Highest Order Voronoi Diagram for Region-Based Spatial Query Processing

Kiki Maulana Adhinugraha

Thesis Submitted

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy (0190)

Faculty of Information Technology
Monash University

November, 2015

Contents

List of Tables

List of Figures

List of Abbreviations

HSVD	Highest Order Voronoi Diagram
VD	Voronoi Diagram
HOVD	Higher Order Voronoi Diagram
RNN	Reverse Nearest Neighbours
RFN	Reverse Farthest Neighbours
(M/B)NN	(Monochromatic/Bichromatic) Nearest Neighbours
(M/B)FN	(Monochromatic/Bichromatic) Farthest Neighbours
GRNN	Group Reverse Nearest Neighbours
GRFN	Group Reverse Farthest Neighbours
CLI	Cell Level Index
FLIP	Fast Labeling with Interchange Position
FINCH	Fast Intersections' Convex Hull
VDRFN	Voronoi Diagram-based Reverse Farthest Neighbours
CHFC	Convex Hull Furthest Cell algorithm

Abstract

A spatial database is a database that is optimized for storing and querying data that represents objects as points, lines or polygons. A spatial query is a mechanism for retrieving objects stored in the database and consists of a specific question with certain parameters in a map. In general, a spatial query is intended to retrieve the objects either as a set of points of interests or a region for the answer. To get the answer, the query can be processed in two different ways: point-to-point calculation or region-based calculation. In point-to-point calculation, the query can be solved by choosing appropriate objects on the map that can be used to answer the query. In region-based calculation, the query can be solved by constructing the region that contains the correct objects that will answer the query. Region-based calculation has one major advantage over point-to-point calculation: this method does not need to check each object one-by-one; hence this method will not suffer from performance degradation where there is a high number of objects.

A region-based calculation method that is commonly used to solve spatial queries is the **Voronoi diagram**. This method divides the map into smaller spaces based on the nearest distance to an object. A Voronoi diagram can mimic human visual intuition, where humans can easily identify whether an object is located inside or outside a closed shape. Even though a Voronoi diagram has been applied widely for various spatial query types, this diagram has some problems, which are: (1) Most queries use a Voronoi diagram only to prune the map to reduce the objects verification time, (2) The region to answer a spatial query cannot be retrieved directly from a complete Voronoi diagram even though the region for a spatial query is part of a Voronoi diagram. Each type of query needs a specific method in order to generate the region.

Therefore this thesis will present a new variation of the Voronoi diagram named highest order Voronoi diagram (HSVD) that can be used directly to identify the region for various types of spatial queries. To show the flexibility of this structure, we applied it to commonly known nearest neighbours and reverse nearest neighbours with their queries variations. We also applied this structure to answer polychromatic queries and extend this method for hierarchical queries. Our analysis shows that the HSVD structure is very flexible and can adapt to various types of spatial queries without having to rebuild the current structure to answer variations in the queries.

Highest Order Voronoi Diagram for Region-Based Spatial Query Processing

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Kiki Maulana Adhinugraha
November 12, 2015

Acknowledgments

I wish to thank foremost to my supervisor Associate Professor David Taniar. Deepest appreciation for his support for my PhD application to Monash University and my scholarship application. I really appreciate his vast knowledge in many area, patience, continues guidance, motivations, encouragement, and rapid feedbacks anytime, day and night, even during his overseas trips or vacations.

I would like to thank my co-supervisor, Dr Maria Indrawan-Santiago for her rapid guidance, reviews and valuable advices for all my works during my PhD candidature. Special thanks to Dr Muhammad Aamir Cheema as my co-supervisor for his wonderful works that influence me working in this topic, and also his guidance during my candidature.

Special thanks also to all my friendly research group of fellow students: Sultan Alamri, Haidar Al-Khalidi, Geng Zhao, Kefeng Xuan, Agnes, Arief, Utari, Tenindra, Ammar, Zhou Shao, and Yathindu. It was fun and I learnt a lot from you guys. Special credit goes to Thao P. Nghiem (Jessy), for her intense supported during candidature.

I would also like to thank Faculty of IT and MIGR, Monash University for providing research facilities and supports. I would like to thank Professor Grahamm Farr, Professor Bala Srinivasan, Dr Nandita Bhattacharjee, Dr Campbell Wilson for their valuable input and discussion through my annual review and pre-submission seminars. I also thank Dr Kinh Nguyen from La Trobe University for his assistance and Bruna Pomella for excellent proofreading this thesis.

Further thanks to Directorate General of Higher Education of Indonesia for the scholarship opportunity and Kopertis 4 - West Java & Banten for their supports. Special thank to the Rector of Telkom University, Professor Ir Mochamad Ashari; Vice Rector-II, Djusnimar Zultilisna; Dean of Faculty on Information Technology, Dr Maman Abdurohman; and all my colleagues in Telkom University. Special credit to Muhammad Arief Bijaksana for introducing me to a great supervisor.

Finally, and most importantly, I would like to express my deepest gratitude to my lovely little Angle Nada who always makes me strong even on the hardest situation. And also my lovely wife Laili, who always gives me all the best support she can give. I sincerely thank to all my families in Indonesia. My father (Dede), my mom (Yetty), my dad-in-law(Ibrahim)(deceased) and my mom-in-law (Titun) for their endless support ever. And my best regards also to all my brothers, sisters, nephews and nieces. This thesis will not be completed on time without your supports.

Kiki Maulana Adhinugraha

Monash University
November 2015

List of Publications

Publications arising from this thesis include:

Kiki Maulana Adhinugraha, David Taniar, Maria Indrawan(2014) Finding reverse nearest neighbors by region. In *Concurrency and Computation: Practice and Experience, Vol 26 Issue 5 2014*, DOI: 10.1002/cpe.3056.

KM. Adhinugraha, D. Taniar, M. Indrawan, DMK. Latjuba (2014) Reverse Nearest Neighbour by Region on Mobile Devices. In *The 28th IEEE International Conference on. Advanced Information Networking and Applications (AINA-2014)* Victoria,Canada.

TP. Nghiem, K. Maulana, AB. Waluyo,D. Green, D. Taniar (2013) Bichromatic Reverse Nearest Neighbors in mobile peer-to-peer networks. In *The 11th IEEE Pervasive Computing and Communication (PerCom-2013)* San Diego, USA.

TP. Nghiem, K. Maulana, K. Nguyen, D. Green, AB. Waluyo, D. Taniar(2014) Peer-to-peer bichromatic reverse nearest neighbours in mobile ad-hoc networks. In *Journal of Parallel and Distributed Computing Volume 74, Issue 11 2014*, DOI:10.1016/j.jpdc.2014.07.007.

Anasthasia Agnes Haryanto, David Taniar, Kiki Maulana Adhinugraha(2015) Processing Group Reverse kNN in Spatial Databases. In *The 29th IEEE International Conference on. Advanced Information Networking and Applications (AINA-2015)* Gwangju , South Korea.

Anasthasia Agnes Haryanto, David Taniar, and Kiki Maulana Adhinugraha Group Reverse kNN Query Optimisation. Accepted for publication in Journal of Computational Science, Special Issue Intelligent Mobility. Elsevier.

Zhou Shao, David Taniar, Kiki Maulana Adhinugraha Range-kNN Queries with Proivacy Protection in a Mobile Environment. Accepted for publication in Pervasive and Mobile Computing Journal.

Zhou Shao, David Taniar, Kiki Maulana Adhinugraha Efficient Range-based Nearest Neighbour Search with User Mobility in a Mobile Environment. Under Review for publication in Soft Computing Journal.

Chapter 1

Introduction

1.1 Aims and Background

1.1.1 Aims

A spatial database is a database that is optimized for storing and querying data that represents objects as points, lines or polygons. A spatial query is a mechanism for retrieving objects stored in the database and consists of a specific question with certain parameters in a map. In general, a spatial query is intended to retrieve the objects either as a set of points of interests or a region for the answer. To get the answer, the query can be processed in two different ways: point-to-point calculation or region-based calculation. In point-to-point calculation, the query is solved by choosing appropriate objects on the map that can be used to answer the query. In region-based calculation, the query is solved by constructing the region that contains the correct objects that will answer the query. Region-based calculation has one major advantage over point-to-point calculation: this method does not need to check each object one-by-one; hence this method will not suffer from performance degradation where there is a high number of objects (Adhinugraha et al., 2013).

A region-based calculation method that is commonly used to solve spatial queries is the **Voronoi Diagram**. This method divides the map into smaller spaces based on the nearest distance to an object. A Voronoi diagram can mimic human visual intuition, where humans can easily identify whether an object is located inside or outside a closed shape (Aurenhammer, 1991). Even though a Voronoi diagram has been applied widely for various spatial query types, this diagram has some problems, which are: (1) Most queries use a Voronoi diagram only to prune the map to reduce the objects verification

time (Xuan, Zhao, Taniar, Rahayu, Safar and Srinivasan, 2011; Yao et al., 2009; Tran et al., 2009; Hu et al., 2010; Safar et al., 2009), (2) The region to answer a spatial query cannot be retrieved directly from a complete Voronoi diagram even though the region for a spatial query is part of a Voronoi diagram (Cheema et al., 2011; Adhinugraha et al., 2013; Stanoi et al., 2001). Each type of query needs a specific method to generate the region.

Therefore this thesis will present a new variation of the Voronoi diagram named the Highest Order Voronoi Diagram that can be used directly to identify the region for various types of spatial queries.

1.1.2 Background

In spatial databases, a map is a limited area that contains features such as points of interest, landmarks, or road networks. A common question that people pose when using a map concerns finding the nearest object of interest. For example, assume that a person is located in a certain area and looking for a restaurant as shown in Figure 1.1. The query posed by this person could be "Where is the nearest restaurant from here?" In a spatial information system, this problem is commonly known as **k-Nearest Neighbour (kNN)** and can be solved by checking the distance from the person to all restaurants and choose the shortest one as the answer, which is commonly known as **point-to-point calculation**.

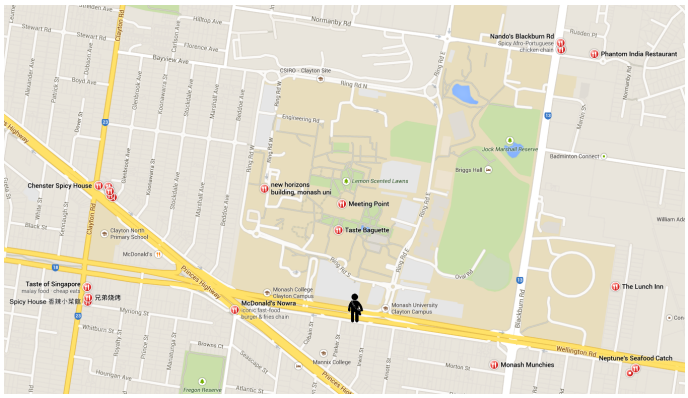


Figure 1.1: Finding Nearest Restaurant around Clayton Campus

Another spatial-based question or query that can be posed to the system is a question asked by a restaurant owner regarding his customers. For example "Do I have customers that consider my restaurant as their nearest restaurant?" To answer this query manually,

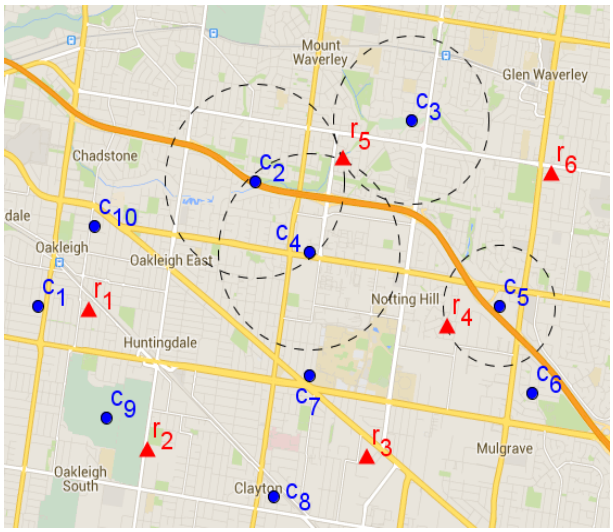


Figure 1.2: Reverse Nearest Neighbour

a restaurant owner needs to identify and locate all the known customers on the map, and compare the distance between the restaurant and each customer to the distance between other restaurants and each of his customers. In other words, he needs to find whether any customer is located closer to his restaurant than to another restaurant. To obtain an answer to this type of query in a spatial information system, the server may perform a kNN query using each customer as the query point and then finding the customers who have his restaurant at the top of the list of the kNN result. This set of customers will consider his restaurant as the closest restaurant from their location. This type of query is called **Reverse Nearest Neighbour (RNN)**.

To illustrate the idea of RNN, consider the example depicted in Figure 1.2. Assume there are five restaurants represented as red triangles r_1 to r_5 and ten customers represented as blue dots c_1 to c_{10}. Assume a query is issued from restaurant r_5. In order to identify which customers consider restaurant r_5 as their nearest restaurant, the server has to perform a kNN query on each customer and obtain his/her first nearest restaurant. In this example, there are three customers (c_2, c_3, c_4) consider restaurant r_5 as their nearest restaurant, while other customers consider other restaurants as their nearest. Hence, the result set for RNN query for restaurant r_5 is the three customers. This is an example of point-to-point calculation in RNN query.

Another approach that can be used is **region-based calculation** to reduce the objects verification phase, where the aim is to find the region where the objects which will answer a particular query are located. This is done by estimating the smallest area that

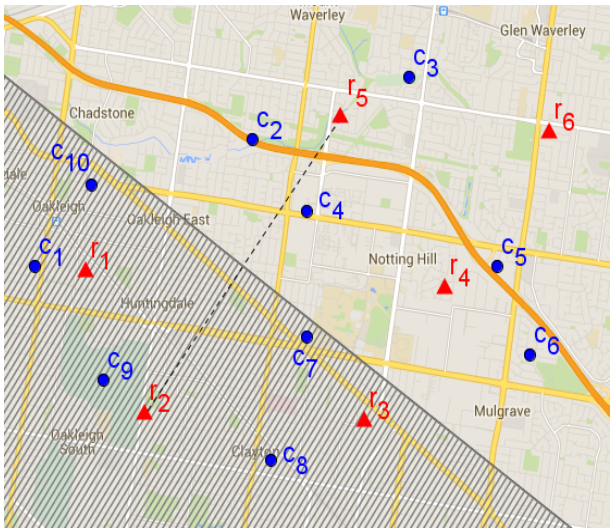
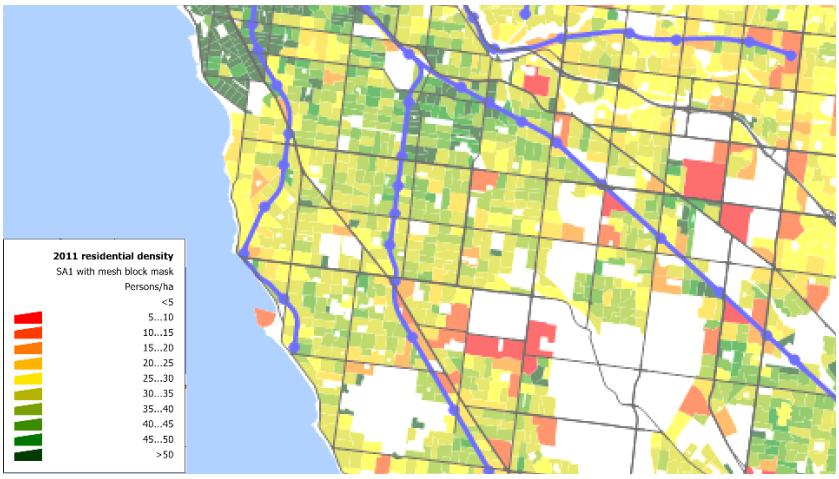


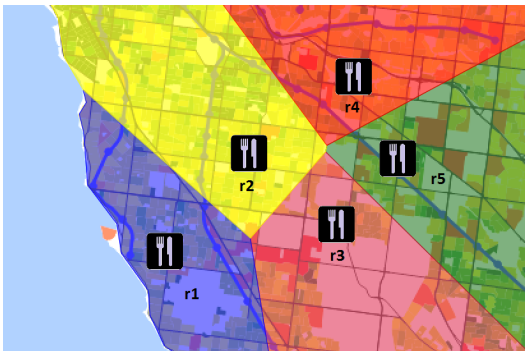
Figure 1.3: Reverse Nearest Neighbour with Region Pruning

contains the candidate objects with the **region pruning** approach followed by the objects verification phase, or **directly** by finding the right region with no objects verification phase. Region pruning is a method in which the main objective is to remove as many unnecessary objects as possible. The question "Who are the customers that consider r_5 as the nearest restaurant?" can be answered by using this logic. Customers who consider the farthest restaurant from r_5 as their nearest will not be considered as the answer for the query. In this example, restaurant r_2 is the farthest restaurant from r_5. All customers who have a shorter distance to r_5 will be considered as the **candidates** while others will not be considered at all. In this example, a straight line between r_5 and r_2 can be used to determine the pruning region as shown in Figure 1.3. As the unnecessary area containing unnecessary objects is pruned, the remaining candidates can be verified to find the answer to the query. The process of finding the right objects from a set of candidates is called the **objects verification** phase. This approach is not effective in reducing unnecessary objects when there are numerous objects on the map.

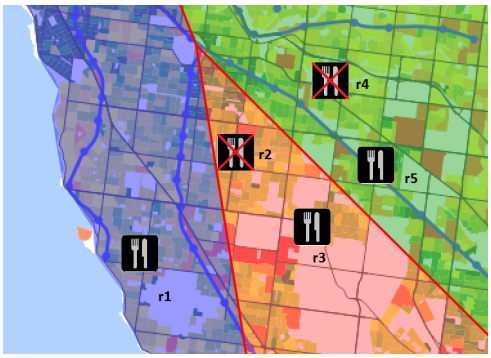
While the point-to-point with region pruning approach seems adequate enough to answer spatial queries, this approach still performs a point-to-point calculation in the object verification phase. Hence this method is not very effective when the number of objects is high. Instead of checking all objects or candidates, another method for obtaining the answer for a query is by **directly** finding the region where the correct objects for a particular query are located.



(a) Original Map



(b) Region of all restaurants



(c) Region of Fast Food Restaurants only

Figure 1.4: Melbourne SE Suburb

Figure 1.4a shows the population density in a South-Eastern suburb of Melbourne in 2011[1] that includes coloured blocks where each color represents average numbers of people lives per hectares. The green color indicates the region with high population density, whereas the red color indicates the region with low population density. To perform market analysis, an analyst can consider the number of people lives per hectares as the number of potential customers per hectares. However there is no information regarding the exact location of each customer on the map, such as address or geo-coordinates; hence, it is not possible to perform point-to-point approach in this situation. In other words, region-based calculation is the better option to answer a spatial query in this case.

A region-based calculation method that is commonly used to solve spatial queries is the **Voronoi Diagram**. This method divides the map into smaller regions based on nearest distance to an object. A Voronoi diagram can mimic human visual intuition, whereby

[1] http://chartingtransport.com/2012/09/21/first-look-2011-density/

humans can easily identity whether an object is located inside or outside a closed shape (Aurenhammer, 1991).

In Figure 1.4b, a Voronoi diagram is used to divide the map into smaller spaces or Voronoi cells. Each restaurant will have its own region, so all the customers that live in a Voronoi cell will consider the restaurant in this region as their nearest restaurant. Knowing this fact, a restaurant owner can analyze market size and compare this with competitors' market size. In other words, if the owner finds that the Voronoi cell where his restaurant is located contains a large coverage of blocked green colour, he would have a large number of potential customers.

In this example, region-based calculation has a great advantage compared to the point-to-point approach, because the former approach does not need objects verification. To answer a query from a restaurant owner "Which customers consider r_2 as the nearest restaurant?" all objects, in this case the customers, do not need to be checked. By using region-based calculation, any customers located in a yellow region will consider restaurant r_2 as their nearest restaurant. From this example, one can see that region r_2 has more green color blocks compared to other regions. Hence, restaurant r_2 has more potential customers that consider this restaurant to be their nearest.

Another spatial query with region-based calculation is shown with this scenario. Assume that all restaurants with odd numbers in Figure 1.4b are fast food restaurants r_1, r_3, r_5. Consider that a fast food restaurant owner r_1 wants to whether the region where his restaurant is located will be considered by customers as their nearest fast food restaurant, but he is not interested in knowing about other non-fast food restaurants as he does not consider them as competitors. The query will be "Where is the region where the customers will consider my fast food restaurant as the nearest?". r_1, r_2, r_3, r_4, r_5 are considered as restaurants; however since restaurants r_2 and r_4 are not fast-food restaurants, their presence will be ignored and the region in the Voronoi Diagram in Figure 1.4b will be invalid. Hence, a new Voronoi Diagram is needed to answer this query.

The new Voronoi Diagram which is constructed from three restaurants is shown in Figure 1.4c. Compared with the previous diagram, the region containing fast food restaurants is bigger because the previous region of non-fast-food restaurants belongs to the nearest fast-food restaurant. This example highlights the situation known as **polychromatic**

spatial queries, a query that involves more than two objects. The objects involved in this example are *fast-food restaurants, non-fast-food restaurants*, and *customers*.

Based on the above needs, this thesis undertakes the following research objectives:

1. It provides a new variation of a Voronoi diagram named **highest order Voronoi diagram** that provides a detailed spatial distance sequence in each Voronoi cell.

2. It provides a generalisation framework to solve the variations of spatial queries by using the regions approach.

3. It presents a framework with region-based approach to solve **nearest neighbour** queries and their variations using a highest order Voronoi diagram.

4. It presents a retrieval index system for the highest order Voronoi diagram that can support **reverse nearest neighbour** queries and their variations

5. It proposes a region-based **polychromatic** query processing that can support nearest neighbour queries and reverse nearest neighbour queries. It also extends this concept into hierarchical query processing for nearest neighbour and reverse nearest neighbour queries. These queries will be answered by using the highest order Voronoi diagram.

1.2 Problem Definitions and Motivations

In region-based calculation, the queries can be answered by obtaining the correct region. The region is a part of the map where any objects located inside will be considered as the answer. To accomplish the main research objectives above, it is important to understand the problems in solving spatial queries by using region-based calculation

- **Challenge 1: Limitation in Voronoi Diagram**

  A Voronoi diagram is a common region-based method to solve spatial databases queries. The region obtained to solve a spatial query is part of the Voronoi diagram; however, not all regions can be retrieved directly from a Voronoi diagram. When $k = 1$, the region for kNN and RNN queries can be obtained directly from a Voronoi cell of an ordinary (order-1) Voronoi diagram. However when $k > 1$, the region cannot be retrieved directly from any higher order Voronoi diagram (Cheema et al.,

2011). Therefore, the first challenge that needs to be solved is "**how to design a custom Voronoi diagram that can be used directly to solve different types of spatial query**".

- **Challenge 2: Requiring Specific Method for Specific Query**

  The region-based approach is an effective way of solving spatial queries in large datasets since this approach will obtain the region that contains only the right objects for a given query. However, in the existing approaches, each region is uniquely created for a specific query and the algorithm to create the region is designed specifically to a particular query type. For example, the *influence zone* (Cheema et al., 2011) creates a region of RNN query that can be used only for RNN query with a single query point. This method cannot be used to solve an RNN query with multiple query points or to find a k^{th} sequence even though these queries fall within the same query type. Therefore, the next challenge that needs to be solved in this thesis is "**how to broaden region-based approach to support various types of spatial queries**"

- **Challenge 3: Creating Region on Demand**

  In region-based approach, a region for a particular query is created after the query is submitted to the system. If the system receives more than one identical query, the system has to recreate the same region repeatedly. Therefore, the next challenge that needs to be addressed in this thesis is "**how to design a region-based approach that can minimize the multiple creations of the same region for identical queries submitted to the system**".

- **Challenge 4: Expensive Cost of Finding the Farthest Region**

  Unlike RNN query, Reverse Farthest Neighbour (RFN) query takes only those candidate objects that have the query point as the farthest neighbour. Hence, the candidate objects must be checked against the query point and all of its competitors (Tran et al., 2009; Agarwal et al., 1992; Supowit, 1990). Defining a reverse farthest neighbour for a query point is challenging since the region can be constructed only if all competitors have also been checked. Therefore, the next challenge is "**how to design an efficient region-based calculation approach to support reverse farthest neighbour query**".

- **Challenge 5: Lack of support for answering Polychromatic Spatial Queries** Polychromatic spatial queries is a generalisation of spatial queries, where the object types involved in the query might be more than two types. Current works in spatial query processing focus only on monochromatic which is single object type and bichromatic which is two object types. Although polychromatic queries can appear naturally in our daily lives, these types of queries have not attracted many researchers. There is only one work in multiple objects query processing and their method is specific only to nearest neighbour queries (Zhao et al., 2009). Therefore, the last problem in this thesis is "**how to design a region-based approach that can address various type of polychromatic spatial queries**".

1.3 Contributions

To address aforementioned research challenges, this thesis introduces a new method in the region-based approach to solve spatial queries. We propose a variant of the Voronoi diagram named highest order Voronoi diagram that can be used to solve various spatial query types. We focus the application of highest order Voronoi diagram on nearest neighbour queries, farthest neighbour queries, reverse nearest neighbour queries and reverse farthest neighbour queries. We also propose further application of highest order Voronoi diagram for polychromatic spatial queries and extend this application into hierarchical spatial queries.

1.3.1 Highest Order Voronoi Diagram

This thesis proposes a new variant of the Voronoi diagram along with its construction algorithm to help solve various spatial queries, called highest order Voronoi diagram (HSVD). Highest order Voronoi diagram is an extension of higher order Voronoi diagram (HOVD); however our proposed diagram has wider possible applications.

Figure 1.5 shows an example of highest order Voronoi diagram with four generator points p_1, p_2, p_3, p_4. Compared with HOVD or Ordered HOVD, a sequence in highest order Voronoi diagram has more spatial information that can be used to solve more types of spatial queries. This thesis also proposes an algorithm to construct a highest order Voronoi diagram called the Fast Labelling and Interchange Position (FLIP) that can be

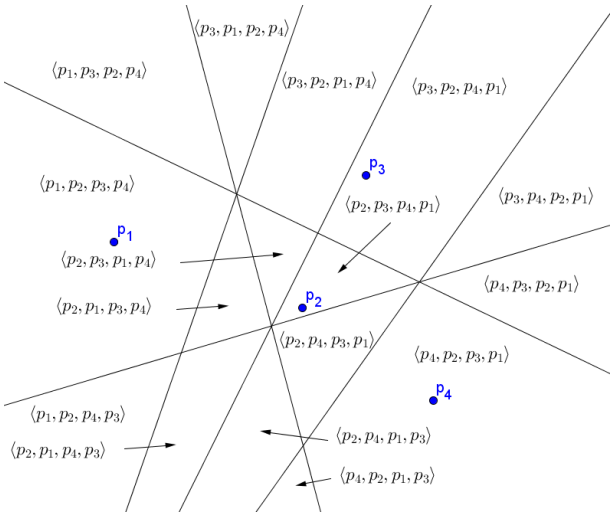


Figure 1.5: Highest Order Voronoi Diagram

used to generate other Voronoi diagram variants, and is not limited to the highest order Voronoi diagram.

1.3.2 Region-based Approach in Nearest Neighbour Query

We applied the region-based approach based on HSVD in some variation of nearest neighbour and farthest neighbour queries. Unlike other nearest neighbour methods, this approach can also be used to identify k^{th} object either in nearest neighbour queries or farthest neighbour queries.

1.3.3 Region-based Approach in RNN Query

We applied the HSVD to various types of reverse nearest neighbour (RNN) queries, such as variation in the number of query points and variation of k value. In the more general form, RNN is expressed as RkNN where k represents the number of nearest facility points from the users location which includes the query point as well. We also introduce $Rk^{th}NN$ query and Group RNN query.

