)	MDA	MPA	Baseline
2%	55782285	56133445	810350000
4%	57731875	58302465	∞
8%	68949950	70007595	∞
16%	69102365	69391260	∞
32%	74646890	69288190	∞
64%	72281560	69840325	∞