Compared with existing region-based calculation methods, the highest order Voronoi diagram can handle several variations in RNN queries, and at the same time avoiding the repetitive region creation for identical queries.

1.3.4 Region-based Calculation in RFN Query

We also applied the HSVD to reverse farthest neighbour (RFN) queries. Similar to the previous application, the proposed method also can handle variations in RFN queries, such as variation in number of query points and variation in the k values. For RFN variants, we introduce $Rk^{th}FN$ query and Group RFN query.

Compared with existing methods in RFN query, our proposed method can provide a comprehensive region-based solution for a wider range of RFN queries, whereas other methods still rely on the area pruning method that requires further verification to find the correct object within the candidate set to solve RFN problem.

1.3.5 Polychromatic Spatial Queries

In this thesis, we propose a region-based polychromatic query processing, where the number of object types involved in the queries are more than two. This problem has not been intensively studied by other researchers, even though the problems are very relevant in our daily lives. Compared with other methods that can only cover nearest neighbour query with area pruning, our proposed concept can have wider application of polychromatic query, not only in nearest neighbour, but also in reverse nearest neighbour and reverse farthest neighbour queries. We also extend the concept of polychromatic query further to hierarchical spatial queries, where the queries are structured based on the hierarchical structure of the objects. Our analysis shows that both polychromatic and hierarchical spatial queries can be solved with region-based approach by using highest order Voronoi diagram.

1.4 Thesis Structure

An overview of this thesis is illustrated in Figure 1.6. The thesis is structured as follows:

- All related state-of-the art works are discussed in Chapter 2.
- The main contributions are as follows:
  - The main proposed method to address **challenge 1** named highest order Voronoi diagram (HSVD), is discussed in Chapter 3.

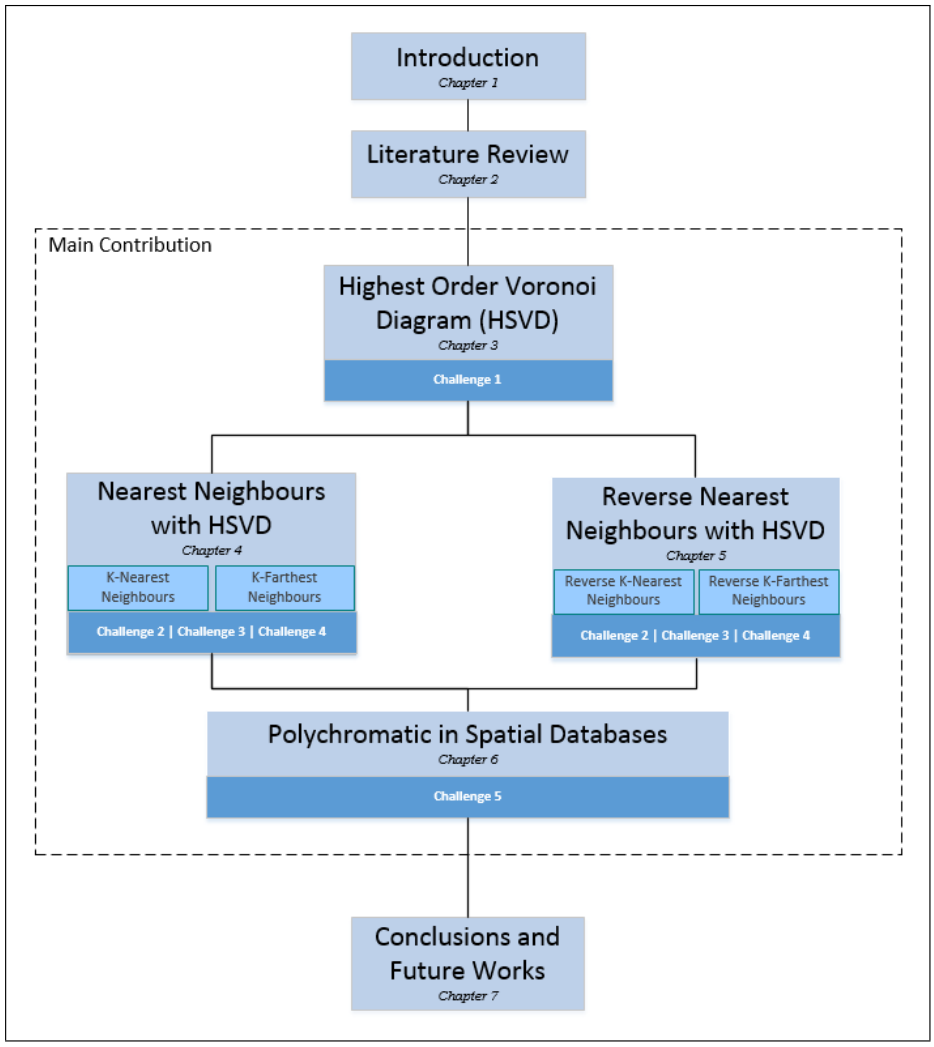


Figure 1.6: Thesis Structure

– The first implementation of HSVD is in nearest neighbour to address queries and their variations is presented in Chapter 4. A nearest neighbour framework is proposed to address **Challenge 2,3**; and one of the variations of nearest neighbour query, named farthest neighbours algorithm based on this framework, is proposed in order to address **Challenge 4** specifically.

– The second implementation of HSVD is in reverse nearest neighbour query presented in Chapter 5. A reverse nearest neighbours framework based on HSVD is proposed in order to address **Challenge 3,4** in a reverse nearest neighbour environment. Another variation of reverse query named reverse farthest neighbour algorithm based on this framework, is proposed in order to address **Challenge 4**.

  - We propose region-based polychromatic and hierarchical query processing and applied this problem in both nearest neighbour and reverse nearest neighbour queries in Chapter 6 to answer **challenge 5**.

- Chapter 7 summarizes the discoveries, evaluation, outcomes and significant impacts in general. In addition, the potential research arising from this research is discussed in this chapter.

Chapter 2

Literature Review

2.1 Overview

This chapter reviews related works in some of the nearest neighbour variations in spatial queries from two perspectives: the query processing perspective and the region-based perspective. Various methods used for spatial query processing will be discussed in section 2.2, where we will highlight some of the query types such as k-Nearest Neighbour, k-Farthest Neighbour, Reverse Nearest Neighbour and Reverse Farthest Neighbour. Query processing from the region-based perspective will be explained in section 2.3, where we classify region-based approach into two categories based on the region created to solve the queries which are *Candidate Region* and *True Region*. The limitation of these previous researchers will be shown in section 2.4 and section 2.6 concludes the chapter. We explain common spatial metrics (*Euclidean distance* and *network distance*) in spatial databases in the next subsection.

2.1.1 Spatial Metrics

In a spatial database, the distance between two objects is determined by a certain metric. Two common metrics are Euclidean distance and network distance. The Euclidean distance can defined as the length of a direct and straight line from one object to another. Let p_1 and p_2 be points in 2-dimensional space where the location of these objects are $p_1(x_{(p_1)}, y_{(p_1)})$ and $p_2(x_{(p_2)}, y_{(p_2)})$. The Euclidean distance between p_1 and p_2 can be expressed as follows:

$$dist_E(p_1, p_2) = \sqrt{(x_{(p_1)} - x_{(p_2)})^2 + (y_{(p_1)} - y_{(p_2)})^2} \tag{2.1.1}$$

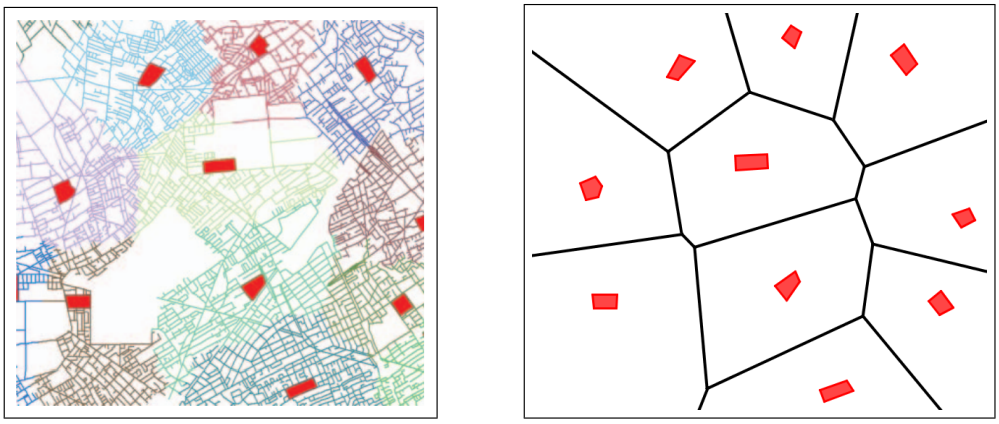
The network distance metric in spatial database refers to a predefined path that objects are restricted to move on such as e.g roads. The network distance between two objects can be defined as the cost of the shortest path between two objects. Spatial road networks can be formally represented by a graph, in which each vertex is an object and each edge is a road or link from one intersection to another. An object in a spatial road network can be a road intersection, turn or object location. Each edge is assigned a weight that is network distance.

(a) Euclidean and Network Distance (b) Road Network

Figure 2.1: Euclidean distance (dashed line) and network distance (solid line) between Monash University and Springvale Train Station

Figure 2.1 shows the Euclidean distance and network distance from Monash University in Clayton to Springvale Station. Figure 2.1a shows the shortest distance in Euclidean space and shortest network distance between these objects. Assuming that we consider only major road segments denoted by yellow lines, the complete road network can be seen in Figure 2.1b. From this figure, it can be seen that there are two alternatives road that can be used. The first one is $o_1 - n_1 - n_2 - n_3 - o_2$ for 6km, whereas the other route alternative is $o_1 - n_4 - n_3 - o_2$ for 5.5km. It is clear that the second alternative is the shortest route between o_1 and o_2.

Network distance and Euclidean distance can also be used to calculate the area between certain points, since area is two dimensional. When network distance is used to construct



(a) Voronoi Diagram in Network Distance

(b) Voronoi Diagram in Euclidean Distance

Figure 2.2: Voronoi Diagram constructed with network distance and Euclidean distance

a Voronoi diagram, we call it a network Voronoi diagram (Okabe et al., 2008; Furuta T., 2005). Figure 2.2 shows the differences between a network a Voronoi diagram and an Euclidean Voronoi diagram generated by elementary schools (red polygon) in Sagamihara, Japan (Okabe et al., 2008). The distance between two schools can be determined by more than one alternative route; hence, the region of a school is the area where all possible network distances consider this school as the nearest, as shown in Figure 2.2a. Meanwhile, Figure 2.2b shows the Voronoi diagram with Euclidean distance, where the region for a school is determined as the area where this school is the nearest regardless of the available route.

Since in network distance we have to identify every possible alternative path which will lead to higher complexity and calculation, in this thesis, we will utilize only Euclidean distance in region-based calculation in solving spatial queries.

2.1.2 Monochromatic and Bichromatic Queries

In a spatial database, there are two types of queries based on the type of query point and the objects to answer the query, named **Monochromatic** and **Bichromatic** query. Each type needs a different method to solve it and has a different purpose as well.

A monochromatic query is a query where the object that issues the query and the answers are the same. Monochromatic means that only a single object type is used in the query. For example, assume that a commercial plane pilot is flying in a certain location and needs to know other aeroplanes within a certain range by means of his radar to avoid

the collision. The Monochromatic query can be seen in Figure 2.3, where the query point for this is an aeroplane and the answer to this query is a set of aeroplanes. Since the object that issues a query and the answer to this query are all aeroplanes, this query is considered to be monochromatic.

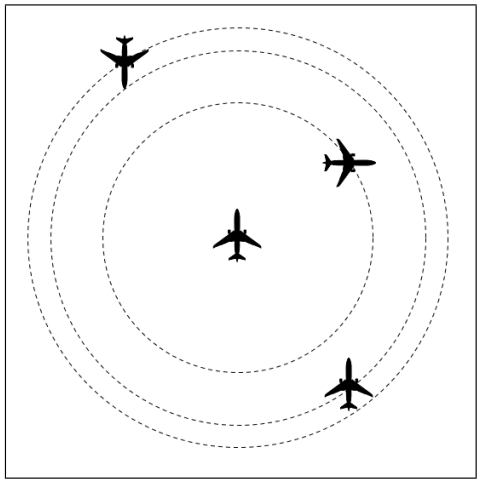


Figure 2.3: Example of Monochromatic Query

A bichromatic query is a query where the object that issues the query is different from the answers to this query. Bichromatic means there are two types of objects involved in the query. For example, assume that the control tower of an airport needs to identify all aeroplanes within a certain area of airspace. The object that issues a query is an airport and the answer to this query is the aeroplanes. Since the object that issues the query is different from the objects answering the query, this query is considered as bichromatic. This query is illustrated in Figure 2.4.

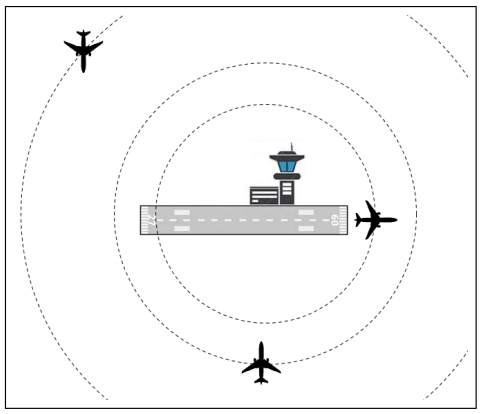


Figure 2.4: Example of Bichromatic Query

In this thesis, we discuss the application of monochromatic and bichromatic queries where a region-based approach can be applied. The following section explains several types of spatial queries and the various methods used to solve them.

2.2 Spatial Query Processing

This section will give an overview of spatial queries processing and also the state-of-the-art methods, which cover the most well-known nearest neighbour queries, followed by farthest neighbour queries. The reverse version of these queries which are reverse nearest neighbours and reverse farthest neighbours will also be explained in this section. In particular, we highlight the region-based methods that have been proposed for each query type.

2.2.1 Nearest Neighbour Queries

A nearest neighbour (NN) query is a method of finding closest neighbours from a given query point. The number of neighbours that can be found is indicated by k value. When $k > 1$, NN is also known as kNN. The nearest neighbour concept is widely applied in many disciplines including, but not limited to data mining, image processing, machine learning, pattern recognition and spatial database. This thesis will focus on the application of the nearest neighbour concept in spatial databases.

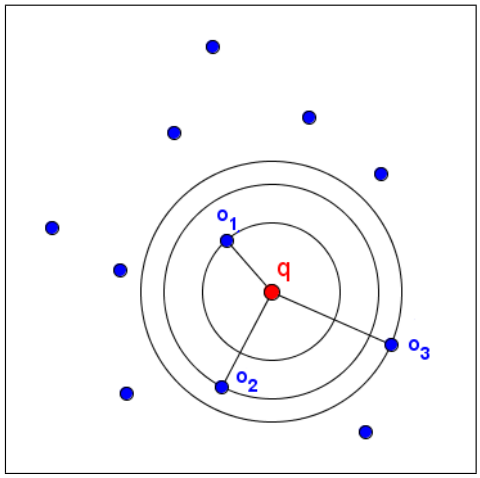


Figure 2.5: Example of 3NN query

Figure 2.5 shows an example of a kNN query from a query point q where $k = 3$. In this example, the main objective is to find the three closest objects to the query point where

the Euclidean distance is used as the metric. The Euclidean distance between an object and the query point is indicated by the circle with q as the center. From this figure, o_1 is 1^{st}NN of q, o_2 is 2^{nd}NN of q and o_3 is 3^{rd}NN of q. Hence, three nearest objects to q are o_1, o_2 and o_3.

A common method used to solve a NN query is by using R-Tree indexing structure or its variants, followed by various pruning methods based on the chosen index structures. Some of the index structures used for kNN pruning are R-Tree (Roussopoulos et al., 1995; Papadopoulos and Manolopoulos, 1997), SR-Tree (Katayama and Satoh, 1997) which is the enhancement of R-Tree, PK-Tree (Wang et al., 2000) which is based on quadtree, PK+Tree (Wang et al., 2005) which is also enhancement of PK-Tree. The basic NN search by using R-tree is by pruning unnecessary candidates to shorten the processing time. The result is the k objects nearest to the query point location.

Another method used to solve nearest neighbour queries is that of area pruning to remove unnecessary objects and concentrate the search process on the remaining candidates. A Voronoi diagram can be used to limit the search area (Kolahdouzan and Shahabi, 2004; Zhao et al., 2011; Xuan, Zhao, Taniar, Rahayu, Safar and Srinivasan, 2011; Xuan et al., 2009; Furuta T., 2005; Safar, 2005), where instead of using an index to prune unnecessary objects, they use a Voronoi diagram to prune the area. A Voronoi diagram consists of a number of Voronoi cells. If a query object q is located in a Voronoi cell, the center of the Voronoi cell is considered as the first nearest neighbour of q as shown in Figure 2.6.

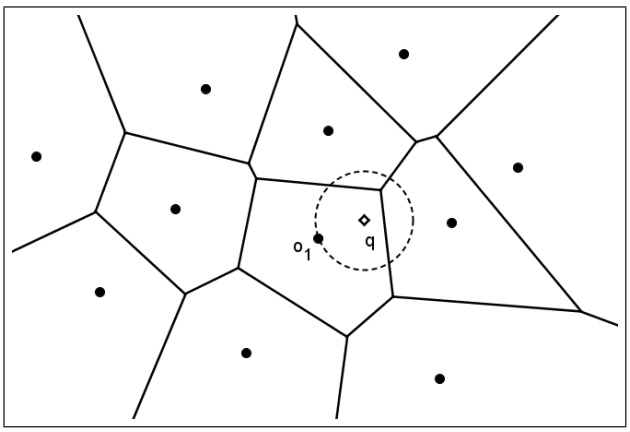


Figure 2.6: Voronoi Diagram for kNN query. Object o_1 has query point q in its cell.

A Voronoi diagram can also be used to solve k nearest neighbour problems by providing the area for k nearest objects (Shamos and Hoey, 1975a; Lee, 1982; Sharifzadeh and Shahabi, 2010). They use the generalization of a Voronoi diagram, called Voronoi diagram

order k, where each cell has k nearest objects instead of 1. Figure 2.8 shows how to solve a 3NN query with Voronoi diagram order-3. As we can see from this diagram, a query point q issues a 3NN query. Since this query is located in a Voronoi cell where o_1, o_2, o_3 are the generator points, these points are also 3NN of q. When q' issues the same query from another location in the cell, the answer to this query will remain the same. Authors (Zhao et al., 2009) discuss how to identify nearest neighbours objects from multiple object types by either stacking multiple Voronoi diagrams for each object type, or using a Voronoi diagram created from multiple object types. Since they use only the order-1 Voronoi diagram, they have to use a specific expansion method to find k nearest objects around the query points.

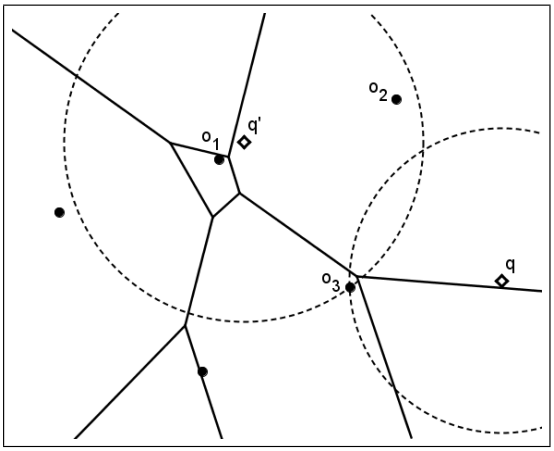


Figure 2.7: 3NN query with Voronoi diagram order-3

A Voronoi diagram also has a "dual tessellation", called the Delaunay triangulation (Okabe et al., 2009). The authors in (Hu et al., 2010) use pre-computed Delaunay triangulation to create a region which can be used for kNN query verification as seen in Figure 2.8. While this method can directly verify the nearest neighbour for $k = 1$, this method cannot be directly used when $k > 1$. Authors (Sharifzadeh and Shahabi, 2010) apply and index to this diagram and solve more query types kNN, RNN and k-ANN.

Most of the recent works related to nearest neighbour problems are focused on obtaining the nearest objects with predefined k, and the methods vary using either index structures or area. Both methods have the same aim: to prune as many unnecessary objects as they can, and the answer to a nearest neighbour query can be retrieved using post-processing methods. Only (Shamos and Hoey, 1975a; Lee, 1982) use the Voronoi diagram order-k to identify the region of kNN.

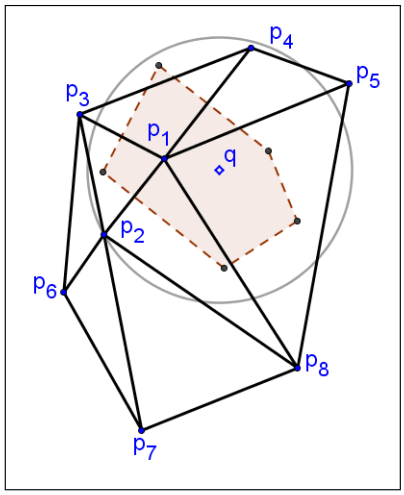


Figure 2.8: 3NN verification using Delaunay Triangulation

Nearest neighbour query is a common one related to mobility of the objects where the purpose of this query is to find the object nearest to the query point (Xuan et al., 2009; Xuan, Zhao, Taniar, Safar and Srinivasan, 2011) or nearest objects during the movement of the query point (Xuan, Zhao, Taniar, Rahayu, Safar and Srinivasan, 2011; Zhao et al., 2010, 2011, 2013) in a server based system, or in a peer-to-peer system (Nghiem and Waluyo, 2011; Nghiem, Waluyo and Taniar, 2013; Nghiem, Green and Taniar, 2013)

2.2.2 Farthest Neighbour Queries

A farthest neighbour (FN) query is a method of finding the farthest object from a given query point. The number of objects is indicated by a predefined k value. When $k > 1$, FN is also known as kFN. Unlike kNN, kFN does not attract too many researchers and implementations due to a higher computation cost.

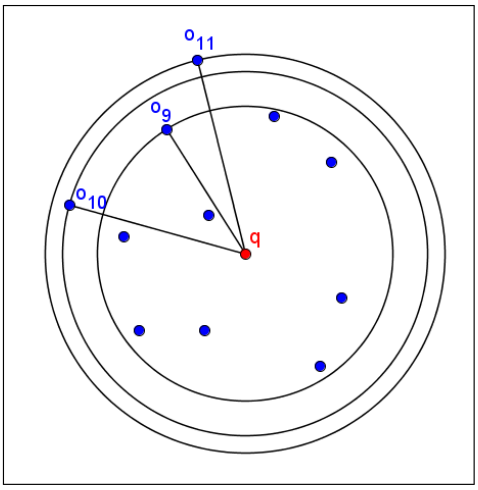


Figure 2.9: Example of 3FN query

Figure 2.9 shows an example of a kFN query from a query point q where $k = 3$. The main objective of kFN query is to find the k objects farthest from query point. From this figure, o_{11} is the 1^{st} farthest object since there are no other objects outside the circle of o_{11}. Object o_{10} is 2^{nd}FN of q since there is only one object outside the circle of o_{10}, and object o_9 is $3rd$FN of q. Hence, the three neighbours farthest from q are o_{11}, o_{10}, o_9.

The authors in (Shamos and Hoey, 1975a) propose a farthest-point Voronoi diagram to identify a farthest point problem as seen in Figure 2.10. By using this diagram, the authors define a Voronoi cell as the farthest region and each region will have a unique farthest point. No two regions will have the same farthest point. Let the diamond icons be the customers and the black dots are the restaurants. The customer q issues a query to find the farthest restaurant. Since q is located inside Region 7 and the farthest point from Region 7 is p_8, then the farthest restaurant from q is restaurant p_8. A higher order Voronoi diagram is also used by (Katoh and Iwano, 1992) to solve bichromatic k farthest neighbour problems.

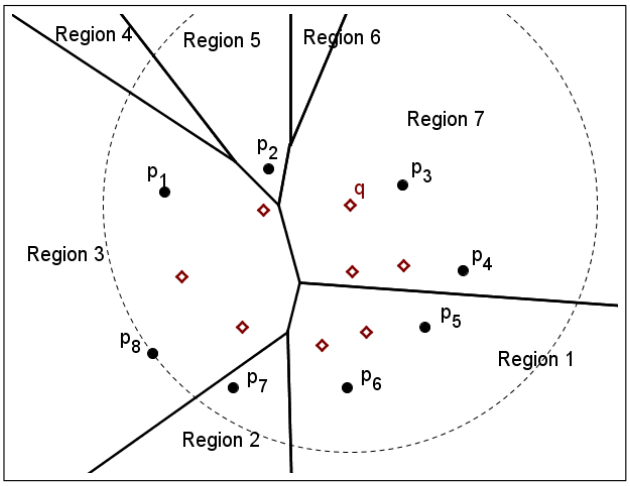


Figure 2.10: A Farthest-point Voronoi diagram

In the point-to-point approach, authors in (Supowit, 1990) use the multidimensional divide-and-conquer method to define the farthest problems, in the presence of deletions. Authors (Agarwal et al., 1991) use a randomized algorithm to solve bichromatic farthest neighbour problems.

2.2.3 Reverse Nearest Neighbour Queries

Reverse nearest neighbour (RNN) concept was firstly proposed by Korn *et al* (Korn and Muthukrishnan, 2000) where the idea is to find the objects that consider the query point

as the nearest. The answer to RNN query can be obtained by pre-calculating kNN query for each data object with predefined k value. The answer to the RNN query will be all objects that have q as part of their k nearest neighbour. The method used to solve this query can be seen in Figure 2.11. Consider red triangles $\{p_1, p_2, p_3, p_4, p_5, p_6\}$ are set of facility points and blue circle $\{o_1 ... o_{10}\}$ are objects of interest. Let p_5 be the query point and predefined $k = 1$, $R_1NN(p_5)$ can be referred as finding all objects of interest that consider p_5 as the nearest neighbour. By running kNN pre-calculation on each object, the answer to this query will be all objects that have p_5 as their kNN result. From this figure, $\{o_4, o_5, o_6\}$ are the answer to this query

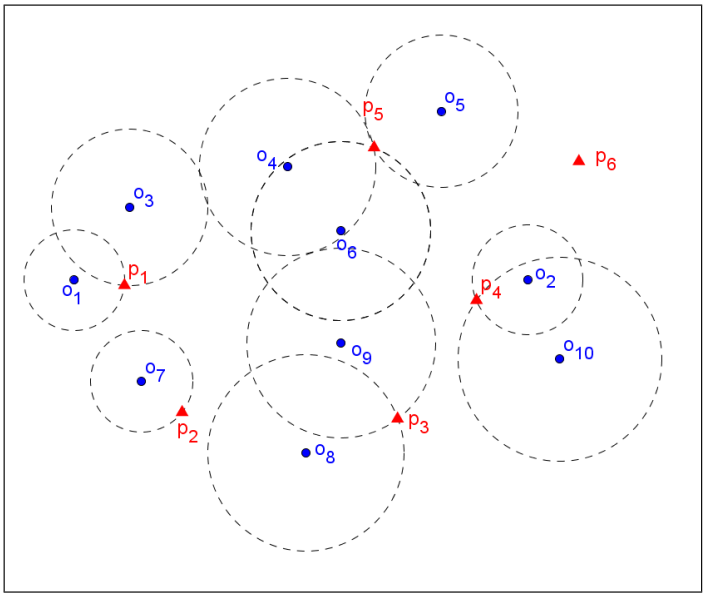


Figure 2.11: Pre-calculation in RNN query

However, conducting kNN pre-computation for all objects on every RNN query is not an efficient approach. To avoid the need for pre-computation for all objects, some authors use space pruning approach to remove objects that will not be the answer to the query. However, these methods cannot eliminate all unwanted objects. Hence, the verification step for the remaining objects called 'candidates' will be needed to obtain the answer to the query. Authors (Stanoi et al., 2000) divide the space into six equal regions from a query point q as shown in Figure 2.12. In this figure, assume that the query point is p_5. This method divides the space into six equal areas (S1..S6) where the query point p_5 is the center. In each area, the nearest object from p_5 will be considered as the candidates. o_5 in S1, o_4 in S4, o_6 in S5 and O_{11} in S6, while there are no nearest objects in S2 or S3. Meanwhile, non-candidate objects will be discarded and the verification process will

be conducted only for candidate objects. Object o_2 will not be considered as the answer because this object has p_4 as the nearest neighbour. Objects o_4, o_5, o_6 are considered as the answer to this query since these objects consider p_5 as the nearest.

This query type can also be used with the indexing of structures approach. The index is used to identify the objects' distances and the query will be based on a predefined distances index. Several indexing structures have been proposed, such as RdNN-Tree (Yang and Lin, 2001), MR$_k$NNCoP-Tree (Achtert et al., 2006b), IUR-Tree (Lu et al., 2011), AMR$_k$NN-Tree where the solution is based on approximation distances (Achtert et al., 2007). Some authors also apply the indexing method to the Voronoi structure. For example, such as (Yan et al., 2012) use a Voronoi cell index to answer RNN queries relating to land surfaces, VoR-Tree index (Sharifzadeh and Shahabi, 2010) that can be used for solving kNN, RkNN and k Aggregate NN (kANN) and VNR-Tree (Slimani et al., 2011).

This type of query can also be solved using the approximation approach such as Boolean Range Query (BRQ)(Singh et al., 2003), Monochromatic/Bichromatic Probabilistic Reverse Skyline (MPRS/BPRS)(Lian and Chen, 2008), MaxOverlap(Wong et al., 2009). Author (Stanoi et al., 2001) propose approximation through Influence Sets, whereby an initial region is created based on surrounding points and then the region is refined to obtain the proper region. Authors in (Achtert et al., 2006a) use approximation of the nearest-neighbour-distances in order to prune the search space. Authors in (Figueroa and Paredes, 2009) use predefined permutation index to select some nearby objects from the query point and then followed by approximation algorithm to refine the results.

Authors (Tao et al., 2004; Wu et al., 2008b) use the property of perpendicular bisectors to prune the search space. Consider the example in Figure 2.13 where a bisector line between p_i and p_j is shown as $Bis(p_i : p_j)$ which divides the space into two half-spaces. The half-space that contains p_i is denoted as $H(p_i, p_j)$ and the half-space that contains p_j is denoted as $H(p_j, p_i)$. Any objects that are located in $H(p_i, p_j)$ will consider p_i as being closer than p_j. Assume that the query point $q = p_5$ and $k = 1$, then any objects that are located in the half-space of p_5 will be considered as the candidates. In Figure 2.13, all objects in the shaded area will be considered as the candidates, while in the white area they can be pruned. After all non-candidate objects have been pruned, TPL method uses R-tree to verify the candidates in unpruned space in order to obtain an answer to

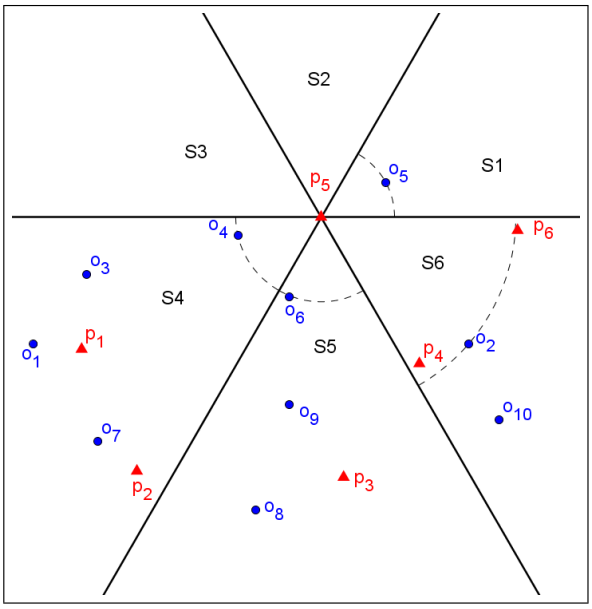


Figure 2.12: Six-regions pruning

the query (Tao et al., 2004). Meanwhile, instead of using bisector lines to directly prune the area, FINCH uses a convex polygon that approximates the unpruned area (Wu et al., 2008b). Meanwhile, pruning rule based on R-Tree is proposed by (Cheema et al., 2010) to solve R$_k$NN with uncertain datasets.

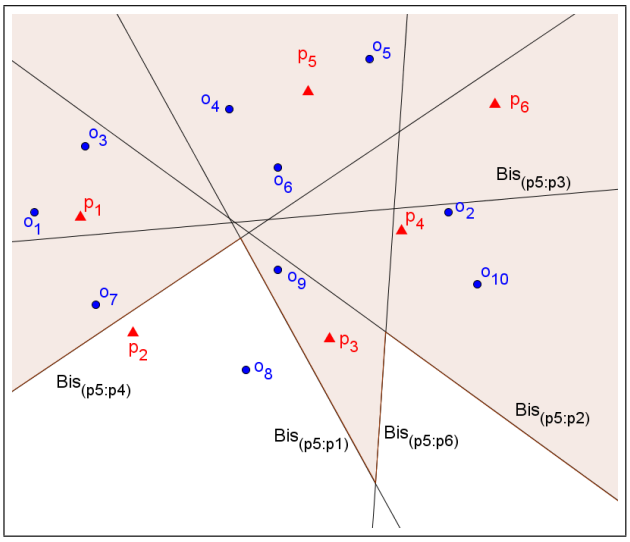


Figure 2.13: Half-space pruning in TPL and FINCH

Authors (Bohan and Xiaolin, 2009; Safar et al., 2009; Tran et al., 2010) use the properties of a Voronoi diagram to solve the RNN query. If a generator point in a Voronoi diagram is considered as the query point, then the Voronoi cell for this generator point is considered as the region of $R1NN$ query and all objects in this region will be considered as

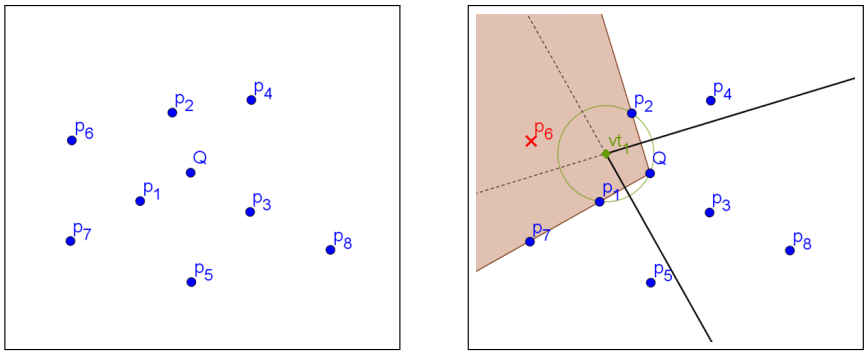
the answer to $R1NN$ query. When $k > 1$, the candidates can be searched from neighbour cells around a Voronoi cell of query point.

Another method of finding a solution to RNN queries is by identifying the region for RNN query where any objects located in this region will be considered as the answer for the query. Although this method employs a half-space pruning technique, unlike previous methods, a region-based calculation will not have candidates objects after the pruning is finished. Authors (Cheema et al., 2011) proposed an Influence Zone, where this method defines the region for $RkNN$ queries. Authors (Adhinugraha et al., 2013) proposed a Contact Zone to effectively identify prunable objects as shown in Figure 2.14.

Some authors have also proposed methods that deal with mobility in RNN queries. An indexing system can be used to monitor the movement of the objects, such as TPR-tree (Benetis et al., 2002, 2006) or P2PRdNN tree (Chen et al., 2006). In order to minimize processing cost, the region pruning approach for mobility in an RNN query is proposed by (Kang et al., 2007; Xia and Zhang, 2006; Emrich et al., 2010; Wu et al., 2008a). However these methods requires an excessive objects refinement step, and author (Cheema, Zhang, Lin, Zhang and Li, 2012) proposed a better solution by using *safe region*. Author (Adhinugraha et al., 2014) applied *Contact Zone* to obtain the region for an RNN query from a mobile device and initial region of *Contact Zone*is applied in a peer-to-peer system (Nghiem, Maulana, Waluyo, Green and Taniar, 2013).

2.2.4 Reverse Farthest Neighbour Queries

Reverse farthest neighbour is a method used to find objects that consider the query point q as the farthest neighbour. This method is based on the farthest point concept that has been applied in k farthest neighbours problems (Shamos and Hoey, 1975a; Agarwal et al., 1991; Katoh and Iwano, 1992). Unlike the RNN query types that has attracted so much attention, the RFN query has not received much attention due to high computational cost. In the RFN query, an object will be considered as the answer to the query if the distance from this object to any other points p_i is less than the distance to a query point q. Figure 2.15 shows an example of RFN that can be solved by using either the point-to-point approach or the region approach. In Figure 2.15a, let the query point $q = p_6$. An RFN query from p_6 will find all objects o_i that are the longest distance to p_6. From this figure, $o_1, o_3, o_4, o_6, o_7, o_8, o_9$ are the objects that have the longest distance to point p_6



(a) Query point is set

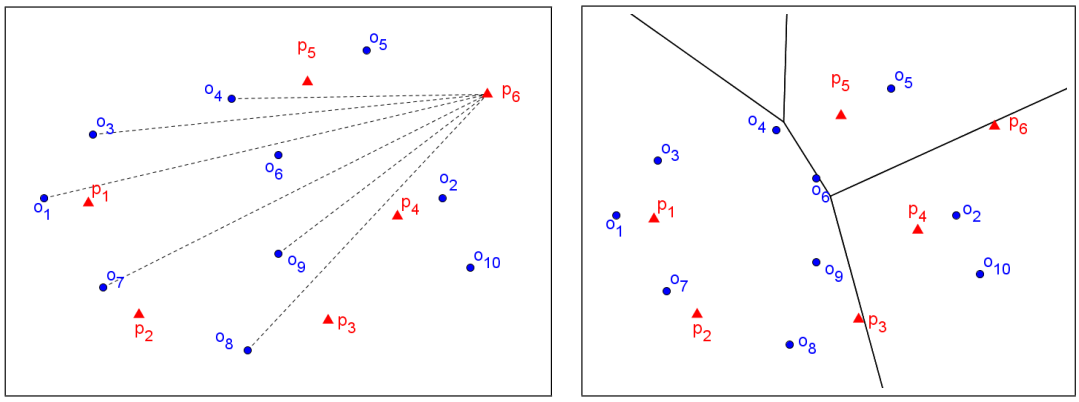
(b) p_6 is eliminated by CZ from vt_1

(c) p_7, p_8 are eliminated by CZ from vt_2, vt_3

(d) RNN Region

Figure 2.14: Contact Zone for RNN Query

than to any other points. Hence, these objects are RFN of p_6. Figure 2.15b shows how RFN can be solved by using farthest-point Voronoi diagram (Okabe et al., 2009). From this figure we can see that all objects for $RFN(p_6)$ are located in the same cell, and this cell considers p_6 as the farthest point.



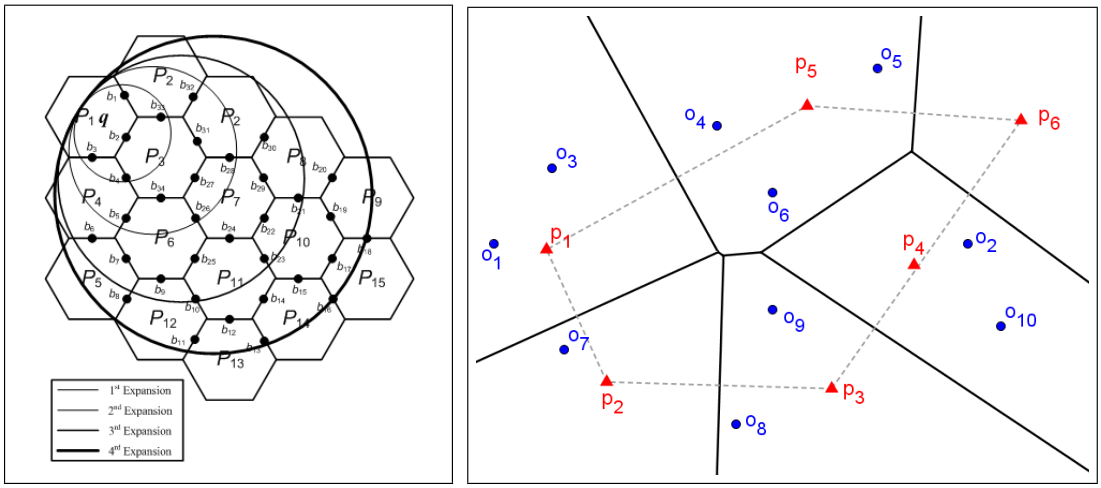
(a) RFN with point-to-point

(b) Farthest-Point Voronoi Diagram

Figure 2.15: Example of RFN

RFN queries are mentioned as the counterparts of RNN queries in (Kumar et al., 2008) and referred to as reverse proximity problems. The authors use an approximation polygon

to obtain the candidate objects to answer the query. Another approach is to use a Voronoi diagram shown in Figure 2.16. Authors (Tran et al., 2009) use ordinary Voronoi diagram with adjacent cells expansion to identify the objects as shown in Figure 2.16a. In this method, an ordinary Voronoi diagram is used. They find the cell that contains query point q and expand to the adjacent cells to find the candidates of RFN objects. Authors in (Liu and Yuan, 2013) combine a Voronoi diagram with convex hull and use the VDRFN algorithm to solve RFN instead of using a farthest neighbour Voronoi diagram.



(a) Voronoi Diagram expansion (b) Voronoi Diagram with Convex Hull

Figure 2.16: Voronoi Diagram for RFN query

Authors (Yao et al., 2009) use a farthest Voronoi diagram and convex hull properties combined with an R-Tree based algorithm called CHFC to solve RFN queries. The convex hull is used to verify whether a query point will have farthest objects or not, then the objects are retrieved from the farthest-point cell in the farthest-point Voronoi diagram. This method was revised by (Liu et al., 2010) who used convex hull and an external pivot to construct a metrics index for storing the distances, known as PIV algorithm. This method was further revised by (Liu et al., 2012) with PIV+ by implementing a safe area to speed up RFN query processing.

2.2.5 Discussion

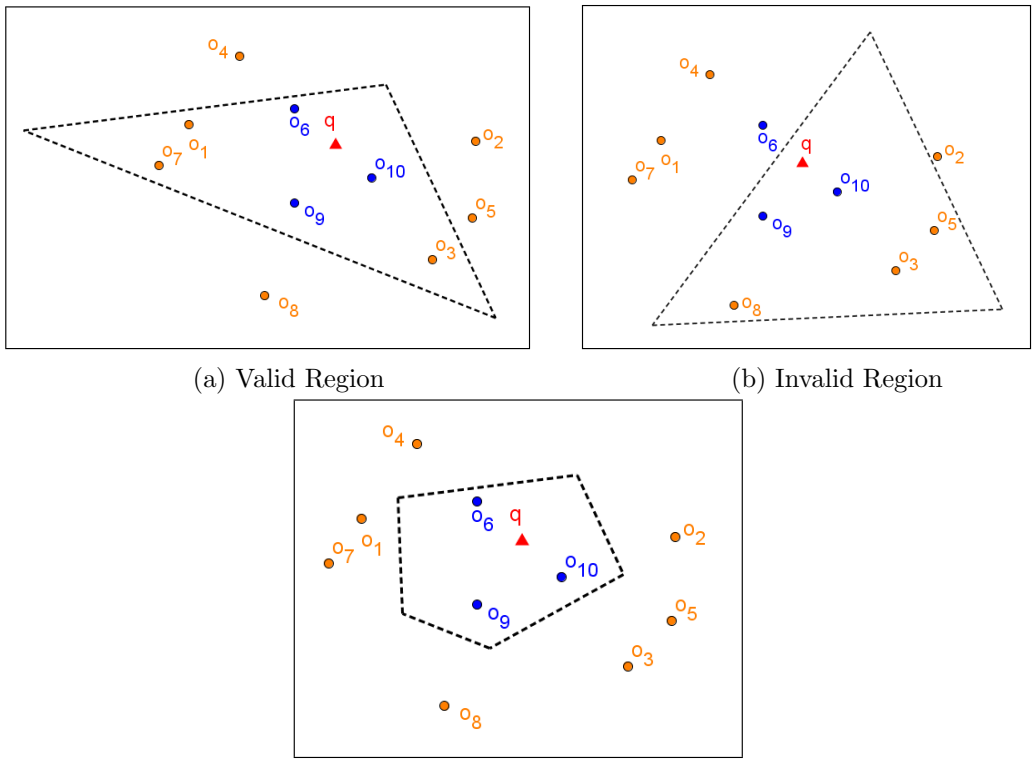
Current methods for solving spatial queries (nearest neighbour, farthest neighbour, reverse nearest neighbour and reverse farthest neighbour) rely on a point to point approach where the effort can be minimized by using an index or region pruning. Only a few methods use the region approach where the answer to a query is the objects inside the region.

Furthermore, most of the methods proposed are applicable only to a specific query type with a specific condition; k, query point location, number of objects. Very few of the methods can be used to solve more than one type of query. Overall, our work differs from all of the previous research as it proposes a new Voronoi structure that can be used to identify the exact region of kNN, kFN, RkNN and RkFN queries without the objects verification phase. Furthermore, with the highest order Voronoi diagram we also extend our solution to cover k^{th}-order, group queries and also polychromatic spatial queries.

2.3 Region-based Approach for Spatial Query Processing

Region-based approach for spatial queries is a method used to retrieve the region where the candidate objects which answer a specific spatial query are located. Candidates objects are temporary objects before the exact objects have been found. When the region contains all correct objects to answer the query, the region will be considered as a valid region, and will be invalid otherwise. In region-based approach, some methods need a post-processing step to verify the objects, while other methods do not. In order to perform the objects verification phase, the region-based calculation methods are always used to generate a valid region.

Figure 2.17 shows examples of regions to solve a query. Assume that a query point q issues a query; blue objects are the objects to answer query $\{o_6, o_9, o_{10}\}$ and will be called **true objects**, and orange objects are not the answer for query $\{o_1, o_2, o_3, o_4, o_5, o_7, o_8\}$ and will be called **false objects**. **Candidate objects** are the set of objects that must contain all true objects. The region indicated by the dashed line is an area within the map that contains candidate objects. Figure 2.17a shows the valid region because this region contains true objects, while Figure 2.17b shows an invalid region, because one of the true objects, o_6, is not located in this region. When there are only true objects in the region, this region is called **true region**, while if the region contains candidate objects, this region will be called **candidate region**. In order to answer a spatial query with region-based approach, the method relies on candidate region or true region. Unlike true region that does not need the object verification step, candidate region may contain false objects in the answer, and therefore, further verification processes are needed.



(a) Valid Region (b) Invalid Region

(c) True Region

Figure 2.17: Example of Regions

The spatial data for query processing, however, can be in raw data or data that has been processed through some partitioning/indexing method. In the query processing, we will not include the initial data treatment as part of query processing, but as a variation of the data sources that can be used for spatial query processing. The whole process of region-based spatial query processing can be seen in Figure 2.18.

In this section, we discuss the region-based approach in spatial query processing where the regions created are a *Candidate region* or *True region*.

2.3.1 Query Processing with Candidate Region

In query processing using candidate region, the first step is to obtain the *initial region* that contains candidate objects to start the object verification phase. This can be done by reducing the size of the map, or using a partitioned map and performing the query processing on particular cells. We divide the methods in this group into 3 subgroups, which are: (1) space pruning method, (2) uniform space division method, (3) structured space division method.

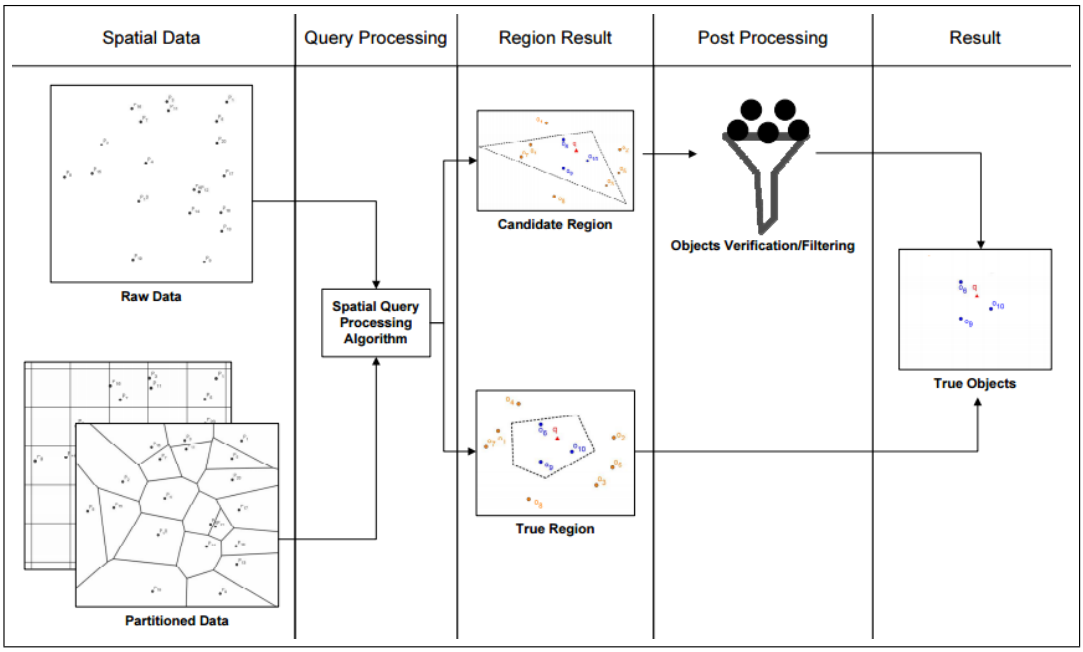


Figure 2.18: Region-based Spatial Query Processing

Space Pruning

The space pruning method is a means of reducing space by using a query point position related to other points around it. A common space pruning method is halfspace pruning which uses a 'perpendicular bisector line' between two points and this line divides the space into two equal sub-spaces. The space pruning method uses unpartitioned spatial data. Assume that there are two points p_1, p_2 and the perpendicular line between these points is $Bis(p_1 : p_2)$. The distance from p_1 to the bisector line is always equal to the distance from p_2 to the bisector line. Halfspace p_1 of p_2 denoted as $H(p_1, p_2)$ is the region where p_1 is the nearest point, while halfspace p_2 of p_1 denoted as $H(p_2, p_1)$ is the region where p_2 is the nearest point. The concept of halfspace can be seen in figure 2.19.

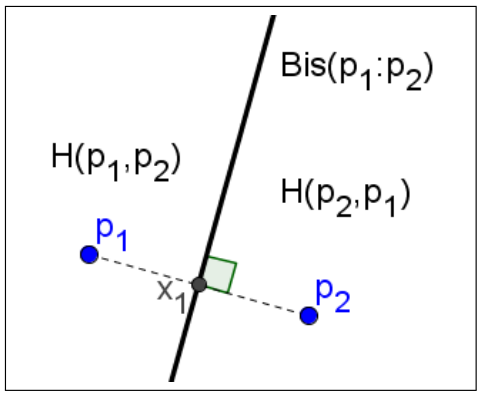


Figure 2.19: Halfspace Pruning

The first halfspace pruning was proposed by (Stanoi et al., 2001) with Influence sets approximation, and refined by (Tao et al., 2004) with the TPL algorithm. This approach was then refined by (Wu et al., 2008b) with FINCH, intended to define an initial region for further objects verification. Authors (Cheema et al., 2009) propose *Lazy Update*, a region to monitor movement for RNN query. Authors (Kumar et al., 2008) create an approximate initial NN region and then use halfspace pruning to refine the region.

Uniform Space Division

Uniform space division is a method that divides the space into several partitions of uniform size, and the objects that can be used to answer the query will be searched within each partition. The space can be considered as the spatial data source, or can be created during the spatial query processing. There are two different types of uniform partitioning methods: pie-slice partitioning and grid partitioning. Authors (Stanoi et al., 2000) divide the space into six regions where the query point q is the center as can be seen in Figure 2.12, and the objects verification will be performed on each region. Authors (Sharifzadeh and Shahabi, 2009) propose the AVC-SW, an approximate algorithm to create a Voronoi cell, where the space is divided into k sectors to create an approximate region for a Voronoi cell as can be seen in Figure 2.20. Authors (Huang et al., 2007) propose an S-grid partition for spatial network that can be used to solve kNN queries. Figure 2.21 shows the partition of a road network into 2x2 cells and the index structure for the grid itself.

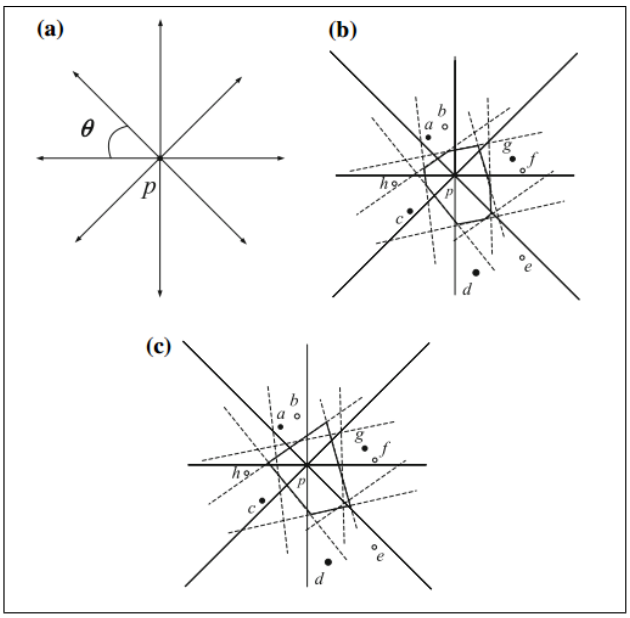


Figure 2.20: Example of AVC-SW with 6-sectors division

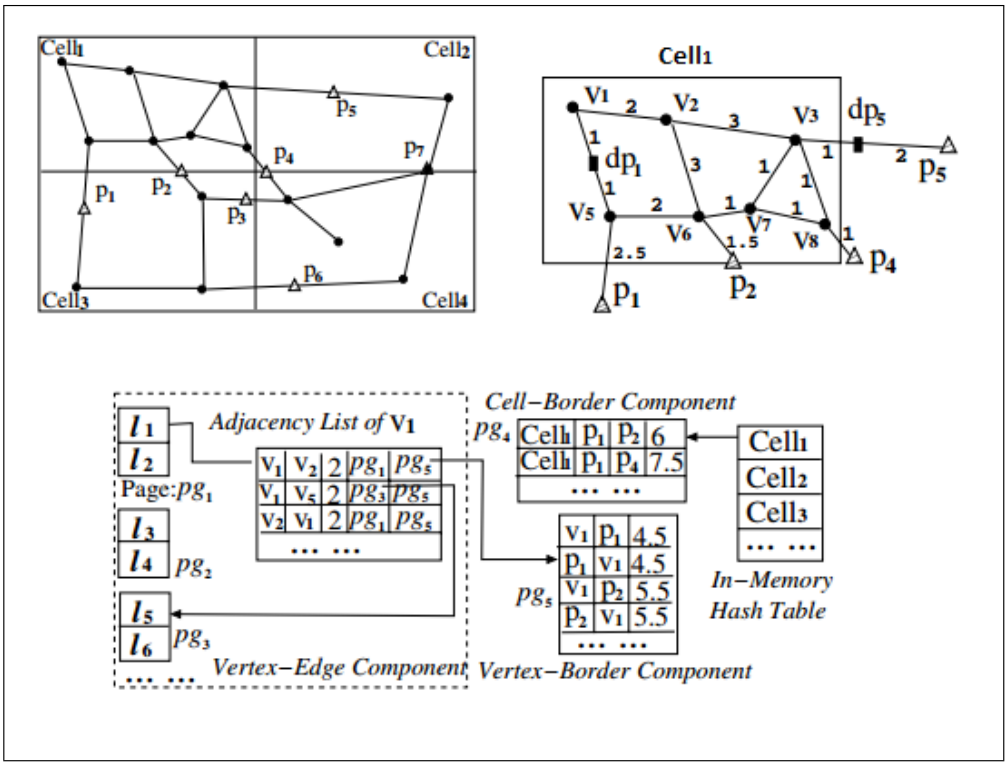


Figure 2.21: Example of S-Grid

Structured Space Division

Structured space division is a method of space partitioning where the partition size is not uniformly divided, but depends on the position of the objects on the map. The structured partition can be defined as a spatial data source or it can be created during spatial query processing. A commonly-used space division is the *Voronoi diagram*, a space division method based on the position of the generator points (Okabe et al., 2009). A Voronoi diagram can be used in the form of *Ordinary Voronoi diagram* to solve spatial queries (Xuan, Zhao, Taniar, Rahayu, Safar and Srinivasan, 2011; Safar et al., 2009; Tran et al., 2010, 2009; Kolahdouzan and Shahabi, 2004) or higher order Voronoi diagram (Yao et al., 2009; Agarwal et al., 1992; Katoh and Iwano, 1992; Smid, 1995; Liu and Yuan, 2013). Ordinary Voronoi diagram is also known as order-1 Voronoi diagram. Since this diagram only provides support for $k = 1$, further post processing methods are needed to answer spatial query where $k > 1$. A common solution for $k > 1$ is by checking neighbouring Voronoi cells around the cell where the query point is located (Xuan, Zhao, Taniar, Rahayu, Safar and Srinivasan, 2011; Kolahdouzan and Shahabi, 2004; Zhao et al., 2011). This diagram itself can be used either as a full Voronoi diagram or as a Voronoi cell (Cheema et al., 2013; Yan et al., 2012; Zhang et al., 2003).

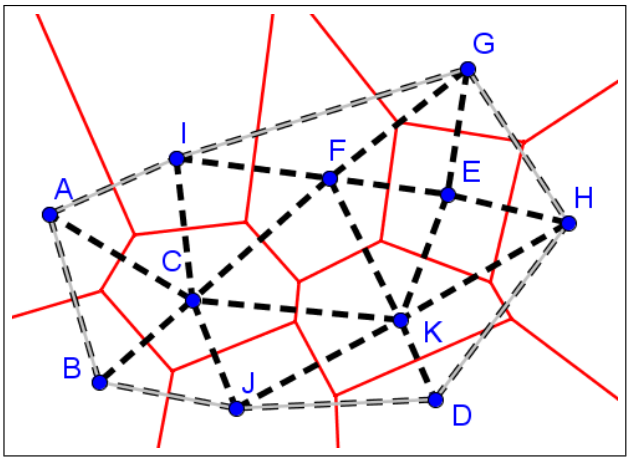


Figure 2.22: Example of Voronoi diagram with Delaunay Triangulation and Convex Hull

Some authors also combine the Voronoi diagram with indexing techniques to enhance its performance (Sharifzadeh and Shahabi, 2010; Slimani et al., 2011; Demiryurek and Shahabi, 2012). Meanwhile, another form of Voronoi diagram, called *Delaunay triangulation* is also used to solve spatial queries. *Delaunay triangulation* shows nearest neighbours for each generator point in a Voronoi diagram (Hu et al., 2010). Hence, it is closely related to nearest neighbour queries. The outer path in Delaunay triangulation is called 'Convex Hull', where this diagram shows farthest relationship for generator points. Hence, this method is used to solve Farthest Neighbour queries (Liu et al., 2012; Liu and Yuan, 2013). Figure 2.22 shows a Voronoi diagram from 11 generator points (shown in solid red lines) along with Delaunay triangulation (shown in dashed red) and Convex Hull (shown in solid gray lines).

2.3.2 Query Processing with True Region

True region is a region without false objects. Since there are no false objects in a true region, this method does not need any objects verification phase. There are only a few works that use this method. Authors (Cheema et al., 2011; Cheema, Zhang, Lin and Zhang, 2012) use Influence zone to identify RkNN region. Author (Adhinugraha et al., 2013) use Contact Zone to create the region of RNN where the region itself can be identified as open region or closed region. This method then is adapted in mobile environment (Adhinugraha et al., 2014). Authors (Nghiem, Maulana, Waluyo, Green and Taniar, 2013) use initial region to create an RNN region in mobile peer-to-peer environment. Authors

(Agnes et al.,2014) aggregate multiple RNN queries and propose a method to identify the region of Group RNN queries.

2.3.3 Discussion

The current research on the region-based approach can be divided into two categories. The first one uses region as a way to reduce unnecessary objects before the answer to the query can be retrieved. The second one that use region to obtain the exact region where the objects to answer the query are located in order to avoid the objects verification phase. Our work is focused on answering queries by identifying the exact region for each query. Our work will be based on partitioned data with a highest order Voronoi diagram (HSVD), which creates partitions based on points located in the space, and distance order information is added in each partition. Since each partition has information about distance order to all objects, this information can be used to identify the true region from various types of spatial queries. Hence, one structure of HSVD can be used to identify the true region of various types of spatial queries with different k values, and eliminate the need of objects verification step.

2.4 Limitations

2.4.1 Limitation of Existing Works in Spatial Queries

Based on related works in nearest neighbour, farthest neighbour, reverse nearest neighbour and reverse farthest neighbour query processing, the existing techniques have seven main limitations as follows:

- L1 : Relying on pruning, either with space pruning or index pruning, and post processing techniques to filter the candidates.
- L2 : Pruning with Voronoi diagram requires a verification step, since Voronoi diagram has limited distance information to a limited number of generator points.
- L3 : Region-based calculation method only apply in RNN query; however, the region must be recreated for each query issued.

To address these issues, our proposed method is a region-based method where no pruning or verification steps are needed to identify the query region. We use a HSVD that

provides distance order information to all generator points for each cell to determine the true region of a query.

2.4.2 Limitation of Existing Works in Region-based Approach

- L5 : Candidate regions need a further verification process to obtain the true objects.
- L6 : No Voronoi diagram that can be used to represent spatial queries where $k > 1$.
- L7 : Region pruning must be performed on each query processing.

In response to existing issues, we proposed a modified Voronoi diagram with k order where each cell has a distance order to all generator points, enabling the Voronoi cell to identify the objects to answer spatial query in any k without having to perform a verification step. Moreover, the use of the Voronoi diagram is not limited to only one query.

To the best of our knowledge, the study in this thesis is the first to use region-based approach to solve four types of queries: k-Nearest Neighbour, k-Farthest Neighbour, Reverse Nearest Neighbour, and Reverse Farthest Neighbour with additional polychromatic spatial query for each query type.

2.5 Chapter Summary

This chapter has provided common understanding and discussion about state-of-the-art approaches regarding various spatial queries and region-based approaches to solve spatial queries. Our research is focused on a region-based approach in spatial queries based on a Voronoi diagram, where the aim is to obtain a region as the answer for these queries. The main challenge is how to extract this region from a Voronoi diagram and how to modify a Voronoi diagram so that it provides distance order information to all generator points in each Voronoi cell.

The following chapters will discuss how to construct the highest order Voronoi diagram and how to employ this diagram to solve various types of spatial queries.

Chapter 3

Highest Order Voronoi Diagram

3.1 Overview

In this chapter, we will explain one of our contributions to answer Challenge 1 regarding the limitations of the existing Voronoi diagram structure in solving spatial queries with the region-based approach. We propose an extended variation of the Voronoi diagram constructed through naive method that we call highest order Voronoi diagram (*HSVD*). Section 3.3 will review the Voronoi diagram and construction categories, and will explain the main purpose of HSVD. The definitions and properties of a generalized Voronoi diagram will be discussed in section 3.4. The definition of HSVD will be explained in section 3.5, and the properties of HSVD will be given in section 3.6. Section 3.7 will describe the construction method for HSVD with a FLIP algorithm. We will also describe the dual tessellation of HSVD, called adjacency graphs and its properties in section 3.8. The discussion and summary of this chapter will be provided in sections 3.9 and 3.10 respectively.

3.2 Notations and Symbols

All notations that will be used in this thesis are listed in Table 3.1. In this thesis, we also introduce several customs operators to be used in ordered tuples and sets as well as the regions as listed in Table 3.2.

Notation	Definition
S	A fixed axis-parallel rectangle in Euclidean space $S \subseteq \mathbb{R}^2$
x	an arbitrary set of points in Euclidean space $x \cap S \neq \emptyset$
$P = \{p_1, p_2, ..p_m\}$	Facility Points in S $P \subseteq S$
m	The number of Facility Points $2 \leq m < \infty$
n	Order for Voronoi Diagram $1 \leq n \leq m$
$D^{(n)}$	Set of unordered pairs of distinct n elements of P
$D^{(n)}[i]$	An unordered pairs of distinct n elements $D^{(n)}[i] \subseteq P, \lvert D^{(n)}[i]\rvert \leq \lvert P\rvert, \lvert D^{(n)}[i]\rvert = n$
$T^{\langle n,i\rangle}$	Set of distinct ordered n-tuples (sequences) of $D[i]^{(n)}$
$T^{\langle n,i\rangle}[j]$	A distinct ordered of n-tuples
F	Facility Points P that don't belong to $D^{(n)}[i]$ $F = P - D^{(n)}[i]$, $\lvert F\rvert = m - n$
$Bis(p_i : p_j)$	Bisector line between p_i and p_j
$H(p_i, p_j)$	Half-space of p_i that is created by $Bis(p_i : p_j)$
$R(p_i)$	Voronoi cell where p_i as the generator point
$R(D^{(n)}[i])$	Voronoi cell order-k where pairs $D^{(n)}[i]$ as the generator point
$R(T^{\langle n,i\rangle}[j])$	Ordered Voronoi cell order-n where sequence $T^{\langle n,i\rangle}[j]$ as the generator point
$R(\langle d_i, d_j\rangle)$	Ordered Voronoi cell where d_i is the 1^{st}-nearest generator point and d_j as the 2^{nd}-nearest generator point
V^n	Voronoi Diagram order n created from P
Vo^n	Ordered Voronoi Diagram order n
Vh	Highest order Voronoi Diagram
$C(n, m)$	Combination of n facility points from m facility points
$P(n, m)$	Permutation of n facility points from m facility points

Table 3.1: List of Notations

Operator	Definition	Operator Type
$\sqcup$	Merger of regions	Region
$\sqcap$	Overlap area of regions	Region
$\sqsubseteq$	Subregion	Region
$\preceq$	Predecessor elements of sequences	sequences

Table 3.2: List of Operators

3.3 Overview of Voronoi Diagram

The Voronoi diagram is one of the most fundamental data structures in computational geometry and has been widely applied in a variety of fields, either inside or outside computer science, such as nearest neighbour queries, path planning, closest pairs problem,

city planning and clustering. Given a number of points in the plane, the Voronoi diagram divides the plane according to the nearest neighbour rule where each point has the region of the plane closest to it. Voronoi diagram arises in natural situation where human intuition is often guided by visual perception. Some reasons why the Voronoi diagram receive so much attention have been given by (Aurenhammer, 1991). The regions in a Voronoi diagram are called Voronoi cells. When a cell considers a point as the first nearest neighbour, this Voronoi diagram is called an order-1 Voronoi diagram; when a cell considers more than one point as k nearest neighbour, this diagram is called *Higher Order Voronoi diagram* (Krussel and Schaudt, 1994; Fort and Sellars, 2009; Wang et al., 2011). Authors (Okabe et al., 2009) divide Voronoi diagram construction into seven categories, which can be summarized as follows

1. **Naive Method**

   Naive method is the easiest to understand since it is a direct translation of the definition of Voronoi diagram (Rhynsburger, 1973). This method is not meant to generate the Voronoi diagram itself, but to generate many samples of Voronoi polygons (Crain, 1978; Quine and Watson, 1984), but does not include explicit information about the topological structure of the diagram.

2. **Walking Method**

   Walking method is a method of constructing a Voronoi diagram, whereby Voronoi vertices and Voronoi edges are constructed one by one in just the order in which a traveller walks along the edges of the diagram. This method was described by (Brassel and Reif, 1979) and (Cromley and Grogan, 1985). This method was revised using the divide-and-conquer method.

3. **Flip Method**

   In this method, an initial diagram is constructed first, and then it is modified step by step until it converges to the Voronoi diagram. Authors (Sibson, 1978) showed that, starting with any triangulation of convex hull, we can modify it into Delaunay triangulation by flipping the diagonals of convex quadrilaterals according to the local max-min angle criterion. This Flip method is slightly different from our FLIP method. Our FLIP method is meant to obtain the distance order sequence on each Voronoi polygons constructed with naive method.

4. **Incremental Method**

   This is a simple and powerful method, which build a Voronoi diagram beginning with two or three generator points and modifies it by adding generators one by one. It was introduced by (Green and Sibson, 1978). In the worst case scenario, the total time complexity is $O(n^2)$; however, it can be decreased to $O(n)$ by using buckets and quartenary tree data structures (Iri et al., 1984). Randomization techniques are also useful for decreasing the average time complexity (De Berg et al., 1995). A more robust version of this method was proposed by (Sugihara and Iri, 1989, 1992).

5. **Divide-and-Conquer Method**

   In this method, the set of generator points is recursively divided into smaller subsets and the Voronoi diagrams for those subsets of generators are merged into the final diagram. This was firstly proposed by (Shamos and Hoey, 1975b) and then written in a simpler form of Delaunay triangulation terminology by (Lee and R. L. Drysdale, 1981). The robust version of this method was proposed by (Oishi and Sugihara, 1995)

6. **Plane Sweep**

   The sixth method is the plane sweep method (Fortune, 1987) where a vertical line, called a sweep line, is moved over the plane from left to right, and the Voronoi diagram is constructed along this line.

7. **Lift-up Method**

   This method utilizes the relationship between a two-dimensional Voronoi diagram and a three-dimensional convex hull. Firstly, the generators in the plane are transformed to certain points in three-dimensional space, then their convex hull is generated, and finally the convex hull is inversely transformed to the original plane to obtain the Delaunay triangulation. Authors (Brown, 1979; Aurenhammer and Edelsbrunner, 1984) used transformation from the plane to sphere, and (O'Rourke et al., 1986; Edelsbrunner and Seidel, 1986) used a transformation from the plane to a paraboloid of revolution. This method can be extended to any dimension, and also can be used to construct a higher order Voronoi diagram (Agarwal et al., 1994; Dwyer, 1991).

In this thesis, the first aim is to be able to construct a structure consists of smaller regions that have distance information to all available points on the map. This structure can be achieved through Naive method in Voronoi diagram construction. However, naive method itself is lack of explicit information about the distance information in the diagram. Therefore, in this thesis we introduce **Fast Labelling and Interchange Position (FLIP)** method to obtain distance information information to all generator points on the map, and use this information for further possible application.

3.4 Generalization of Voronoi Diagram

3.4.1 Definition of Voronoi Diagram

Let S be a fixed axis-parallel rectangle in Euclidean space $S \subseteq \mathbb{R}^2$, a set of facility points $P = \{p_1, p_2, p_3, .., p_m\} \subseteq S$ and $m =\mid P \mid$. Voronoi diagram can be defined as a method in dividing space into a number of regions where each region will consider a facility point p_i as the nearest point. The nearest point is called the *generator point* (Okabe et al., 2009). The regions in the Voronoi diagram are called $Voronoi\ cells$.

Definition 3.4.1. The **region for a generator point** p_i or a **Voronoi cell** can be defined as an intersection of all halfspace of p_i with other generator points $H(p_i, p_j)$. Since there are no overlap for any cells, $R(p_i) \sqcap R(p_j) = \emptyset$

$$R(p_i) = \sqcap_{j=1}^{m[\backslash i]} H(p_i, p_j) \mid i \neq j, (i, j) \leq m \tag{3.4.1}$$

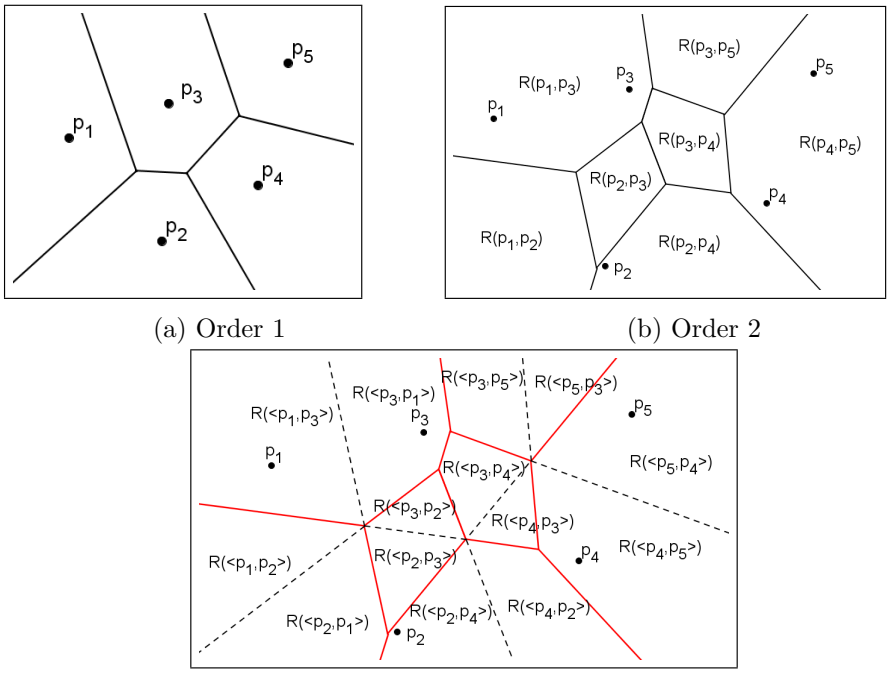
Let x be an arbitrary point in S and $\{x\} \cap S \neq \emptyset$, the Euclidean distance between x and p_i is given by $d(x, p_i)$. If p_i is the nearest generator point from x, then $d(x, p_i) \leq d(x, p_j)$ where $i \neq j$, $1 \leq i, j \leq m$. Therefore, we can rewrite Voronoi cell definition as follow:

Definition 3.4.2. The **Voronoi cell** can also be defined as the region where any point x located in a Voronoi cell $R(p_i)$ will consider p_i as the nearest generator point.

$$R(p_i) = \{x \mid d(x, p_i) \leq d(x, pj), i \neq j, 1 \leq i, j \leq m\} \tag{3.4.2}$$

Since each facility point has its own Voronoi cell, the union of all Voronoi cells will construct the Voronoi diagram.

Definition 3.4.3. Voronoi cell $R(p_i)$ is an **adjacent cell** of $R(p_j)$ if these cells share the same edge.



(a) Order 1 (b) Order 2

(c) Ordered Order 2

Figure 3.1: Voronoi Diagram from 5 points

Figure 3.1a is an example of a Voronoi diagram from five facility points. A Voronoi cell $R(p_1)$ is an adjacent cell of Voronoi cells $R(p_3)$ and $R(p_2)$ since these cells share the same region edge.

Definition 3.4.4. Voronoi diagram V can be defined as the union of all Voronoi cells.

$$V = \bigsqcup_{i=1}^{m} R(p_i) \tag{3.4.3}$$

3.4.2 Definitions of Higher Order Voronoi Diagram

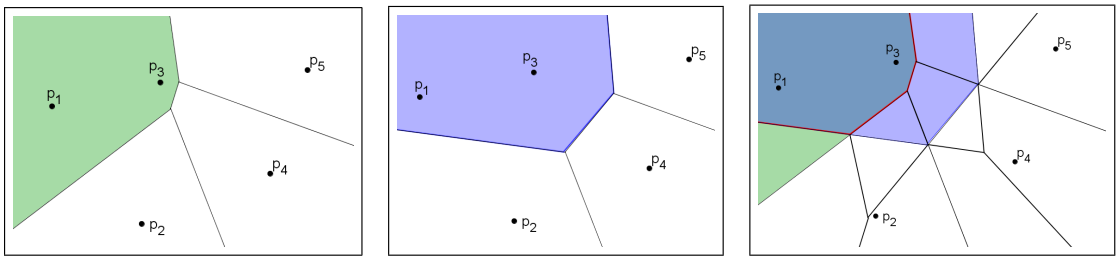
In a higher order Voronoi diagram, the number of generator points for one cell depends on the order of the Voronoi diagram itself. An order-n Voronoi diagram is another form of Voronoi diagram where each cell has n generator point; hence, each Voronoi cell has n-nearest generator points. An order-n Voronoi diagram is also called *Higher Order Voronoi diagram (HOVD)* (Agarwal et al., 1994).

Definition 3.4.5. Let $D^{(n)}$ is a set of unordered pairs of distinct n elements of P. $D^{(n)}[i] = \{d_{i1}, d_{i2}, ..., d_{in}\}$ is an unordered pairs from $D^{(n)}$ where $D^{(n)}[i] \subseteq P$. The **maximum number of unordered pairs of distinct n elements** is a combination of n facility points from m facility points.

$$C(n,m) = \frac{m!}{n!(m-n)!}$$

Definition 3.4.6. Let $F = \{f_1, f_2, ..., f_j\}$ is all Ps that do not belong to $D^{(n)}$ where $F = P - D^{(n)}, |F| = m - n$. **Region in Order-$n$ HOVD** $R(D^{(n)}[i])$ can be considered as a Voronoi cell where this cell has n generator points.

$$R(D^{(n)}[i]) = (\sqcap_{j=1}^{j=(m-n)} H(d_{i1}, f_j)) \sqcap .... \sqcap (\sqcap_{j=1}^{j=(m-n)} H(d_n, f_j)) \tag{3.4.4}$$



(a) $R(p_1 \backslash p_3)$ (b) $R(p_3 \backslash p_1)$ (c) $R(p_1 \backslash p_3) \sqcap R(p_3 \backslash p_1)$

Figure 3.2: $R(D^{(2)}\{p_1, p_3\})$

Figure 3.2 explains the construction of an order-2 Voronoi cell from five facility points $P = \{p_1, p_2, p_3, p_4, p_5\}$. Assume that we have a Voronoi cell where p_1, p_3 as the nearest facility points $D^{(2)}[i] = \{p_1, p_3\}$. All points that do not include in $D^{(2)}[i]$ is $F = \{p_2, p_4, p_5\}$. Figure 3.2a is the region of p_1 that ignores p_3 since p_3 will share the region with p_1. $R(p_1 \backslash p_3) = \sqcap_{j=1}^{j=3} H(p_1, f_j)$. Figure 3.2b is the region of p_3 that excludes p_1. $R(p_3 \backslash p_1) = \sqcap_{j=1}^{j=3} H(p_1, f_j)$. Voronoi cell of p_1, p_3 in 3.2c can be written as

$$R(D^{(2)}\{p_1, p_3\}) = R(p_1 \backslash p_3) \sqcap R(p_1 \backslash p_3)$$

The Euclidean distance from x to all points in set $D^{(n)}[i]$ is given by $d(x, d_{il}), 1 \leq i \leq C(n,m), n < m, 1 \leq l \leq n$. Points d_{il} in set $D^{(n)}[i]$ are the n-nearest points from x compared to other points in set F, so $d(x, d_{il}) \leq d(x - f_j)$, $1 \leq i \leq C(n,m)$, $1 \leq j \leq (m-n), n < m, 1 \leq l \leq n$. From this definition, the definition of a higher order Voronoi cell can be defined as follows:

Definition 3.4.7. The **Voronoi cell in a Higher Order Voronoi diagram** is the area where any point x located in a Voronoi cell $R(D^{(n)}[i])$ will consider points in set $D^{(n)}[i]$ as the n-nearest generator points.

$$R(D^{(n)}[i]) = \{x|\ d(x,d_{il}) \leq d(x,f_j),\ 1 \leq i \leq C(n,m), \\ 1 \leq j \leq (m-n), n < m, 1 \leq l \leq n\} \tag{3.4.5}$$

Figure 3.1b shows an order-2 Voronoi diagram from five facility points. Let $P = \{p_1, p_2, p_3, p_4, p_5\}$ and $n = 2$. $D^{(2)}$ can be defined as a set of unordered pairs of two distinct elements of P

$$D^{(2)} = \Big\{\{p_1,p_2\},\{p_1,p_3\},\{p_1,p_4\},\{p_1,p_5\},...,\{p_4,p_5\}\Big\}$$

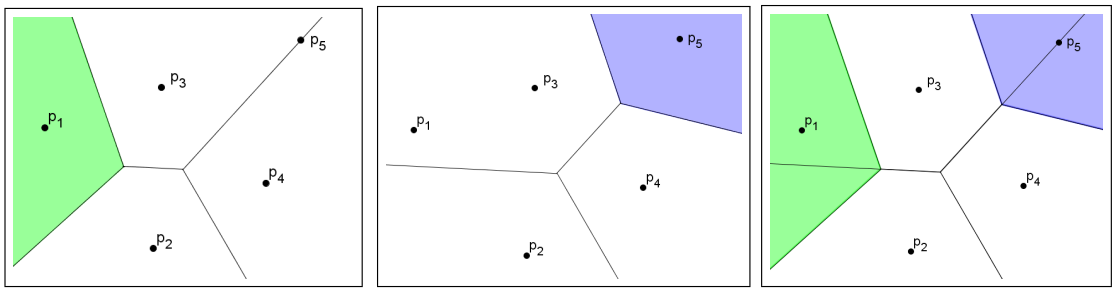
We can obtain unordered pairs of distinct n elements of P with the related F as can be seen in Table 3.3.

$D^{(2)}[i]$	F
$\{p_1,p_2\}$	$\{p_3,p_4,p_5\}$
$\{p_1,p_3\}$	$\{p_2,p_4,p_5\}$
$\{p_1,p_4\}$	$\{p_2,p_3,p_5\}$
$\{p_1,p_5\}$	$\{p_2,p_3,p_4\}$
$\{p_2,p_3\}$	$\{p_1,p_4,p_5\}$
$\{p_2,p_4\}$	$\{p_1,p_3,p_5\}$
$\{p_2,p_5\}$	$\{p_1,p_3,p_4\}$
$\{p_3,p_4\}$	$\{p_1,p_2,p_5\}$
$\{p_3,p_5\}$	$\{p_1,p_2,p_4\}$
$\{p_4,p_5\}$	$\{p_1,p_2,p_3\}$

Table 3.3: Unordered pairs where $k = 2$

The first pairs $D^{(2)}[1] = \{p_1,p_2\}$, and $F = \{p_3,p_4,p_5\}$. According to definition 3.4.7, A Voronoi cell for this unordered pair $R(D^{(2)}[1])$ is as the region where any point x in this cell will consider points p_1, p_2 as the nearest facility points, and will not consider any points in F as part of 2-nearest facility point.

The number of possible pairs can be obtained from $C(2,5) = \frac{5!}{2!(5-2)!}$ which are 10 set combinations of unordered pairs of two distinct elements. However, not all combinations constitute a Voronoi cell as described in *Definition 3.4.6.* A Voronoi cell can be constructed if the region of each members in unordered pairs have overlap region with other members. Figure 3.3 shows Voronoi cell from p_1, p_5 that never exist, since region $R(p_1 \backslash p_5)$ and region $R(p_5 \backslash p_1)$ do not have any overlapping region. $R(p_1 \backslash p_5) \sqcap R(p_5 \backslash p_1) = \emptyset$.



(a) $R(p_1 \backslash p_5)$ (b) $R(p_5 \backslash p_1)$ (c) $R(p_1 \backslash p_5) \sqcap R(p_5 \backslash p_1)$

Figure 3.3: Non existed Voronoi cell from p_1, p_5

From *Definition 3.4.4,* a Voronoi diagram is the union of all Voronoi cells. Since the number of Voronoi cells depends on the number of possible combinations, a higher order Voronoi diagram can be redefined as follows:

Definition 3.4.8. Higher order Voronoi diagram is the union of all combinations of order-n Voronoi cell.

$$V^n = \bigsqcup_{i=1}^{C(n,m)} R(D^{(n)}[i]) \tag{3.4.6}$$

3.4.3 Definitions of Ordered Higher Order Voronoi Diagram

The other variation of *HOVD* with ordered generator points for each cell is the *ordered HOVD* or *ordered order-n Voronoi diagram* (Okabe et al., 2009; Wang et al., 2011). In this diagram, each cell has a set of generator points in a certain order. Figure 3.1 shows the different structure of a Voronoi diagram from five facility points, In Figure 3.1a, each cell considers only one facility point as the nearest generator point, in Figure 3.1b, each cell has two facility points as the nearest generator points without considering the sequence. In Figure 3.1c, each cell has two facility points as the nearest generator points, and the distance order is represented by the label attached to each region. For example $R(\langle p_1, p_3 \rangle)$ means the region that considers p_1 as the first nearest and p_3 as the second nearest. In this thesis, the distance order sequence is also called 'sequence'.

Definition 3.4.9. Let $D^{(n)}$ is a set of unordered pairs of distinct n elements of P, $T^{\langle n,i \rangle}$ is a set of distinct ordered(sequence) n-tuples of $D^{(n)}[i]$. **The maximum number of possible n-tuples sequences that can be achieved from m facility points** can be defined as n-permutation of m.

$$P(n,m) = \frac{m!}{(m-n)!}$$

For each $D^{(n)}[i]$, the number of possible ordered n-tuples can be defined as $n!$.

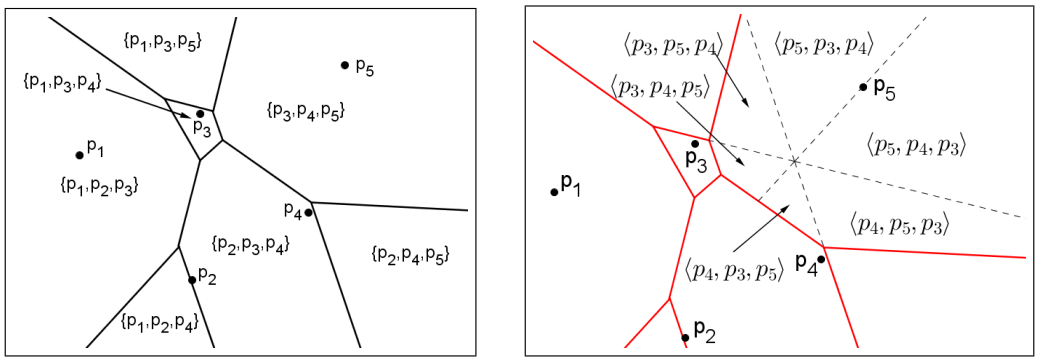
Definition 3.4.10. Let $D^{(n)}$ is a set of unordered pairs of distinct n elements of P, $T^{\langle n,i\rangle}$ is a set of distinct ordered n-tuples of $D^{(n)}[i]$. **Voronoi cell** $R(T^{\langle n,i\rangle}[y])$ is constructed from all intersections of halfspace in a certain order, and located inside $R(D^{(n)}[i])$. This Voronoi cell can be defined as

$$R(T^{\langle n,i\rangle}[y]) = \Big(H(t_{y1},t_{y2}) \sqcap ... \sqcap H(t_{y1},t_{yn}) \sqcap ... \sqcap H(t_{y(n-1)},t_{yn})\Big) \\ \sqcap R(D^{(n)})[i], 1 \leq y \leq n!, 1 \leq i \leq n, 1 \leq n < m \tag{3.4.7}$$

Let x be an arbitrary point in S. The Euclidean distance from x to all points in sequence $T^{\langle n,i\rangle}[y]$ is ordered by $d(x,t_{y1}) \leq d(x,t_{y2}) \leq ... \leq d(x,t_{yn}), 1 \leq n < m, 1 \leq i \leq C(n,m), 1 \leq y \leq n!$.

Definition 3.4.11. Let $F = \{f_1, f_2, ..., f_j\}$ is all Ps that do not belong to $D^{(n)}$ where $F = P - D^{(n)}$. $D^{(n)}[i]$ is an unordered pair of distinct n elements of P, and $T^{\langle n,i\rangle}$ is a set of distinct ordered n-tuples of $D^{(n)}[i]$. $T^{\langle n,i\rangle}[y]$ is an ordered distinct n-tuples. **The ordered order-n Voronoi cell** $R(T^{\langle n,i\rangle}[y])$ can be defined as the region where point x in $R(T^{\langle n,i\rangle}[y])$ considers t_{yz} as its z^{th}-nearest generator point.

$$R(T^{\langle n,i\rangle}[y]) = \Big\{x | \{d(x,t_{yz}) \leq d(x,f_j)\} \wedge \{d(x,t_{y1}) \leq d(x,t_{y2}) \leq ... \leq d(x,t_{yk})\}, \\ 1 \leq n < m, 1 \leq z \leq n, 1 \leq j \leq (m-n), 1 \leq y \leq n!, 1 \leq i \leq C(n,m)\Big\} \tag{3.4.8}$$



(a) Order 3 (b) Ordered order-3

Figure 3.4: Ordered Order-3 Voronoi cell

Figure 3.4a shows an order-3 Voronoi diagram from five facility points. To simplify, we will focus only on pairs $D^{(3)}[i] = \{p_3, p_4, p_5\}$ and $F = \{p_1, p_2\}$. The set of distinct ordered 3-tuples of $D^{(3)}[i]$ can be listed as.

$$T^{\langle 3,i\rangle} = \Big\{\langle p_3, p_4, p_5\rangle, \langle p_3, p_5, p_4\rangle, \langle p_4, p_3, p_5\rangle, \\ \langle p_4, p_5, p_3\rangle, \langle p_5, p_4, p_3\rangle, \langle p_5, p_3, p_4\rangle\Big\}$$

From the set above, we can have $T^{\langle 3,i\rangle}[1] = \langle p_3, p_4, p_5\rangle$. Voronoi cell of $R(T^{\langle 3,i\rangle}[1])$ can be defined as the region where any point x in this region will consider point p_3 as the first nearest generator point and point p_4 as the second nearest generator point and p_5 as the third nearest generator point, as shown in figure 3.4b.

$$R(T^{\langle 3,i\rangle}[1]) = \Big(H(p_3, p_4) \sqcap H(p_3, p_5) \sqcap H(p_4, p_5)\Big) \sqcap R(\{p_3, p_4, p_5\})$$

The number of possible ordered 3-tuples that can be obtained from $D^{(3)}[i]$ is $P(3,3) = \frac{3!}{(3-3)!}$ which is equal to 3!, therefore there are six possible sequences from $D^{(3)}[i]$. However, not all ordered tuples can be defined as an ordered order-3 Voronoi diagram. As in *Definition 3.4.10*, a Voronoi cell can be created if there exists an overlapping region from all members in a certain order. Hence, if there is no overlapping region from all members, the ordered order-n Voronoi cell cannot be created.

From the previous definition, we can now have an ordered order-n Voronoi diagram as the union of all ordered order-n Voronoi cell.

Definition 3.4.12. Ordered order-n Voronoi diagram is a HOVD where the generator points in each cell are sorted in a certain order. An ordered order-n Voronoi diagram can be defined as the union of all ordered order-n Voronoi cell.

$$Vo^n = \bigsqcup_{i=1}^{C(n,m)} \Big(\bigsqcup_{y=1}^{n!} R(T^{\langle n,i\rangle}[y])\Big) \tag{3.4.9}$$

3.4.4 Properties of Voronoi Diagram

Based on the definition of a Voronoi diagram provided in the previous section, we will now discuss some of their geometric properties (Okabe et al., 2009) that can be used to solve spatial queries. These properties are derived from Voronoi diagram, HOVD and ordered HOVD.

Property 3.4.1 (Overlapping property)**.** *There are no overlapping regions in Voronoi diagram.*

A Voronoi diagram is constructed based on perpendicular bisector line. This line is constructed from one pair of points, where the distance from one point to the bisector line is equal with the distance from another point to the bisector line.

Property 3.4.2 (Nearest points property)**.** *The nearest facility points from p_i create Voronoi cell edges $R(P_i)$.*

From this property, it is clear that a Voronoi cell of p_i is constructed by nearest surrounding facility points, while all other facility points will not be considered as the nearest.

Property 3.4.3 (Share edges property)**.** *Point p_i is near point p_j if $R(p_i)$ and $R(p_j)$ share the same Voronoi edge.*

Figure 3.4a shows a Voronoi diagram from five points. As we can see, the region $R(p_1)$ is constructed from p_2 and p_1 because these are the nearest points according to property 3.4.2. From this figure, we can also see that p_1 also creates the edge for $R(p_2)$ and $R(p_3)$, hence p_1, p_2 share the same Voronoi edge as well as p_1, p_3.

Property 3.4.4 (Inside property)**.** *Let x be the arbitrary point in S. Point x is located in region $R(p_i)$ if $d(x, p_i) < d(x, p_j), i \neq j, 1 \leq i, j \leq m$.*

This property matches *Definition 3.4.2* where if a point x is located in a Voronoi cell, this point will consider the generator point p_i as the nearest facility point compared to the other facility points.

Property 3.4.5 (No region property)**.** *In a HOVD, region $R(D^{(n)}[i])$ might be empty if $R(d_{ij}) \sqcap R(d_{il}) = \emptyset$, $\{d_{ij}, d_{il}\} \subset D^{(n)}[i]$*

Definition 3.4.6 describes how a Voronoi cell in HOVD is created. Since $R(D^{(n)}[i])$ must include all intersections of $R(d_{ij})$, a Voronoi cell $R(D^{(n)}[i])$ cannot be created if the regions of all members in $D^{(n)}[i]$ do not intersect with all other members.

From this property, we can also have the next property for the k nearest point:

Property 3.4.6 (Nearest points property)**.** *Let x be an arbitrary point in Euclidean space and $R(D^{(n)}[i])$ is a Voronoi cell of Order-n Voronoi from set $D^{(n)}$. Point x in $R(D^{(n)}[i])$ will consider $D^{(n)}[i]$ as n-nearest generator points.*

Property 3.4.7 (Maximum n property)**.** *Maximum n in HOVD is $(m-1)$, which can be stated as $n < m$.*

The reason for this property is straightforward. From property 3.4.6, point x in $R(D^{(n)}[i])$ will consider $D^{(n)}[i]$ as n-nearest generator point. If $n = m$, then $D^{(n)}[i] = P$. Hence, an order-n Voronoi cell will be an order-m Voronoi cell, and $R(P^{(m)})$ is the region where all points in P are m-nearest generator points, which is the Space S itself, so $R(P^{(m)}) = S$. Hence, n cannot be greater than m for HOVD.

Property 3.4.8 (Subregion property)**.** *An ordered order-n Voronoi cell is subset of order-n Voronoi cell.*

$$R(T^{\langle n,i\rangle}[y]) \sqsubseteq R(D^{(n)}[i]) \tag{3.4.10}$$

This property can be simply explained from definition 3.4.10, where $T^{\langle n,i\rangle}$ is a set of distinct ordered n-tuples of $D^{(n)}[i]$, hence ordered $T^{\langle n,i\rangle}[y]$ is also part of pairs in $D^{(n)}[i]$. Due to this circumstance, $R(T^{\langle n,i\rangle}[y])$ is also a sub-region of $R(D^{(n)}[i])$.

From this property, we can also obtain the next property.

Property 3.4.9 (Super region property)**.** *The union of an ordered order-n Voronoi cell with the same members of generator points will construct a Voronoi cell of Order-n Voronoi diagram.*

$$R(D^{(n)}[i]) = \bigsqcup_{y=1}^{n!} R(T^{\langle n,i\rangle}[y]), 1 \leq i \leq C(n,m), 1 \leq n < m \tag{3.4.11}$$

In an order-n Voronoi cell, the ordered list of generator points is not necessary as long as these points are the n-nearest generator points. However, in an ordered order-n Voronoi diagram, the 'ordered' is the main thing that must be considered.

In Figure 3.1c, the red line shows an order-2 Voronoi diagram and the dashed line shows an ordered order-2 Voronoi diagram. $R(p_1, p_3)$ in Figure 3.1b shows a Voronoi cell where p_1, p_3 are 2-nearest generator points in this cell. In Figure 3.1c, this cell consists two ordered subregions $R(\langle p_1, p_3\rangle)$ and $R(\langle p_3, p_1\rangle)$, where in $R(\langle p_1, p_3\rangle)$ point p_1 is the first nearest generator point and p_3 is the second nearest generator point.

Property 3.4.10 (Sequence property)**.** *For x in $R(T^{\langle n,i\rangle}[y])$ where $T^{\langle n,i\rangle}[y] = \langle t_{y1}, t_{y2}, ..., t_{yn}\rangle$, the following rule applies: $n^{th}NN(x) = t_{yn}$, because $d(x, t_{y1}) \leq d(x, t_{y2}) \leq ... \leq d(x, t_{yn})$*

Based on this property, we can also derive the next property based on the ordering sequence.

Property 3.4.11 (Prefix property)**.** *Let* $T^{\langle n,i\rangle}[y] = \langle t_{y1}, t_{y2}, ..., t_{yn}\rangle$ *is a sequence with* n *members and* $T^{\langle (n-l),i\rangle}[z] = \langle t_{z1}, t_{z1}, ..., t_{z(n-l)}\rangle$. $T^{\langle (n-l),i\rangle}[z]$ *is prefix of sequence* $T^{\langle n,i\rangle}[y]$ *if all tuples in* $T^{\langle (n-l),i\rangle}[z]$ *exist in* $T^{\langle n,i\rangle}[y]$ *in the same order.*

$$T^{\langle (n-l),i\rangle}[z] \succeq T^{\langle n,i\rangle}[y], \quad if:$$
$$\{T^{\langle (n-l),i\rangle}[z] \cap T^{\langle n,i\rangle}[y] = T^{\langle (n-l),i\rangle}[z]\},$$
$$\{(t_{y1} = t_{z1}) \wedge (t_{y2} = t_{z2}) \wedge ...(t_{y(n-l)} = t_{z(n-l)}), 1 \leq y \leq n!, 1 \leq z \leq (n-l)!, l < n\} \tag{3.4.12}$$

This property shows that a sequence $T^{\langle (n-l),i\rangle[z]}$ can become a prefix of $T^{\langle n,i\rangle[y]}$ if $T^{\langle n,i\rangle[y]}$ has more tuples than $T^{\langle (n-l),i\rangle[z]}$ and all tuples of $T^{\langle (n-l),i\rangle[z]}$ appear in the same sequence in $T^{\langle n,i\rangle[y]}$. From property 3.4.11 and property 3.4.9, we obtain the next property.

Property 3.4.12 (Ordered sub region property)**.** *The region of* $R(T^{\langle n,i\rangle}[y])$ *is an ordered subregion of* $R(T^{\langle (n-l),i\rangle}[z])$ *if* $T^{\langle (n-l),i\rangle}[z]$ *is prefix of* $T^{\langle n,i\rangle}[y]$.

$$R(T^{\langle n,i\rangle}[y]) \sqsubseteq R(T^{\langle (n-l),i\rangle}[z]) \quad if \quad T^{\langle (n-l),i\rangle}[z] \preceq T^{\langle n,i\rangle}[y] \tag{3.4.13}$$

Figure 3.5 explains the sub-ordered region property where $D = \{p_1, p_2, p_3, p_4, p_5\}$. Figure 3.5a shows a Voronoi cell $R(T^{\langle 1,i\rangle})$ where $T^{\langle 1,i\rangle}[y] = \langle p_4\rangle$ and p_4 is the first nearest generator point. Figure 3.5b shows a Voronoi cell $R(T^{\langle 2,i\rangle}) = \{< p_4, p_2 >, < p_4, p_3 >, < p_4, p_5 >\}$. As we can see from this figure, Voronoi cell $R(Do^2)$ is located inside Voronoi cell $R(Do^1)$. Figure 3.5c shows a Voronoi cell where $k = 3$. Assume $Do^3 =< p_4, p_2, p_3 >$ and $Do^2 =< p_4, p_2 >$, then $R(< p_4, p_2, p_3 >)$ is located inside of Voronoi cell $R(< p_4, p_2 >)$. Since $Do^3 \succeq Do^2$ then $R(Do^3) \sqsubset R(Do^2)$. Hence, $R(Do^n) \sqsubset R(Dr^t)$ if $Dr^t \succeq Do^n$.

From this property, we can conclude the next property about the the relationship between an ordered order-n Voronoi diagram and an ordered order-t Voronoi diagram, where $0 < t \leq n$.

Property 3.4.13 (Sub Voronoi property)**.** *An Ordered order-t Voronoi diagram can be constructed from an ordered order-n Voronoi diagram where* $0 < t \leq n$.

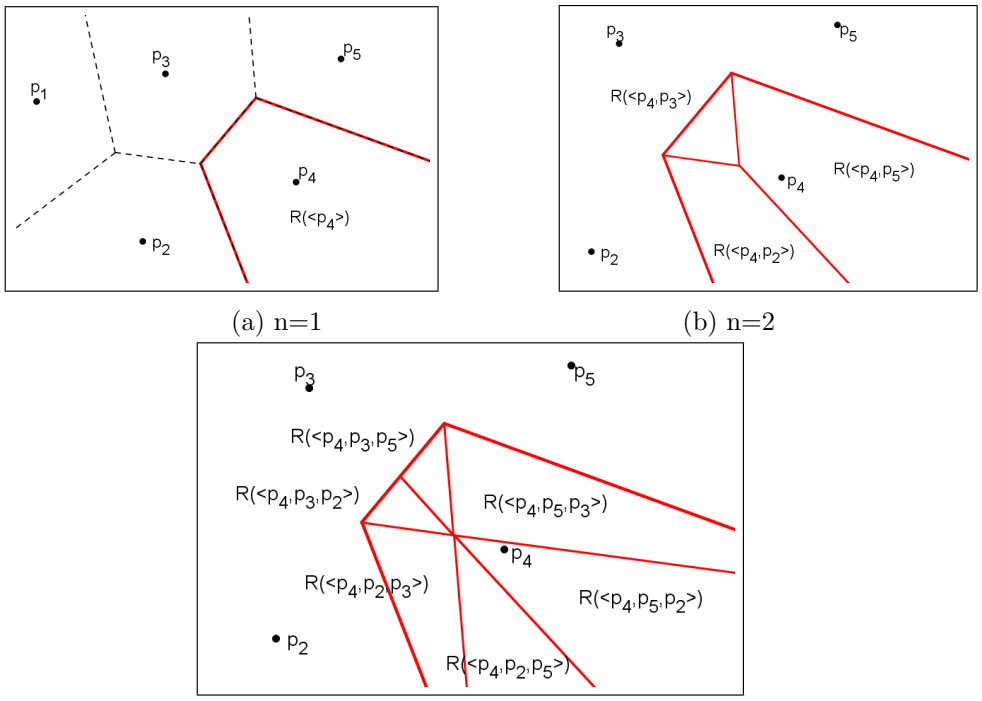


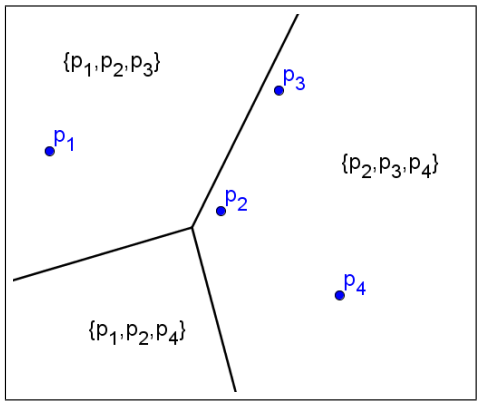
Figure 3.5: Region of sequence of p_4

Property 3.4.12 describes the sub-region of a Voronoi cell. Since each ordered order-n Voronoi cell is also a sub-region of an ordered order-t Voronoi cell where $0 < t \leq n$, the ordered order-n Voronoi diagram is a detailed version of an ordered order-t Voronoi diagram.

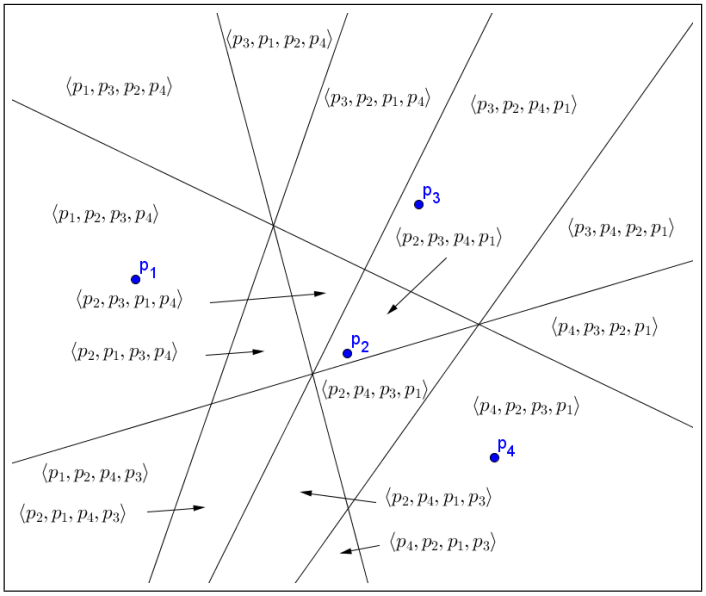
An ordered order-n Voronoi diagram can show the order of the generator points in every Voronoi cell. Hence, this diagram has the potential to be used to solve spatial problems. However, there are several problems that make this diagram unsuitable for some queries. The first one is the maximum number of order-n, where maximum $n = m - 1$. From Property 3.4.7, it is not possible to create an order-m Voronoi diagram because the order- m Voronoi cell of P is the whole space S. The second one, ordered order-n Voronoi diagram can be identified from an order-n Voronoi diagram. Hence, an ordered order-n Voronoi diagram cannot be achieved unless an order-n Voronoi diagram exists. Pre computing order-n Voronoi diagram is not an option due to computation cost and n cannot be predicted (Cheema et al., 2011). In the next section, we will discuss a HSVD to overcome these problems.

3.5 Definition of Highest Order Voronoi Diagram

A highest order Voronoi diagram (*HSVD*) is a new variation of a Voronoi diagram intended to extend *HOVD* capability. HSVD consists of Voronoi cells with distance information to all generator points. Let m be the number of facility points P, with each cell in *HSVD* having unique m ordered generator points. It means that no cells have the same m ordered generator points, and no generator points have the same order. In short, a highest order Voronoi diagram is an ordered order-m Voronoi diagram. Based on our knowledge, there are no variations of higher order Voronoi diagrams that have order-m.



(a) Order-3 VD



(b) HSVD

Figure 3.6: HOVD and HSVD

Figure 3.6 shows the differences between an order-3 Voronoi diagram and a highest order Voronoi diagram from four facility points. Figure 3.6a shows an order-3 Voronoi diagram from four facility points, where each cell has three generator points as the nearest

facility points while Figure 3.6b shows a highest order Voronoi diagram from the same facility points. Unlike *HOVD*, the generator points in each cell are sorted from the nearest to the farthest. Label $\langle p_1, p_4, p_2, p_3 \rangle$ means that any point x in this region will consider p_1 as the first nearest facility point, p_4 as the second nearest facility point, p_2 as the third nearest facility point and p_3 as the fourth facility point.

Definition 3.5.1. Let $n = m$, $D^{(n)}$ is a set of unordered pairs of distinct n elements of P. Since there is only one combination of n elements of P,$D^{(n)} = D^{(n)}[1]$, and $D^{(n)} = S$. $T^{\langle n,i \rangle}$ is a set of distinct ordered(sequence) n-tuples of $D^{(n)}[i]$, $i = 1$. **The maximum number of possible n-tuples sequences that can be achieved from m facility points** can be defined as n-permutation of m.

$$P(n, m) = \frac{m!}{(m-n)!}$$

since $n = m$

$$P(n, n) = \frac{n!}{(n-n)!} = n!$$

Definition 3.5.2. Let $D^{(n)}$ is a set of unordered pairs of distinct n elements of P, $T^{\langle n,1 \rangle}$ is a set of distinct ordered(sequence) n-tuples of $D^{(n)}[1]$. A **Voronoi cell** $R(T^{\langle n,i \rangle}[y])$ is constructed from all intersections of halfspace in a certain order, and are located inside $R(D^{(n)}[1])$. This Voronoi cell can be defined as

$$R(T^{\langle n,1 \rangle}[y]) = H(t_{y1}, t_{y2}) \sqcap ... \sqcap H(t_{y1}, t_{yn}) \sqcap ... \sqcap H(t_{y(n-1)}, t_{yn}), 1 \leq y \leq n!, n = m \quad (3.5.1)$$

Definition 3.5.3. In Euclidean distance, the **highest order Voronoi cell** $R(T^{\langle n,1 \rangle)}[y])$ can also be defined as the region where for any point x in $R(T^{\langle n,1 \rangle)}[y])$, this point considers t_{yz} as its z^{th}-nearest generator point.

$$R(T^{\langle n,1 \rangle}[y]) = \{x | d(x, t_{y1}) \leq d(x, t_{y2}) \leq ... \leq d(x, t_{yn}), n = m, 1 \leq y \leq n!\} \quad (3.5.2)$$

We will explain this definition with an example from Figure 3.6, where there are four facility points in the space S.

Figure 3.6b shows a highest order Voronoi diagram from 4 facility points. Let $n = 4$, $D^{(4)} = \Big\{\{p_1, p_2, p_3, p_4\}\Big\}$. There are no other combinations since $C(4, 4) = \frac{n!}{n!(n-n)!} = 1$.

Hence, an order-4 Voronoi diagram is the same as its Voronoi cell, which is the whole space S.

We can obtain 4! distinct ordered 4-tuples of $D^{(4)}[1]$ as follows

y	$T^{\langle 4,1\rangle}$
1	$\langle p_1, p_2, p_3, p_4\rangle$
2	$\langle p_1, p_2, p_4, p_3\rangle$
3	$\langle p_1, p_3, p_2, p_4\rangle$
...	...
$\{p_4, p_5\}$	$\langle p_4, p_3, p_2, p_1\rangle$

Table 3.4: Distinct ordered 4-tuples of $D^{(4)}[1]$

In a region $R(T^{\langle 4,1\rangle}[y])$, where $T^{\langle 4,1\rangle}[y] = \langle t_{y1}, t_{y2}, t_{y3}, t_{y4}\rangle$, a point x will consider t_{yj} as the j^{th} nearest generator point. In region $R(T^{\langle 4,1\rangle}[1])$, generator point p_1 is the first nearest point and p_4 is the 4^{th} nearest point.

Since a Voronoi diagram is the union of all Voronoi cells, a highest order Voronoi diagram can be defined as follows:

Definition 3.5.4. Highest order Voronoi diagram (HOVD) is the union of all possible highest order Voronoi cells.

$$Vh = \bigsqcup_{y=1}^{n!} R(T^{\langle n,1\rangle}[y]) \tag{3.5.3}$$

Similar to the other Voronoi diagram, a highest order Voronoi diagram is also constructed from the union of all available highest order Voronoi cells.

3.6 Properties of Highest Order Voronoi Diagram

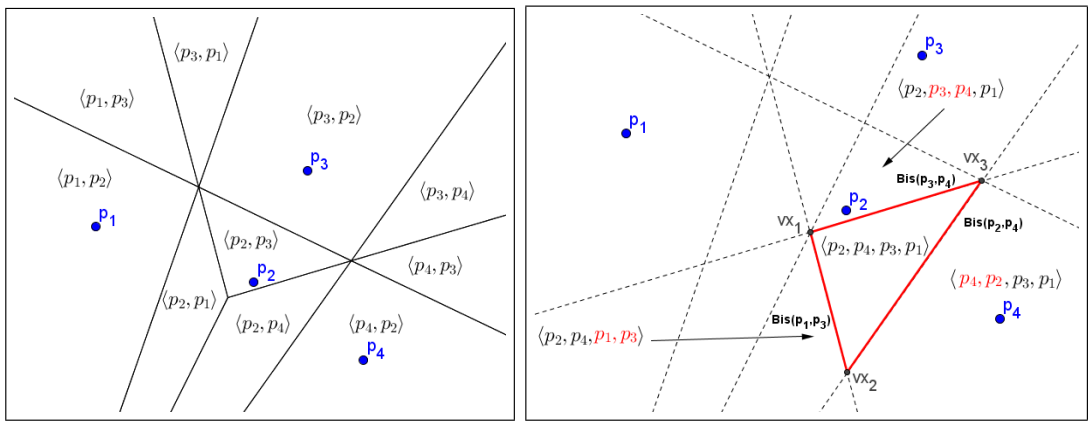
Since each cell in HSVD has distance order information to m generator points, a highest order Voronoi diagram is also an ordered order-m Voronoi diagram, hence all properties from HOVD will also apply on HSVD. Next, we present the properties that can only be applied in HSVD:

Property 3.6.1 (m number property)**.** *The number of elements in each tuple is m. Highest order Voronoi diagram can also be called ordered order-m Voronoi diagram.*

This property is explained in definition 3.5.2, where there are m-distinct tuples with a specific order in each region. In the other words, each region has a unique generator points sequence that indicates the nearest generator point to the farthest generator point.

Property 3.6.2 (Adjacent cells property)**.** *Each adjacent cell will always have one pair of different elements tuples.*

A Voronoi diagram is constructed by using a perpendicular bisector from all generator points. The intersection of two bisector lines will become the vertex of a Voronoi cell, and a Voronoi cell edge is a segment between two vertices in a perpendicular bisector. This bisector itself is made from p_i, p_j where $H(p_i, p_j)$ is the halfplane of p_i and $H(p_j, p_i)$ is the halfplane of p_j. Since a Voronoi cell is made up of segments of the bisector line, all adjacent cells will share the same segment/edge. Assume cell R_a has adjacent cell R_b in segment $S_{(a,b)}$, and $S_{(a,b)}$ is on the perpendicular bisector line $Bis(p_i, p_j)$. If in cell R_a $d(p_i, R_a) < d(p_i, R_b)$, then p_i is considered as the closer point in R_a rather than p_j; hence, the ordered generator point will be p_i, p_j. However, in R_b, p_j will be considered as the nearest point since R_a and R_b share the same segment that is part of bisector $Bis(p_i, p_j)$. Hence, the ordered generator point in R_b will be p_j, p_i.



(a) Ordered order-2 (b) Highest Order Voronoi cell

Figure 3.7: Ordered Voronoi cell from 4 facility points

Figure 3.7 shows the ordered tuple from an ordered order-2 Voronoi diagram and highest order Voronoi diagram. Figure 3.7b shows the different pairs from each Voronoi cell. As we can see from this figure, Voronoi cell $R(\langle p_2, p_4, p_3, p_1\rangle)$ has three adjacent cells which are:

1. Voronoi cell $R(\langle p_2, p_3, p_4, p_1\rangle)$ that shares an edge obtained from bisector line $Bis(p_3, p_4)$; hence, this adjacent Voronoi cells must swap the order of p_3 and p_4

2. $R(\langle p_2, p_4, p_1, p_3\rangle)$ that shares an edge obtained from bisector line $Bis(p_1, p_3)$; hence, this adjacent Voronoi cells must swap the order of p_1 and p_3

3. $R(\langle p_4, p_2, p_3, p_1\rangle)$ that shares an edge obtained from bisector line $Bis(p_2, p_4)$; hence, this adjacent Voronoi cells must swap the order of p_2 and p_4

From Figure 3.7a, we can see that there are no swapped pairs among adjacent Voronoi cells in an ordered order-2 Voronoi diagram; hence, this property can be applied only in a highest order Voronoi diagram.

Property 3.6.3 (All bisectors property)**.** *A highest order Voronoi diagram can be constructed by using all available perpendicular bisector lines.*

In a highest order Voronoi diagram, all possible sequences of ordered n-tuples will be created and each possible sequence will have its own cell. Voronoi cell is constructed from the intersections of perpendicular lines. Since all possible Voronoi cells will be constructed, all perpendicular bisector lines will be used.

3.7 Highest Order Voronoi Diagram Construction

This section will explain the method for highest order Voronoi diagram construction. Here we concentrate mainly on the most basic method used for constructing a highest order Voronoi diagram based on the definitions and properties that have been discussed in the previous section. First, we define the data structures that will be used in this method; second, we define the basic concept of construction; and last, we present the algorithm for constructing this Voronoi diagram. The HSVD construction is based on the naive method in order to generate Voronoi cells and this is followed by the FLIP method to add distance information on each Voronoi cell.

3.7.1 Computational Preliminaries

As has been discussed in the previous section, let S be the space in a 2-dimensional axis $S \subseteq \mathbb{R}^2$, $P = \{p_1, p_2, ..., p_m\}, P \in S$ be the set of facility points, $T^{\langle m,1\rangle}[y] = \langle t_{y1}, t_{y2}, ..., t_{ym}\rangle$ be

the ordered distinct m-tuples, $R(T^{\langle m,1\rangle}[y])$ be the ordered order-m Voronoi cell, and highest order Voronoi diagram will be written as $Vh = \left\{R(T^{\langle m,1\rangle}[1]), R(T^{\langle m,1\rangle}[2]), ..., R(T^{\langle m,1\rangle}[m!])\right\}$. From the previous section, the construction of a highest order Voronoi diagram is a procedure for generating Vh from Voronoi cells $R(T^{\langle m,1\rangle}[y])$ according to Definition 3.5.4. However, in this section, we will simplify the construction based on several properties that have been discussed previously.

3.7.2 Data Structure

A Voronoi diagram is constructed from the union of Voronoi cells where each cell is a convex polygon. Property 3.4.1 explains that each cell is unique and there are no overlapping cells in a Voronoi diagram. Property 3.4.3 explains that each cell has other adjacent cells that share the same edge. This means that this edge can be used to determine the neighbour of a particular cell. Since each cell is constructed from m facility points with a specific order and no cells have the same order, each cell can be identified by this order of m-tuples(sequence) as in property 3.4.6.

The edge of a Voronoi cell is a segment in a perpendicular bisector line, where this bisector line is constructed from two facility points p_i and p_j; hence, the edge of a Voronoi cell is basically constructed from facility points as well.

Since a Voronoi diagram is constructed from the union of all Voronoi cells, we have to determine a Voronoi cell that complies with previous explanations. We store a Voronoi cell as a representation of vectors (Okabe et al., 2009).

Structure	**Attributes**	**Description**
Voronoi Cell	sequence	Ordered m-tuples
	edges	List of segments
Segments	start point	Segment start point
	end point	Segment end point
	bisec line	Bisector line for segment
Bisector Line	line formula	Line equation of bisector
	p_1	Facility point 1
	p_2	Facility point 2
Facility Point	name	Facility point name
	location	Facility point coordinate

Table 3.5: Basic data structure for Voronoi cell

Table 3.5 shows the basic data structure that will be used to define a Voronoi cell and to construct Voronoi diagram. This data structure is mainly used for a highest order Voronoi diagram, but it can also be applied for a higher order Voronoi diagram. This diagram is built based on the definitions and properties of a Voronoi diagram, higher order Voronoi diagram and highest order Voronoi diagram. Since we cannot use a region as the basic structure for a Voronoi diagram, we will represent the cell as the set of edges, where the edges are represented as a vector (Okabe et al., 2009; Berg et al., 2008).

3.7.3 Construction Concept

In this section, we will describe how to construct a Voronoi diagram without having to identify each Voronoi cell one by one, since we will use all perpendicular bisectors.

Algorithm 1 is the main algorithm used to construct a highest order Voronoi diagram. At the first step, this algorithm will create all possible perpendicular bisectors that can be constructed. Assume that there are m facility points, since the bisector line is constructed from two facility points; the number of possible bisector lines can be defined as

$$C(2, m) = \frac{m!}{2!(m-2)!}$$

The next step is to find all possible intersection points; this is followed by identifying all available segments and using them to identify the regions that will be used for the Voronoi cells. Since a cell is stored as a vector representation, we also represent segment as vector as well.

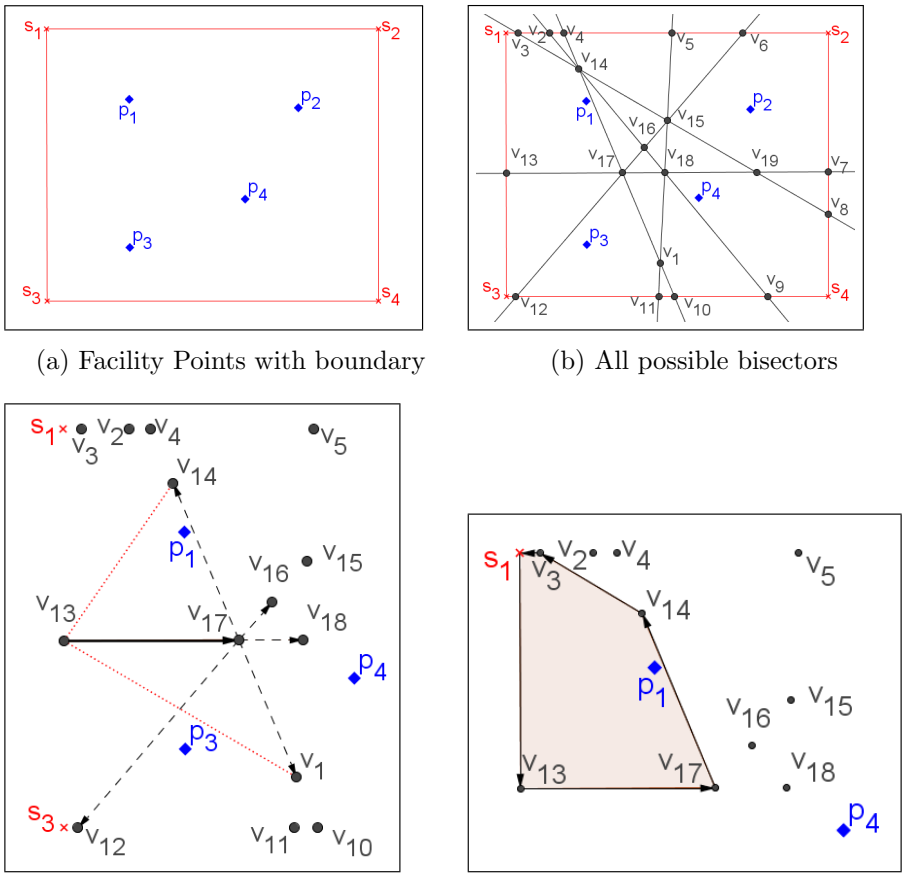
Figure 3.8 shows how to identify the region. Firstly, the boundary is set as shown in Figure 3.8a, and then all possible bisectors are created as in Figure 3.8b. The next step is to choose a segment as the starting edge, in this case $Seg_{(v13,v17)}$ and then determine the next segments. Figure 3.8c shows five possible segments, which are $Seg_{(v17,v14)}$, $Seg_{(v17,16)}$, $Seg_{(v17,v18)}$, $Seg_{(v17,v1)}$, $Seg_{(v17,12)}$. As we can see, $Seg_{(v17,v1)}$ and $Seg_{(v17,v16)}$ cannot be chosen because if we create a segment $Seg_{(v_{13},v_1)}$ or $Seg_{(v_{13},v_{16})}$, these segments will intersect with other segments, which are $Seg_{(v_{17},v_{14})}$ or $Seg_{(v_{13},v_{12})}$. Segment $Seg_{(v_{17},v_{18})}$ cannot be chosen because it is located on the same bisector as segment $Seg_{(v_{13},v_{17})}$. Hence, we can only choose segment $Seg_{(v_{17},v_{14})}$ or $Seg_{(v_{17},v_{12})}$ as the next segment. Assuming that we choose $Seg_{(v_{17},v_{14})}$ as the next segment, the next step is to choose subsequent segments from v_{14}. Previous rules need to be applied when choosing the next segments. Finally,

Algorithm 1: Highest Order VD Construction

Data: Set of points P, Limited area S
Result: Highest order Voronoi diagram Vh

```
   /* Create all possible bisector lines from P with naive method and
      put in the list                                                  */
 1 m ← count(P)
 2 for i=1 to m do
 3 |   for j=i to m do
 4 |   |   BisList ← getBisector(p_i, p_j)
 5 |   end
 6 end
   /* Edges of area S are also considered as bisector line            */
 7 BisList ← S
   /* Identify all bisector intersections (vertices) on each bisector
      line.                                                            */
 8 for i=1 to count(BisList) do
 9 |   for j=i to count(BisList) do
10 |   |   vertex_point ← getVertex(BisList[i], BisList[j])
11 |   |   BisList[i].addVertex(vertex_point)
12 |   |   BisList[j].addVertex(vertex_point)
13 |   end
14 end
   /* Identify all available segments.  A Segment is considered as 2
      consecutive vertices at the same bisector line and stored as 2-way
      vector                                                           */
15 for i=1 to count(BisList) do
16 |   SegList ← getSegment(BisList[i])
17 end
   /* Identify all available cells.                                    */
18 regCount ← 1
19 for i=1 to count(SegList) do
20 |   if !isUsed(SegList[i]) then
21 |   |   while !isRegion(Region[regCount]) do
22 |   |   |   currentSeg ← SegList[i]
23 |   |   |   Region[regCount].addSegment(currentSeg)
24 |   |   |   currentSeg.used ← true
25 |   |   |   currentSeg ← getNext(currentSeg)
26 |   |   end
27 |   end
28 |   regCount++
29 end
   /* Use FLIP to identify the sequence of each Voronoi cell           */
30 for i=1 to count(Region) do
31 |   FLIP(Region[i])
32 end
```

region R is defined as a circuit from $v_{13} \rightarrow v_{17} \rightarrow v_{14} \rightarrow v_3 \rightarrow s_1 \rightarrow v_{13}$ as shown in Figure 3.8d.



(a) Facility Points with boundary

(b) All possible bisectors

(c) Segment Tracing

(d) Recognized Region

Figure 3.8: Region Identification

The number of possible Voronoi cell in a highest order Voronoi diagram is $m!$ as mentioned in definition 3.5.1. After all regions have been identified, the next step is identify the ordered m-tuples or sequence for each Voronoi cell. This step can be done with the FLIP algorithm that will be discussed in the next section.

3.7.4 Fast Labeling and Interchange Position (FLIP) Algorithm

The FLIP method is used to identify the sequence in a highest order Voronoi cell. This method uses a Voronoi cell as the reference by identifying the sequence in this cell with kNN query, and moves to all adjacent cells. The FLIP method is based on property 3.6.2, where each adjacent Voronoi cell will definitely have a swapped-pair order tuples. Hence, if we can identify the sequence in a Voronoi cell, we will be able to identify all sequences of Voronoi cells in a highest order Voronoi diagram.

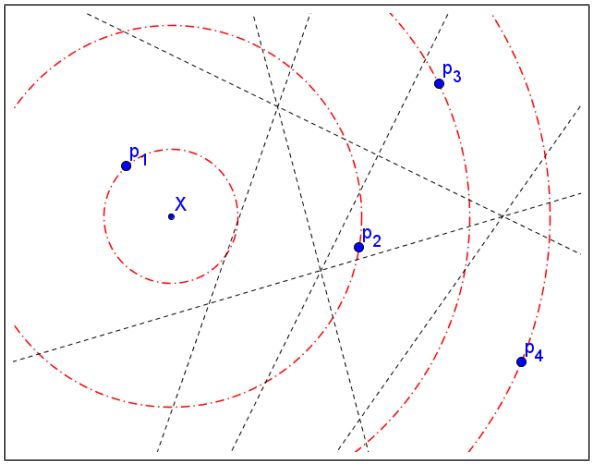
This algorithm 2 is simply the first sequence on an unlabeled cell that applies kNN query to it where k is the number of points $P - 1$ and the sequence is the result of this

Algorithm 2: FLIP algorithm

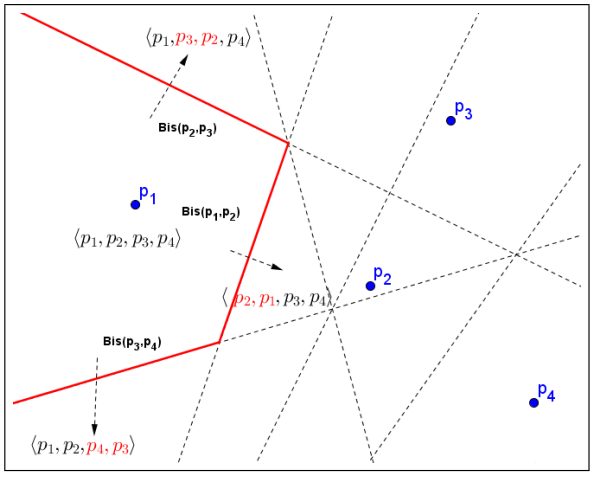
Data: Region R without sequence
Result: Region R with sequence

```
1  k ← count(P) − 1
2  if first(R) then
3  |  R.sequence ← kNN(x);
4  else
5  |  R′ ← getAdjacent(R);
6  |  while R′.sequence = NULL do
7  |  |  R′ ← getAdjacent(R);
8  |  end
9  |  R.sequence ← getSequence(R′, bisector(R, R′));
10 end
```

query. The first cell can be chosen from any cell. For the other cells, the sequence is based on its adjacent cell's sequence and the bisector lines as the shared edge between these cells. This algorithm is illustrated in Figure 3.9



(a) First unlabeled cell



(b) expansion

Figure 3.9: FLIP algorithm preview

Figure 3.9a shows when the first unlabeled cell is being processed. An arbitrary point x inside any region is used and kNN query is issued from this point. Since there are only four facility points, $4NN(x) = \langle p_1, p_2, p_3, p_4 \rangle$. The next cells are all adjacent cells to the current cell. Figure 3.9b shows three adjacent cells, and the next cells that will be processed are indicated by the arrow. The first adjacent cell will have $\langle p_1, p_3, p_2, p_4 \rangle$ as the sequence since they use bisector line $Bis(p_2, p_3)$ for the shared edge. This method applies to other adjacent cells as well. Hence, all adjacent cells have

$$\langle p_1, p_2, p_3, p_4 \rangle \xrightarrow{Bis(p_2,p_3)} \langle p_1, p_3, p_2, p_4 \rangle$$
$$\langle p_1, p_2, p_3, p_4 \rangle \xrightarrow{Bis(p_1,p_2)} \langle p_2, p_1, p_3, p_4 \rangle$$
$$\langle p_1, p_2, p_3, p_4 \rangle \xrightarrow{Bis(p_3,p_4)} \langle p_1, p_2, p_4, p_3 \rangle$$

This algorithm will be repeated until all cells have the sequence label, and highest order Voronoi diagram has been constructed. If there are m facility points, there will be at the most $m!$ Voronoi cells that need to be processed. However, from our experiment, the average number of Voronoi cells from a highest order Voronoi diagram is far below the maximum number.

3.8 Adjacency Graph of Highest Order Voronoi Diagram

A Voronoi diagram has dual tessellation, called Delaunay triangulation, which shows the neighbourhood generator points that construct the Voronoi diagram itself (Okabe et al., 2009). However, this tessellation can only be used in an ordinary Voronoi diagram or order-1 Voronoi diagram. Figure 3.10 shows a Voronoi diagram (red line) constructed from five generator points and its Delaunay triangulation (dashed). As can be seen in this figure, the Delaunay triangulation shows the cells adjacency graph. The nodes in Delaunay triangulation represent the generator points for a Voronoi diagram, and the edges represent the adjacency between cells in Voronoi diagram. For example, in this Delaunay triangulation, generator point p_1 is directly connected to p_2 and p_3, as we can see in a Voronoi diagram, Voronoi cell p_1 has cell p_2 and cell p_3 as its adjacent cells; hence, Delaunay triangulation can also be considered as cells adjacency graph of a Voronoi diagram. To the best of our knowledge, an only ordinary Voronoi diagram has a cells adjacency graph, and no other higher order Voronoi diagrams have cells' adjacency graph representations.

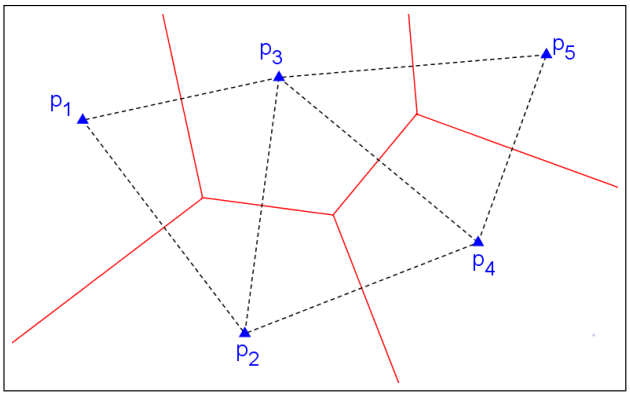


Figure 3.10: Ordinary Voronoi Diagram with Delaunay Triangulation

A highest order Voronoi diagram is a generalisation of a Voronoi diagram, where each cell has a distance order sequence to all generator points. The structure of a cells adjacency graph is useful to determine the neighbouring cells; therefore, in this section we present a cells adjacency graph for a highest order Voronoi diagram.

Since not all cells have a generator point in highest order Voronoi diagram, we have to generate another point that can represent a node in an adjacency graph. We use a **polygon centroid** to represent nodes in a cells adjacency graph of HSVD, because the HSVD cells are always in convex polygon, and the centroid of convex polygon is always located inside the polygon. The edges of a cells adjacency graph will show the adjacency between cells.

Definition 3.8.1. An adjacency graph of highest order Voronoi diagram (AG) is a closed graph that consists of nodes that represent the Voronoi cells of an HSVD, where the edges represent cells adjacency. A polygon centroid calculation is used to transform a cell into a node.

If the set of HSVD cells is represented as $C = \{c_1, c_2, ..., c_i\}$, the centroid that represents a cell as $ct(p_i)$, AG can be represented as $AG = \{ct(p_1), ct(p_2), ..., ct(p_i)\}$. The edge between $ct(p_1)$ and $ct(p_2)$ will be represented as $e(ct(p_1), ct(p_2))$.

Figure 3.11 shows HSVD from five generator points with a corresponding cells adjacency graph. In this graph, the nodes represent cells in HSVD. However, since these cells cannot be used directly in the graph, we use the cells' centroid to be represented as nodes in the graph, so the edges can show adjacency of cells.

In the next following subsection, we discuss the properties of the HSVD adjacency graph and the algorithm used to construct this graph.

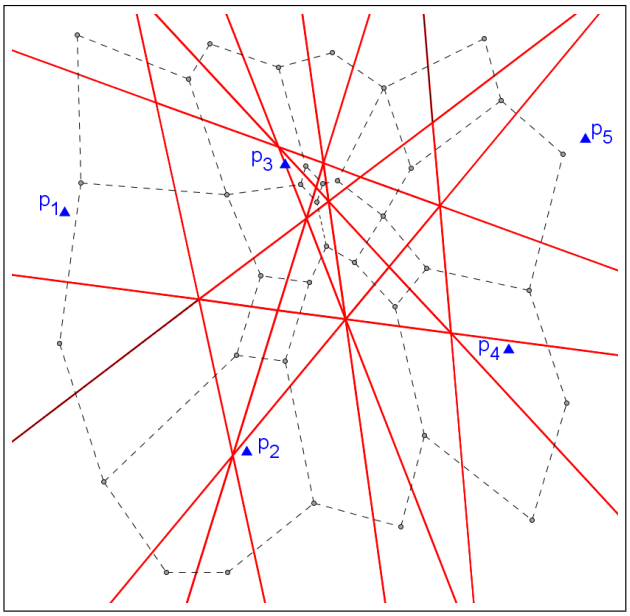


Figure 3.11: HSVD with Cells Adjacency Graph

3.8.1 Adjacency Graph Properties

In this subsection, we discuss the properties of adjacency graph from HSVD structure.

Property 3.8.1. *Adjacency graph is the dual graph of highest order Voronoi diagram*

Property 3.8.2. *The external edges in AG do not constitute the boundary of the convex hull of AG.*

Unlike Delaunay triangulation, the adjacency graph is created based only on the HSVD structure to show adjacency cells in the HSVD structure itself. Therefore, an adjacency graph does not follow the properties of Delaunay triangulation (Okabe et al., 2009). Unlike Delaunay triangulation and the Voronoi diagram where the generator points determine the edges of Voronoi cells and Delaunay triangulation edges, in an AG, a node is created from the centroid of a particular cell without considering the influence of other nodes. Therefore, the external edges in AG do not constitute the boundary of the convex hull.

Property 3.8.3. *An adjacency graph cannot be reverse to an HSVD, however an HSVD can be transformed into an AG.*

Since AG is constructed from cells of HSVD without considering the generator points and these cells are constructed from generator points, it is not possible to reconstruct the HSVD structure from AG without considering the generator points. Hence, AG cannot be used to reconstruct the HSVD structure.

Property 3.8.4. *Adjacency graph is a bidirectional graph*

The adjacency graph is used to trace the adjacency cells in HSVD structure, which means this graph could be plotted in any way. Therefore, the adjacency graph is constructed as a bidirectional graph.

3.8.2 Adjacency Graph Construction

An adjacency graph can be constructed if the highest order Voronoi diagram has been created. This graph will represent the adjacency of each cell in the HSVD structure. Since an AG is a direct representation of the HSVD structure and cannot be used without the HSVD structure itself, the adjacency graph is created as an additional property of HSVD structure and stored in HSVD structure as well.

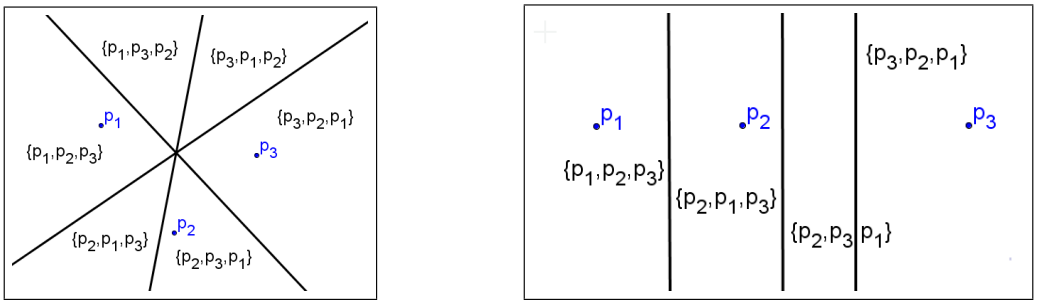
Algorithm 3: Adjacency Graph algorithm

```
   Data: Vh : HSVD without Adjacency Graph
   Result: Vh :HSVD with Adjacency Graphs
 1 for i = 1 to sizeOf(Vh) do
 2 |   if (checkCentroid(Vh[i]) = NULL) then
 3 |   |   Vh[i].centroid ← centroid(Vh[i]);
 4 |   end
 5 |   Adj ← getAdjacent(Vh[i]); /* get all adjacent cells from Vh[i]      */
 6 |   for j = 1 to sizeOf(Adj) do
 7 |   |   if (checkCentroid(Adj[j]) = NULL) then
 8 |   |   |   Vh[getIndex(Adj[j])].centroid ← centroid(Adj[j];
 9 |   |   end
10 |   |   if (checkEdge(edges(Vh[i], Vh[getIndex(Adj[j]) = NULL) then
11 |   |   |   Vh[i].adjEdges.add(edges(Vh[i], Vh[getIndex(Adj[j])].centroid);
12 |   |   end
13 |   end
14 end
```

Algorithm 3 shows the adjacency graph construction. This will involve a full read of HSVD cells, where each cell will be processed one by one. The first step is to check if the centroid has been created for a Voronoi cell (line 2). If it is empty, then the centroid will be created; otherwise, the adjacent cells will be searched (line 5). The next step is to create nodes to all adjacent cells. Each cell will create edges to all other adjacent nodes if this not has not yet been done (line 10). By the end of this algorithm, each cell will have edges to all adjacent cells.

3.9 Analysis

The complexity of the naive method used to construct the HSVD depends on the number of Voronoi cells that need to be constructed. For m generator points, the number of possible cells that can be constructed is equal to the possible combination of distance sequences of m generator points which is $m!$ as stated in Definition 3.5.1. However, the number of possible sequences is determined by the positions of the generator points themselves.



(a) 6 cells from 3 generator points (b) 4 Cells from 3 generator points

Figure 3.12: Number of Cells from 3 Generator points

Figure 3.12 shows all the possible Voronoi cells that can be constructed from three generator points. Figure 3.12a shows six cells, which shows 3!, and Figure 3.12b shows only four possible cells since all generator points are located co-linearly. Hence, $m!$ regions from m generator points are never likely to be achieved.

From Figure 3.12, the regions are constructed from bisector lines, where the bisector lines are constructed from two generator points. The number of regions created from l lines in the space has been defined by (Engel, 1998), where the maximum number of regions p_l created from l lines in 2-dimensional space can be defined as

$$r_l = C(l,0) + C(l,1) + C(l,2) \tag{3.9.1}$$

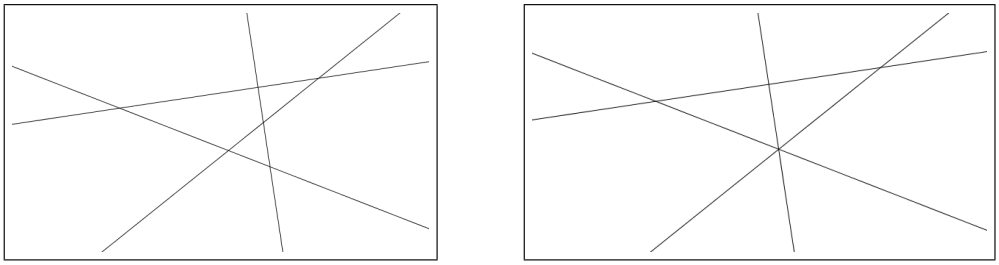
In this equation, the author assumed that a vertex is created from two intersecting lines. However, a vertex in a Voronoi diagram can be constructed from three bisector lines as shown in Figure 3.14, where these three lines are generated from three generator points. The number of regions in equation 3.9.1 will be reduced by one for each vertex that generated from three lines. Figure 3.13 shows the number of regions from four lines, where in figure 3.13a all vertices are constructed from two lines, and in Figure 3.13b, a vertex is constructed from three lines, and the number of region is reduced by one. Hence,

the number of Voronoi cells that can be created from l perpendicular bisector lines can be written as

$$rl_{hsvd} = C(l,0) \ + \ C(l,1) \ + \ C(l,2) - C(m,3) \tag{3.9.2}$$

Since a perpendicular bisector line is created from two generator points, number of l can be defined as $l = \frac{m(m-1)}{2}$. Hence, equation 3.9.2 can be written with m as

$$rl_{hsvd} = \frac{3m^4 - 10m^3 + 21m^2 - 14m + 24}{24} \tag{3.9.3}$$



(a) 2 lines vertex (b) 3 lines vertex

Figure 3.13: Number of regions related to number of lines on vertex

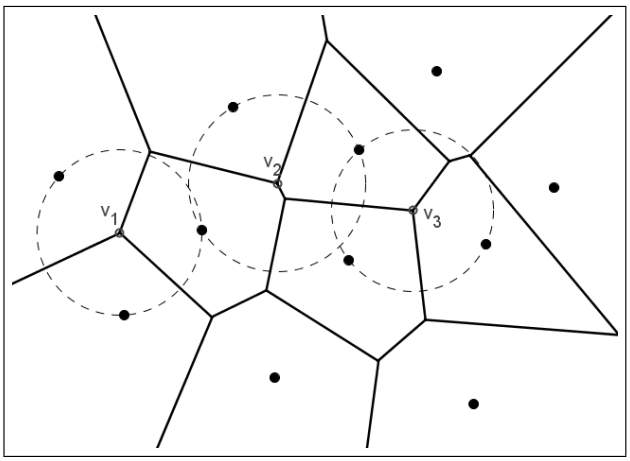


Figure 3.14: Vertex and Generator Points

Performance of HSVD depends on the number of cells created. Figure 3.15 shows that from estimation, for $m < 11$ the number of regions created in HSVD is below m^3. However, we found that the number of regions $< m^3$ where $m < 14$. From equation in 3.9.3, the number of regions generated from m generator points are at the maximum in m^4 instead of $m!$. Hence, overall performance of HSVD at the maximum is on $O(m^4)$, where m is the number of generator points.

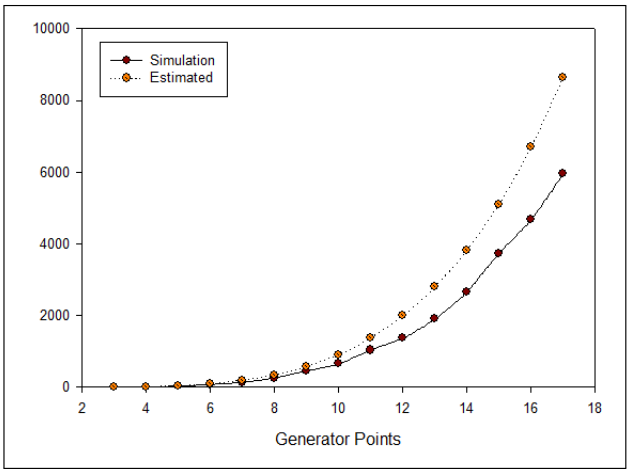
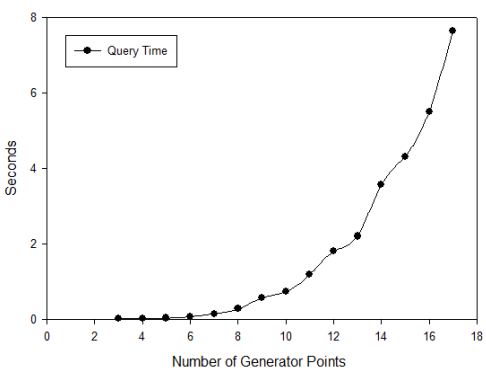
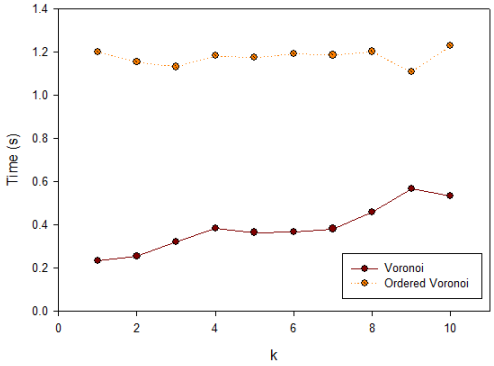


Figure 3.15: Number of cells in HSVD from simulation and estimation

Due to hardware limitation, we only conducted experiments with limited 17 points that were uniformly distributed generated using Matlab R2013b [1]. The system is built on a Java platform and the data structures are stored in MySQL database. Figure 3.15 shows the number of regions generated in simulation and the estimated number of regions from equation 3.9.2. As can be seen from this figure, the number of regions created in simulation has the same trend as the estimated number of regions, and the number of regions in simulation is always fewer than the estimated numbers of the regions.



(a) HSVD Query Time

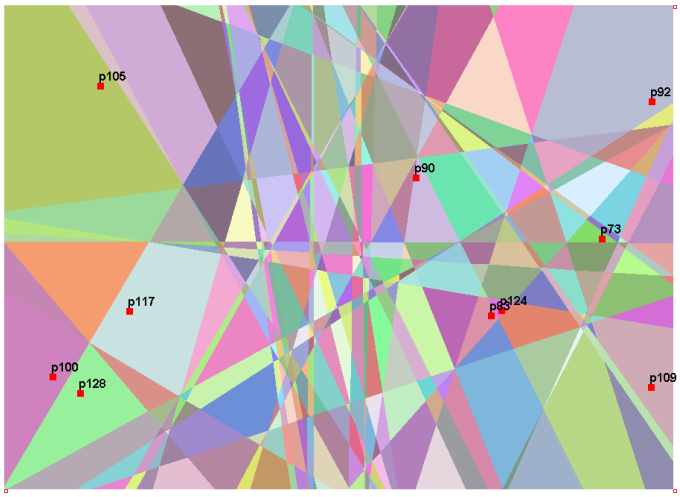


(b) Construction Time from 10 Points HSVD

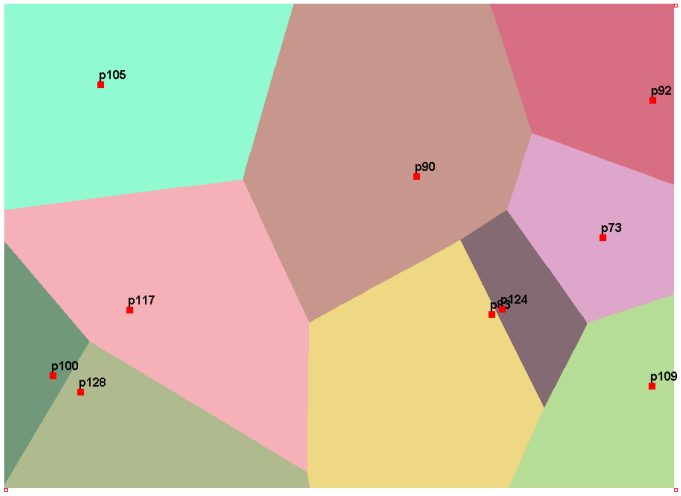
Figure 3.16: HSVD Query Time

Figure 3.16 shows the result of our experiments in reconstructing Voronoi diagram from HSVD. Figure 3.16a shows the query time to create m points Voronoi diagram from m points HSVD, and as expected, the reconstructing time depends on the number of available Voronoi cells created. Figure 3.16b shows the time taken to create a 10-points

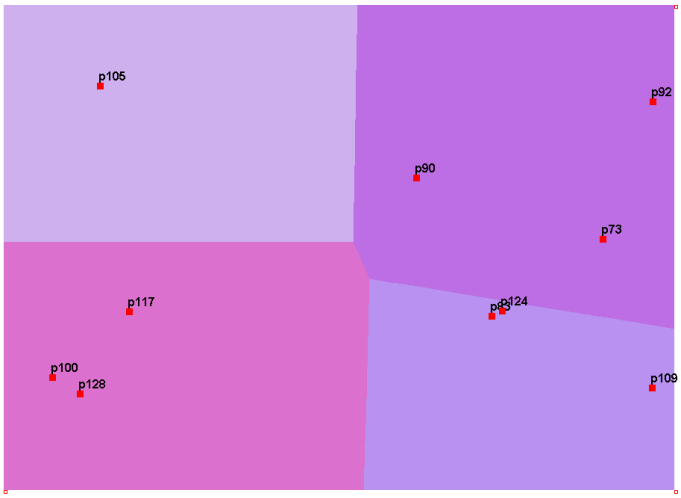
[1] http://au.mathworks.com/products/matlab/



(a) HSVD



(b) Ordinary VD



(c) Farthest Point VD

Figure 3.17: Voronoi Diagram constructed from 10 points HSVD

Voronoi diagram order-k and an ordered order-k Voronoi diagram. We also obtain the result as expected, where the cost to reconstruct the diagrams depends on the number

of HSVD Voronoi cells since the number of HSVD cells are constant. Hence, the time required to create order-k Voronoi diagram and ordered order-k Voronoi diagram are also constant.

Even though the construction cost is quite high compared to ordinary Voronoi diagram that can be built in $O(m)$(Aggarwal et al., 1989; Sugihara and Iri, 1992) or higher order Voronoi diagram that can be built in $O(mk^2\ log\ m + mk\ log^3\ m)$ (Aurenhammer and Schwarzkopf, 1991), HSVD structure has three advantages: (1) it only needs to be built once, (2) it can be used directly for various purposes, (3) it can directly identify farthest points and their regions. Figure 3.17 shows various order of Voronoi diagrams that can be constructed from HSVD (3.17a) without having to reprocess the diagram. In this example, order-1 Voronoi diagram is shown in figure 3.17b while farthest point Voronoi diagram is shown in figure 3.17c. In the next chapters, we will discuss several HSVD applications in spatial queries.

3.10 Chapter Summary

In this chapter, we present the extension of a Voronoi diagram, called highest order Voronoi diagram (HSVD), where each Voronoi cell has distance order information to all generator points. In this thesis, distance order refers as sequence. To obtain all possible Voronoi cells, this diagram is constructed using the naive method followed by the FLIP algorithm to get the distance information for all cells. Unlike a higher order Voronoi diagram, HSVD has two main advantages: (1) order=m, where this order can be used to directly identify farthest points and their regions, (2) it has all distance identification for each cell that can be used for further applications that need region with distance information, such as order-k Voronoi diagram construction or spatial queries. The performance of this diagram depends on the number of Voronoi cells created where the number of cells can be estimated at the maximum of m^4. However, the cost to utilize the diagram to create any diagram from existing m-points HSVD structure will remain constant for any k values.

Due to hardware limitation, we perform Voronoi construction with limited number of generator points. In real life situation, such as in identifying the coverage of public facilities to the surrounding residential areas, we might need to use more generator points. For example, the council wants to know which residential area have a nearest primary

school followed by a secondary school and a public library in a city. Assumed that the public facilities are generator points in HSVD, In this example, we will have to handle more generator points than in previous simulation. Therefore, we are still working to refine the algorithms to handle more generator points while maintain the overall performance as well.

Chapter 4

Nearest Neighbour Queries with HSVD

4.1 Overview

Nearest neighbour query is a common spatial query that has been widely used in various contexts. Nearest neighbour query from a query point q seeks to find all k nearest objects from q location. In this thesis, there are two different types of objects: *facility points* and *users*. A *facility point* p_i is an object that provides service, and a *user* o_j is the object that needs the service of facility points. We discussed various methods used to solve nearest neighbour queries in Section 2.2.1 and region based approach in spatial query processing in Section 2.3.

In literature, kNN queries can be solved by either the point-to-point or region-based method. Point-to-point method (Wang et al., 2000; Roussopoulos et al., 1995; Nghiem, Green and Taniar, 2013) can be used directly though the objects or by using indexing techniques. When the number of objects increases, a region-based method through region pruning or candidate region are proposed to minimize objects verification (Kolahdouzan and Shahabi, 2004; Xuan, Zhao, Taniar, Rahayu, Safar and Srinivasan, 2011; Safar, 2005).

However, since the candidate region still need the object verification step, there are at least two disadvantages: (1) the candidate region depends on the query parameter, such as the location of a query point, objects and k value; hence, the region will always change when any of these parameters changes; (2) the candidate region will have to be recreated when the query is re-issued, even with the same parameters. These problems

have been addressed in Challenges 3 and 4 where the region will always be created when the same query is issued. To answer this challenge, we employ the HSVD structure to answer nearest neighbour queries, where the HSVD structure is created from the set of facility points.

In this chapter, we will apply the solution to two variations of nearest neighbour queries from the sequence perspective. The first one is from nearest object to the farthest which is commonly known as Nearest Neighbour queries, and the second one is from the farthest to the nearest which is commonly known as Farthest Neighbour queries.

We proposed a generalisation framework to solve Nearest Neighbours problems and Farthest Neighbours problems with HSVD, which involves two main steps, which are: (1) identifying the Voronoi cell of a query point and (2) determining the answer based on the sequence of each Voronoi cell. By using HSVD, we can generalize the Nearest Neighbours and Farthest Neighbours problems as well as the variation of k^{th} queries and also both bichromatic and monochromatic query problems.

The overall contributions of this chapter are that we:

1. Introduce the region-based method for Nearest Neighbour queries with Highest Order Voronoi diagram.

2. Propose a generic framework to solve Nearest Neighbours and Farthest Neighbours problem in spatial queries.

3. Apply the framework to solve monochromatic, bichromatic, k and k^{th} variations that might appear in both nearest neighbour or farthest neighbour problems.

4.2 Queries Taxonomy, Notations and Definitions

4.2.1 Queries Taxonomy

We define the taxonomy of nearest neighbour queries with highest order Voronoi diagram based on how the sequence is read to obtain the generator points needed to answer the query. The taxonomy for nearest neighbour query is described as follow:

1. Nearest Neighbours Queries
   (a) Monochromatic k Nearest Neighbours Queries
   (b) Monochromatic kth Nearest Neighbours Queries

   (c) Bichromatic k Nearest Neighbours Queries

   (d) Bichromatic kth Nearest Neighbours Queries

2. Farthest Neighbours Queries

   (a) Monochromatic k Farthest Neighbours Queries

   (b) Monochromatic kth Farthest Neighbours Queries

   (c) Bichromatic k Farthest Neighbours Queries

   (d) Bichromatic kth Farthest Neighbours Queries

In nearest neighbours queries, the sequence will be read in a forward (left to right) manner because the sequence is ordered from the nearest to the farthest. However, in farthest neighbour queries, the sequence will be read backward (right to left).

4.2.2 Notations

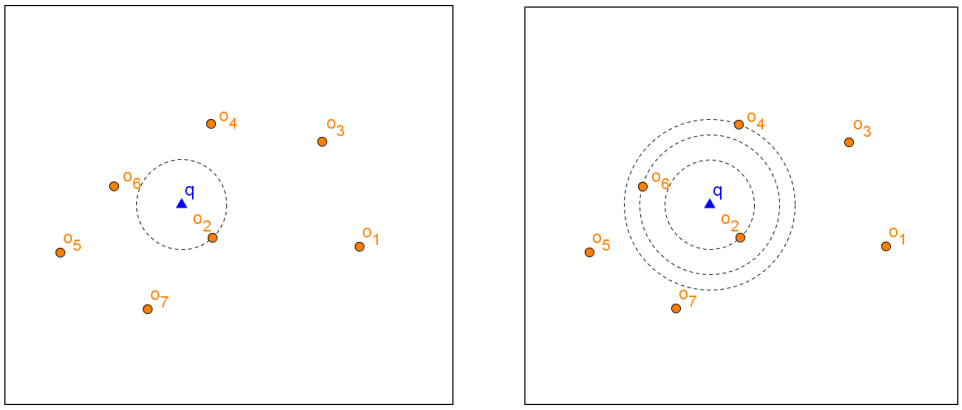
- $P = \{p_1, p_2, ..., p_m\}$ is a set of facility points, where these points are also generators of an HSVD.
- $Vh(P)$ is Highest Order Voronoi diagram from P.
- m is the number of facility points.
- $O = \{o_1, o_2, ..., o_i\}$ is set of objects.
- $U = \{u_1, u_2, ..., u_j\}$ is the set of *users* or non-generator points objects.
- q is a query point. In bichromatic $q \in U$, where in monochromatic $q \in P$
- $seq = \langle p_1, p_2, ..., p_m \rangle$ is a sequence of facility points
- $R_{seq} = R\langle p_1, p_2, ..., p_m \rangle$ is a Voronoi cell where p_1 is the nearest facility point followed by $p_2, p_3, ...$ until p_m as the farthest.
- $seq[j]$ is a facility point at position of j from a sequence. $0 \leq j < (m-1)$.
- k is the number of objects of interest which query node q wants to retrieve.
- first k-sequence is the first k numbers of generator points from a seq.
  If $seq = \langle p_5, p_4, p_2, p_3, p_1 \rangle$, first 2-sequence is $\langle p_5, p_4 \rangle$
- last k-sequence is the last k numbers of generator points from a seq.
  If $seq = \langle p_5, p_4, p_2, p_3, p_1 \rangle$, last 2-sequence is $\langle p_1, p_3 \rangle$
- k^{th}-sequence is the k^{th} nearest generator points from a seq.
  If $seq = \langle p_5, p_4, p_2, p_3, p_1 \rangle$, 4^{th}-sequence is p_3

4.2.3 Definitions

In the context of nearest neighbour queries, there are two variations to the way in which the objects are retrieved, either from the nearest known as *Nearest Neighbour* queries or from the farthest known as *Farthest Neighbour* queries. Let $O = \{o_1, o_2, ..., o_n\}$ be the set of objects, q is the query point and the Euclidean distance between query point and an object of interest is defined as $d(q, o_i)$. $A = \{a_1, a_2, ..., a_k\}$ is the set of answer from Nearest Neighbour queries where $A \subset O$. The nearest neighbour problems can be defined formally as follow:

Definition 4.2.1. Nearest Neighbour (NN) query is a method of finding the nearest object from query point q. $NN(q) = \{a | d(q, a) < d(q, o_i), \forall o_i \in O, o_i \neq a\}$.

An example of a nearest neighbour problem can be seen in figure 4.1. Figure 4.1a shows o_2 as the nearest neighbour of q, since $d(q, o_2) < \forall d(q, o_i), i \neq 2$. The problem of nearest neighbour is actually a special case of k Nearest Neighbour where $k = 1$. For $k > 1$, the definition of nearest neighbour is:



(a) $NN(q)$=o_2 (b) $3NN(q) = \{o_2, o_6, o_4\}$

Figure 4.1: Nearest Neighbour and k-Nearest Neighbours example

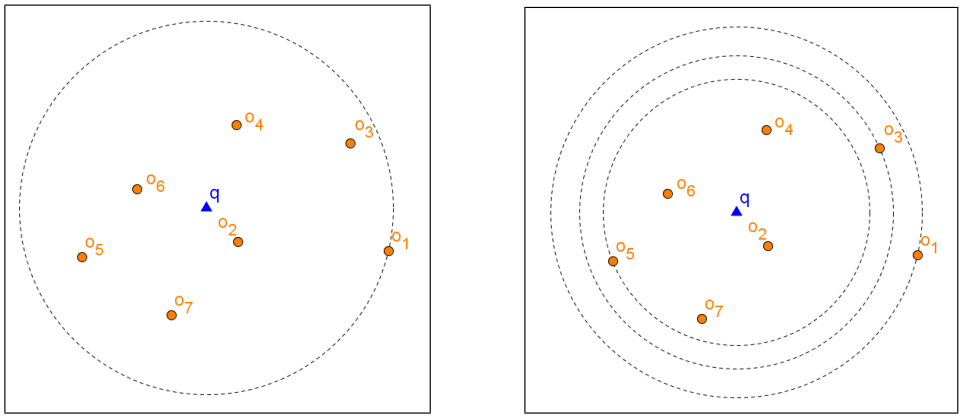
Definition 4.2.2. k **Nearest Neighbours (**k**NN)** query is a method of finding k nearest objects from query point q position. Let $A = \{a_1, .., a_k\}$ be the answer of $kNN(q)$ where $A \subset O$ and A' is the set of objects that are not the answer of the query, $A' = O - A$.

$$kNN(q) = \{A | d(q, a_i) \leq d(q, a'_j), 1 \leq i \leq k, 1 \leq j \leq (n - k)\}$$

Figure 4.1b shows an example of 3NN from q, where objects o_2, o_6, o_4 are the answer to this query.

Definition 4.2.3. Farthest Neighbour (FN) query is the method of finding the farthest object from query point q. $FN(q) = \{a | d(q,a) > d(q,o_i), \forall o_i \in O, o_i \neq a\}$

Examples of Farthest Neighbours are shown in Figure 4.2. Figure 4.2a shows FN from query point q, and the object that satisfy this query is o_1, since $d(q,o_1) > \forall d(q,o_i), i \neq 1$. The Farthest Neighbour problems can be generalized with $k > 1$ as follows



(a) $FN(q)$=o_1 (b) $3FN(q) = \{o_1, o_3, o_5\}$

Figure 4.2: Farthest Neighbour and k-Farthest Neighbours example

Definition 4.2.4. k **Farthest Neighbours (kFN)** query is a method of finding k farthest objects from query point q position. Let $A = \{a_1, .., a_k\}$ be the answer of $kFN(q)$ where $A \subseteq O$ and A' is the set of objects that are not the answer of query $A' = O - A$.

$$kFN(q) = \{A | d(q,a_i) \leq d(q,a'_j), 1 \leq i \leq k, 1 \leq j \leq (n-k)\}$$

Definition 4.2.5. k^{th} **Sequence Neighbour** is the method used to find the k^{th} objects from a query point. This can be seen from either the shortest distance perspective known as k^{th} nearest neighbour ($k^{th}NN$) problem or can also be seen as k^{th} farthest neighbour ($k^{th}FN$) problem. Let n be the number of objects, $k^{th}NN$ query can be written as $k^{th}FN$ query, where $k^{th}NN(q) = ((n+1)-k)^{th}FN(q)$

Figure 4.3 shows seven objects on the map. Object o_4 can be acknowledged as 3^{th} nearest object from query point q, or 5^{th} farthest object from query point q.

Definition 4.2.6. Monochromatic Query problem is a condition where the query point q has the same type as the object of interest. In this thesis, monochromatic refer to the condition where query point q is a part of the facility points or generator points of HSVD, and the answer to this query is all generator points that satisfy the condition.

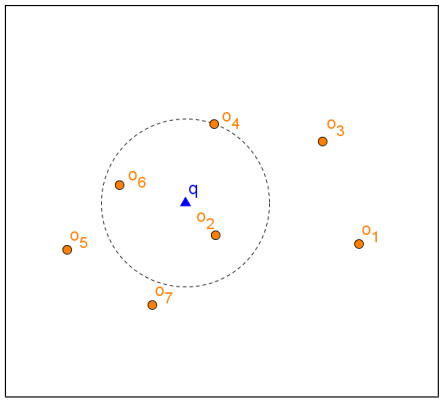


Figure 4.3: 3^{th} NN(q) = 5^{th}FN(q)

Hence, $q \in P$, $MkNN(q) = A$, where $A \subset P$ and $q \notin A$. Monochromatic query can be found in either nearest neighbour query as monochromatic nearest neighbour (MNN) queries or farthest neighbour query as monochromatic farthest neighbour (MFN) query.

Figure 4.5 shows a monochromatic query from p_1, where p_1 is also a generator point and the answer to this query is also the generator point, which is p_5.

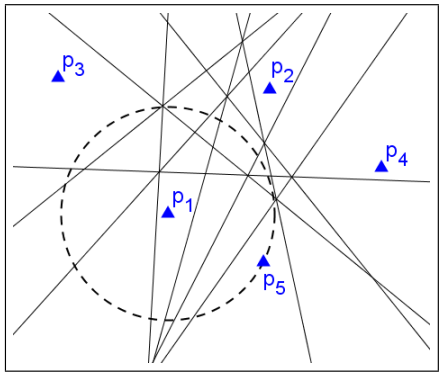


Figure 4.4: $M_1NN(p_1) = p_5$

Definition 4.2.7. Bichromatic Query problem is a condition where there are two different types of objects and the query point q is different from the answers. The HSVD structure is constructed from facility points $P = \{p_1, p_2, ..., p_m\}$, where these facility points are the generator points. In bichromatic query, the *users* are the non-generator points $U = \{u_1, u_2, ..., u_j\}$. The query is issued from non-generator points $q \in U$, and the answer for this query is a set of generator points $A \subseteq P$. Bichromatic query can be found in either nearest neighbour query as bichromatic nearest neighbour (BNN) query or farthest neighbour query as bichromatic farthest neighbour (BFN) query.

Figure 4.5 shows a BNN query from u_4, where u_4 is the *user*. The answer to this query is the generator point p_2. Since the object type from query point and the object of the answers are different, this type of query is considered as a bichromatic nearest neighbour query.

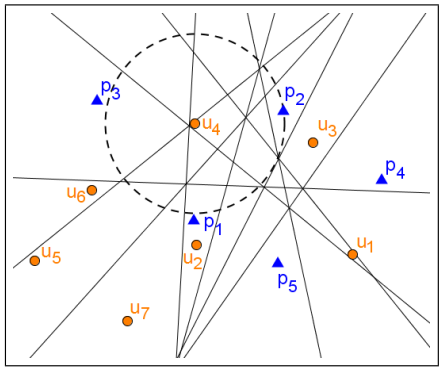


Figure 4.5: $B_1NN(u_4) = p_2$

4.3 Generalization of Nearest Neighbours Framework

The nearest neighbour approach is the method of finding objects from a query point position. Since this is the nearest approach, the process of finding the objects always starts from the nearest object from the query point and extends further; hence the term nearest neighbour. However, the objects can also be found from the farthest and come closer to the query point, this variation is commonly known as farthest neighbour approach. In this section, we divide nearest neighbours approach into two parts based on where the searching starts, from the nearest to the farthest or from the farthest to the nearest.

In nearest neighbour, the number of objects that need to be retrieved must be predefined before the query can be processed. The number of objects that need to be retrieved is denoted by k value. The k value itself is one of two types, k and k^{th}. When k value is used, then the query is expected to retrieve k objects from the start. For example $2NN(q)$ is expected to get two objects starting from the nearest, while $2FN(q)$ is expected to retrieve two objects starting from the farthest. When k^{th} is used, the query is expected to retrieve the object at the k^{th} position, either from the nearest or the farthest. $2^{th}NN(q)$ is meant to obtain the object at 2^{nd} position from the nearest, while $2^{th}FN(q)$ is meant to retrieve the object at 2^{nd} position from the farthest.

The type of objects that will be retrieved can be the same as or different from the query point. When the object type between query point and the answer set are different, this query is called *bichromatic*, while if the object type between query point and answer set are the same, this query is called *monochromatic*. For example, if a vehicle as query point issues a query to find the nearest petrol station, this query is a bichromatic query because vehicle and petrol station are two different types of objects. However, if an aeroplane as a query point issues a query to find other aeroplanes on its radar, this query is called a monochromatic query.

Since a nearest neighbour query has so many variations and these variations can be combined with each other to create more complex query problems, it is important to have a generic framework to solve these problems. For example, if a vehicle needs to find exactly the third nearest petrol station since the trip computer can estimate the remaining fuel on its tank, we can consider this query as bichromatic $3^{th}NN$.

The framework to solve nearest neighbour is based on HSVD. We consider a server that provides spatial data, and HSVD has been constructed on the objects. In the HSVD structure, the facility points are considered as the generator points, and the HSVD structure will be constructed based on facility point locations. Hence, facility points are also the generator points for the HSVD structure. Meanwhile, the users or the objects that will need the service from facility points will be considered as non-generator points objects. In a monochromatic query using the HSVD structure, the query will be issued from a generator point and the answer will be a set of generator points, while in bichromatic query using the HSVD structure, the query will be issued from non-generator points objects and the answer will be a set of generator points.

Since the answer for both monochromatic and bichromatic queries are a set of generator points and each Voronoi cell in HSVD has a sequence of generator points, the answer to nearest neighbours queries can be obtained from this sequence. To obtain the right sequence for a particular query, it is important to know the Voronoi cell where the query point is located.

The *nearest neighbour framework* is a concept of generalisation nearest neighbour solution by using the HSVD structure by considering the similar steps needed to solve the query and the same type of result sets from both monochromatic and bichromatic queries. Figure 4.6 shows the main process in this framework. There are two main steps in this

framework: (1) Voronoi cell identification that contains query point and sequence retrieval (shaded box) which can be applied to all queries variation, and (2) generator points retrieval through sequence parsing which is specific to each problem.

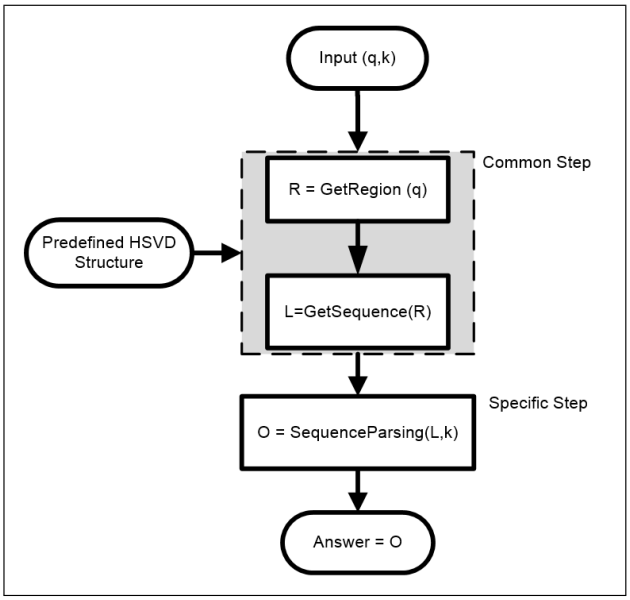


Figure 4.6: Nearest Neighbour Framework

To generalize the *SequenceParsing()* process between nearest and farthest problems for both monochromatic and bichromatic queries, we apply the *inverse()* function that inverts the sequence as shown in Figure 4.7 before choosing the right generator point in a specific position.

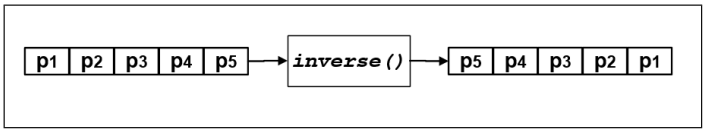


Figure 4.7: function inverse(sequence) → sequence

The inverse function is fully described in algorithm 4. This algorithm accepts sequence as the input, and will produce an inverted sequence as the output.

Algorithm 4: Inverse function

Data: *seq*
Result: *iseq* inverted sequence
1 $ctr \leftarrow 1$;
2 **for** $i = sizeOf(seq)$ ***down*** **to** *1* **do**
3 $\quad iseq[ctr] \leftarrow seq[i]$;
4 $\quad ctr++$;
5 **end**
6 return *iseq*;

The *inverse*() function is used so that we still can use the same algorithm in nearest neighbours to process farthest neighbours queries, since the main difference between these algorithm is only how to read the distance sequence in each cell.

4.4 Nearest Neighbour Queries Processing

In nearest neighbour queries, the answer can be retrieved from the sequence of a Voronoi cell where the query point is located in a left to right manner, since the left-most generator point indicates the first nearest generator point from a Voronoi cell and the right-most generator point indicates the last nearest generator point from a Voronoi cell. In nearest neighbour queries, there are four variations, which are (1) Monochromatic k Nearest Neighbours, (2) Monochromatic k^{th} Nearest Neighbour, (3) Bichromatic k Nearest Neighbour and (4) Bichromatic k^{th} Nearest Neighbour.

4.4.1 Monochromatic k Nearest Neighbours (MkNN)

In MkNN, both query point and the objects that need to be found are the same type. In nearest neighbour using HSVD, we consider monochromatic as a condition where both query point and the answers are generator points. In HSVD, each Voronoi cell has distance sequence identification to all generator points. Hence, to identify the nearest point from a generator point, we only need to find the Voronoi cell that contains the generator point. However, since the generator point is considered as the first nearest generator point in this cell, the first nearest point from this generator point will be the second nearest generator point from this Voronoi cell as can be seen in Figure 4.8 and the maximum k value that can be used from m generator points will be $(m-1)$. Hence, Monochromatic k-Nearest Neighbours can be identified from the second nearest generator point until $(k+1)^{th}$ generator points where $2 \leq k < m$.

Definition 3.4.11 in ordered higher order Voronoi diagram defines a sequence of generator points for each Voronoi cell, where the sequence indicates the ordering distance from any location in the Voronoi cell. Since Property 3.6.1 states that HSVD is a special case of an ordered HOVD, then Definition 3.4.11 is also applied in HSVD. Hence the sequence in each Voronoi cell of HSVD can be used to indicate k nearest generator points in each cell.

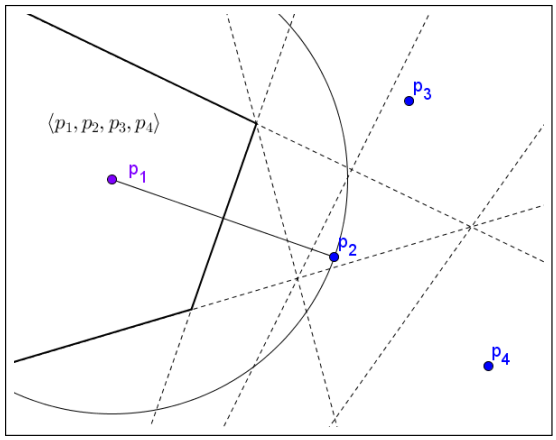


Figure 4.8: Monochromatic First Nearest Neighbour

In MkNN, since the query point is also one of the generator points, the nearest generator point is the query point itself. Hence, the first nearest neighbour from this query point starts with the second position in the sequence.

Algorithm 5: MkNN with HSVD algorithm

```
   Data: k, q
   Result: A = {a_1, a_2, ..., a_k} list of k facility points that satisfy MkNN(q)
1  initialize(A);
2  if (k < numOf(P)) then
3  |  r ← getCell(q); /* Get a cell contains q                          */
4  |  for i = 2 to k do
5  |  |  A.add(getObject(r.seq[i])); /* Get list of generator points of r  */
6  |  end
7  else
8  |  exit;/* k is bigger than number of facility points               */
9  end
```

Algorithm 5 explains the MkNN query processing using the HSVD structure. In this algorithm, the main thing that has to be done is to find the Voronoi cell that contain query point q (line 3). This algorithm assumes that query point q is part of the generator points; hence, the Voronoi cell that contains q will always be found. The next step is to obtain the first k-sequence in this Voronoi cell by skipping the first sequence (line 4, i start with 2 instead of 1). Each element of the sequence is represented as a generator point that will be stored in answer list A. The algorithm will stop when all k generator points have been retrieved.

Figure 4.9 shows an example of a MkNN query that can be solved using HSVD. Let $P = \{p_1, p_2, p_3, p_4\}$ be facility points that will construct HSVD $Vh(P)$. Query point $q = p_1$, and this query point is located in Voronoi cell R_{seq1}, where $seq1 = \langle p_1, p_2, p_3, p_4 \rangle$

is the sequence with a certain distance order. Obviously, p_1 is the nearest generator point in this Voronoi cell, so p_1 cannot be considered as the nearest point from p_1. Instead, the first nearest point from p_1 will be the second nearest generator point from this Voronoi cell, which is p_2. Hence, 3NN from p_1 will be $\langle p_2, p_3, p_4 \rangle$.

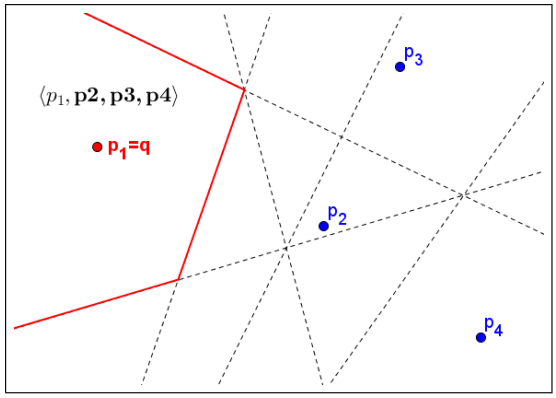


Figure 4.9: $M3NN(q)$=$\{p_2, p_3, p_4\}$

4.4.2 Monochromatic k^{th} Nearest Neighbour (Mk^{th}NN)

Monochromatic k^{th} nearest neighbour query is similar with MkNN query where the query point and the objects that need to be found are the generator points. The range of acceptable value of k will remain the same, which is $2 \leq k < m$. Since the aim of this query is to find a k^{th} nearest object from the query point, only the object in this position will be considered as the answer. Hence, the answer object for this query is a generator point in $(k+1)$ position in the sequence.

Algorithm 6: Mkth NN with HSVD algorithm

Data: k, q
Result: $A = \{a\}$, facility point that satisfies $Mk^{th}NN(q)$

```
1 m ← numOf(P)
2 if (1 ≤ k ≤ (m − 1)) then
3 |   r ← getCell(q);/* Get Voronoi cell contains q                    */
4 |   A.add(getObject(r.seq[k + 1]));/* Get kth generator points from r */
5 else
6 |   exit;/* k is bigger than number of facility points               */
7 end
```

Algorithm 6 explains the Mk^{th}NN algorithm with HSVD. This algorithm is also based on the frameworks and is similar to algorithm 5. This algorithm starts by finding the right Voronoi cell that contains query point q (line 3). Unlike algorithm 5 that retrieves all k

nearest objects, this algorithm only retrieve the object in $k+1$ position of the sequence (line 4).

Figure 4.10 shows an example of Mk^{th}NN query. Assume that there are four generator points $P = \{p_1, p_2, p_3, p_4\}$. The query point p_1 is issued Mk^{th}NN query with $k = 2$. The query point is identified in a Voronoi cell R_{seq1} where $seq1 = \langle p_1, p_2, p_3, p_4 \rangle$. Obviously, the query point q is the nearest generator point in this cell. Hence, the first nearest neighbour is the second nearest generator point, and M2thNN(q) will be the third nearest generator point from the cell, which is p_3. Therefore, $M2^{th}NN(q) = \{p_2\}$.

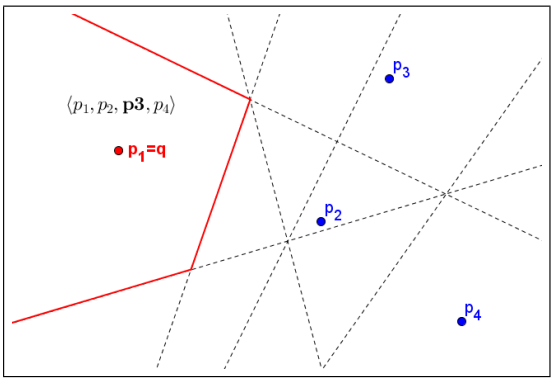


Figure 4.10: $M2^{th}NN(q)$=$\{p_3\}$

4.4.3 Bichromatic k Nearest Neighbour (BkNN)

In bichromatic k nearest neighbour queries, the object that issues a query and the objects that need to be found are different object types. In bichromatic queries with HSVD, the structure of HSVD is constructed by facility points $P = \{p_1, p_2, ..., p_m\}$; hence, the facility points are also generator points of HSVD. Meanwhile, the *users* $U = \{u_1, u_2, ..., u_j\}$ who need the service from facility points are called non-generator points and only users can issue the queries $q \in U$.

In HSVD, each Voronoi cell has sequence identification to all generator points. Hence, to identify the nearest point from a generator point, we only need to find the Voronoi cell that contains the generator point. Unlike the monochromatic queries, in bichromatic queries the query point is not a generator point. Hence, the first nearest generator point can be identified from the first object in a Voronoi cell's sequence as can be seen in Figure 4.11.

Since the first object in the sequence can be considered as the first nearest, the maximum value of k that can be accepted from m generator points is m. Hence, bichromatic

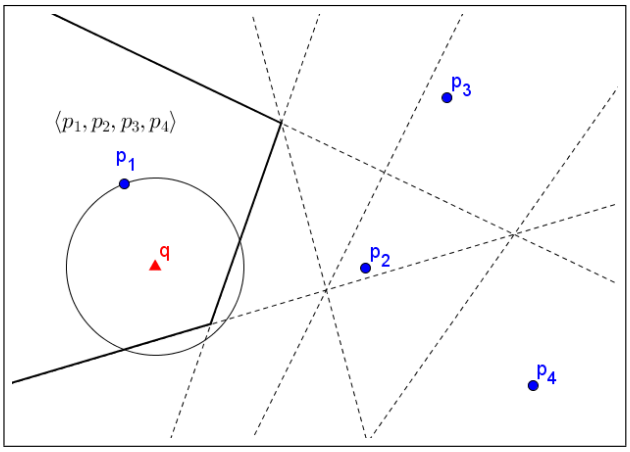


Figure 4.11: Bichromatic First Nearest Neighbour

k nearest neighbour queries can be identified from the first generator point to the m^{th} generator point of a sequence of Voronoi cell where the query point is located, and the value of k is valid only where $1 \leq k \leq m$.

Let $Vh(P)$, $q \in U \wedge q \notin P$ and R_{seq} is the Voronoi cell with certain sequence seq that contains q. Bichromatic k Nearest Neighbours (BkNN(q)) can be identified as all generator points in i-position on seq where $1 \leq i \leq k$ and $1 \leq k \leq m$.

***Algorithm 7**: BkNN with HSVD algorithm*

Data: k, q
Result: $A = \{a_1, a_2, ..., a_k\}$ list of k facility points that satisfy $BkNN(q)$

```
1 initialize(A);
2 if (k ≤ numOf(P)) then
3 |  r ← getCell(q); /* Get a Voronoi cell that contains q          */
4 |  for i = 1 to k do
5 |  |  A.add(getObject(r.seq[i]));/* Get list of Generator points of r  */
6 |  end
7 else
8 |  exit;/* k is bigger than number of facility points             */
9 end
```

Algorithm 7 explains the BkNN query processing using the HSVD structure. Similar to the MkNN algorithm, the main thing that has to be done is to find the Voronoi cell that contains query point q (line 3). This algorithm assumes that query point q is located inside the map, so the Voronoi cell that contains q will always be found. The next step is to obtain the k-sequence in this Voronoi cell (line 4) start from the first sequence. Each element of the sequence is represented as a generator point that will be stored in answer list A. Algorithm will stop when all k generator points have been retrieved.

Figure 4.12 shows an example of a BkNN query that can be solved using HSVD. Let $P = \{p_1, p_2, p_3, p_4\}$ be facility points that will construct HSVD $Vh(P)$. $U = \{u_1, u_2, ..., u_j\}$ is the set of *users* and distributed randomly on the map. Query point $q \in U$, and this query point is located in Voronoi cell R_{seq1}. Let $seq1 = \langle p_1, p_2, p_3, p_4 \rangle$ is the sequence with a certain distance order. In this Voronoi cell, the sequence indicates the ordered distance from Voronoi cell to all generator points. Hence B3NN from q will be $\langle p_1, p_2, p_3 \rangle$.

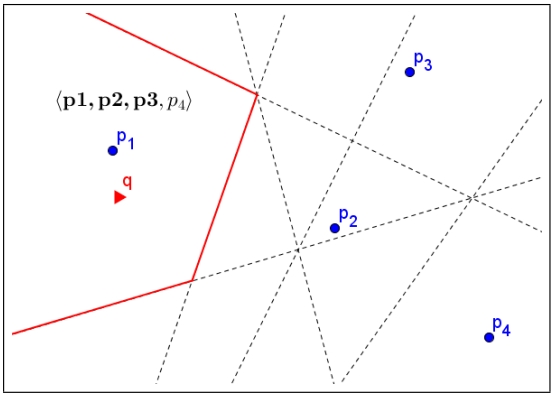


Figure 4.12: $B3NN(q)$=$\{p_1, p_2, p_3\}$

4.4.4 Bichromatic k^{th} Nearest Neighbour (Bk^{th}NN)

Bichromatic k^{th} nearest neighbour query is similar to BkNN query, except that the aim of this query is to find the generator point at the k^{th} position from the nearest. Hence, the answer object for this query is a generator point in k^{th} position in the sequence.

Let $Vh(P)$, $q \notin P$ and R_{seq} be the Voronoi cell with a certain sequence seq that contains q. Bichromatic k^{th} Nearest Neighbours (Bk^{th}NN(q)) can be identified as the generator point in i^{th}-position on seq where $i = k$ and $1 \leq k \leq m$.

Algorithm 8: Bkth NN with HSVD algorithm

Data: k, q
Result: $A = \{a\}$, facility point that satisfy $Bk^{th}NN(q)$

```
1 m ← numOf(P)
2 if (1 ≤ k ≤ m) then
3 |   r ← getCell(q); /* Get Voronoi cell contains q                    */
4 |   A.add(getObject(r.seq[k]));/* Get kth generator points from r      */
5 else
6 |   exit;/* k is bigger than number of facility points                */
7 end
```

Algorithm 8 works to comply with the nearest neighbours framework, where the first thing to do is to find the Voronoi cell that contains query point q (line 3). This algorithm

assumes that the query point is always located within the map range, so there is always a Voronoi cell that has q in it. The next step is to obtain the generator point in k^{th} position (line 4) and the query has been solved.

Figure 4.13 shows a an example of Bk^{th}NN query. Assume that there are four generator points $P = \{p_1, p_2, p_3, p_4\}$. The query point q issues Bk^{th}NN query with $k = 2$. The query point is identified in a Voronoi cell R_{seq1} where $seq1 = \langle p_1, p_2, p_3, p_4 \rangle$. Since this is a bichromatic query, the 2^{nd} nearest neighbour will be the 2^{nd} generator point in the sequence $seq1[2]$ which is p_2. Therefore, $B2^{th}NN(q) = \{p_2\}$.

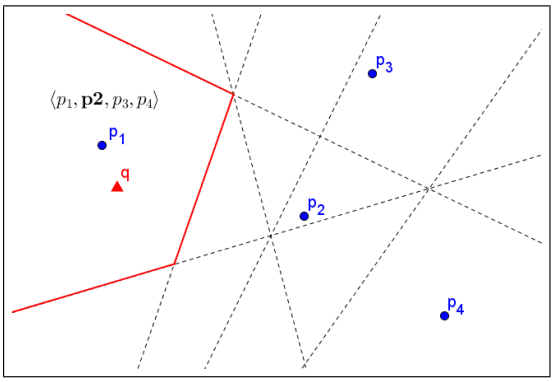


Figure 4.13: $B2^{th}NN(q)$=$\{p_2\}$

4.5 Farthest Neighbour Queries Processing

In farthest neighbour queries, the answer can be retrieved from the sequence of a Voronoi cell where the query point is located from the right to the left, since the right-most generator point indicates the first farthest generator point from a Voronoi cell and the left-most generator point indicates the last farthest generator point from a Voronoi cell. We divide farthest neighbour queries into four variations: (1) Monochromatic k Farthest Neighbours, (2) Monochromatic k^{th} Farthest Neighbour, (3) Bichromatic k Farthest Neighbours and (4) Bichromatic k^{th} Farthest Neighbour.

4.5.1 Monochromatic k Farthest Neighbours (MkFN)

Monochromatic k farthest neighbour query (MkFN) has similar characteristic with MkNN where the object that issues the query and the objects retrieved to answer the query are the same type, which is the generator points type, and $1 \leq k < m$. Unlike MkNN, the objects to answer the query are retrieved from right to left. The first farthest neighbour

is the generator point in m^{th} position on the sequence, so the k farthest objects are k objects from the farthest and continue to the object before farthest. Since the query point in a monochromatic query is also one of the generator points, the value of k is $1 \leq k < m$. Hence, MkFN can be identified from the farthest generator points to the $(m + 1 - k)$ generator points, where $1 \leq k < m$.

To simplify the process, inverse the sequence by using *inverse()* function, so that we can construct the algorithm of MkFN using the MkNN algorithm. However, since this algorithm scans the sequence from the back, the value of k is equal to m.

Algorithm 9: MkFN with HSVD algorithm

```
   Data: k, q
   Result: A = {a_1, a_2, ..., a_k} list of k facility points that satisfy MkFN(q)
 1 initialize(A);
 2 m ← numOf(P)
 3 if (k < m) then
 4 |   r ← getCell(q); /* Get Voronoi cell contains q                    */
 5 |   r.seq ← inverse(r.seq);
   |   /* reverse the order of seq                                       */
 6 |   for i = 1 to k do
 7 |   |   A.add(getObject(r.seq[i]));/* Get list of Generator points of r */
 8 |   end
 9 else
10 |   exit;/* k is bigger than number of facility points               */
11 end
```

Algorithm 9 explains how MkFN works with HSVD in nearest neighbours framework. The first step in answering this query is to find the right Voronoi cell that contains the query point (line 3), followed by reversing the distance sequence with *inverse()* function (line 5). The k farthest generator points are retrieved from the left to the right in the same nearest neighbour manner since the sequence has been inverted (line 6-8). The algorithm will end when all k generator points have been retrieved.

Figure 4.14 shows an example of an MkFN query that can be solved using HSVD. Let $P = \{p_1, p_2, p_3, p_4\}$ be facility points that will construct HSVD $Vh(P)$. Query point $q = p_1$, and this query point is located in Voronoi cell R_{seq1}, where $seq1 = \langle p_1, p_2, p_3, p_4 \rangle$ is the sequence with a certain distance order. Obviously, p_1 is the nearest generator point in this Voronoi cell, so p_1 cannot be considered as the part of the answer. The first farthest neighbour from q will be the generator point on the last sequence, which is $p4$ and the

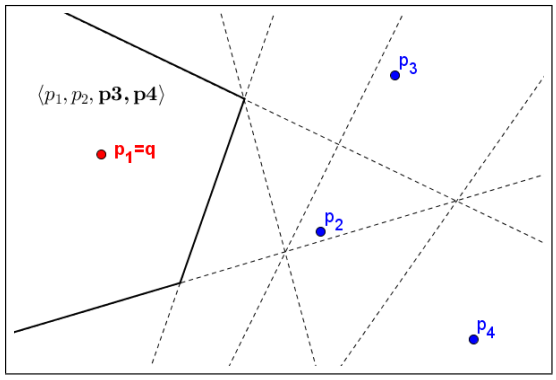


Figure 4.14: $M2FN(q)$=$\{p_3, p_2\}$

second farthest neighbour will be the generator point before the farthest generator point, which is p_3. Hence, M2FN from q will be $\{p_4, p_3\}$.

4.5.2 Monochromatic k^{th} Farthest Neighbour (Mk^{th}FN)

Monochromatic k^{th} farthest neighbour query is a variation of the MkFN query where the answer to this query is a generator point located in k^{th} order from the right on the sequence of a Voronoi cell where the query point is located. Since it is a monochromatic query, the value of k is in the range of $1 \leq k < m$.

Let HSVD of P is $Vh(P)$,q $\in P$ and R_{seq} is the Voronoi cell with certain sequence seq that contains q. Monochromatic k Farthest Neighbours problem (Mk^{th}FN(q)) can be identified as the generator points in i^{th}-position on seq where $i = (m + 1 - k)$ and $1 \leq k < m$.

***Algorithm 10**: Mkth FN with HSVD algorithm*

Data: k, q
Result: $A = \{a\}$, facility point that satisfies $Mk^{th}FN(q)$

```
1 m ← numOf(P)
2 if (k≤ m)then
3 |  r ← getCell(q); /* Get Voronoi cell contains q                    */
4 |  r.seq ← inverse(r.seq);
  |  /* reverse the order of seq                                       */
5 |  A.add(getObject(r.seq[k]));/* Get kth generator points from r     */
6 else
7 |  exit;/* k is bigger than number of facility points                */
8 end
```

Algorithm 10 shows the process of solving a MkthNN query by using the HSVD structure in nearest neighbour framework. As expected, this algorithm is very similar to the MkthNN algorithm, except with additional *inverse()* function. The algorithm starts by

determining the Voronoi cell that contains q (line 3) and followed by reversing the sequence (line 4), and then retrieving the generator point at k^{th} order from inverted sequence (line 5). The algorithm will stop after the generator point has been retrieved.

An example of MkthNN query can be seen in Figure 4.15. Let $P = \{p_1, p_2, p_3, p_4\}$ be facility points that will construct HSVD $Vh(P)$. Query point $q = p_1$, and this query point is located in Voronoi cell R_{seq1}, where $seq1 = \langle p_1, p_2, p_3, p_4 \rangle$ is the sequence with a certain distance order and $k = 2$. Obviously, p_1 is the nearest generator point in this Voronoi cell, so p_1 cannot be considered as the part of the answer. Similar to the MkFN query, the first farthest neighbour from q will be the generator point in the last sequence. Therefore, M2thNN from q will be $\{p_3\}$.

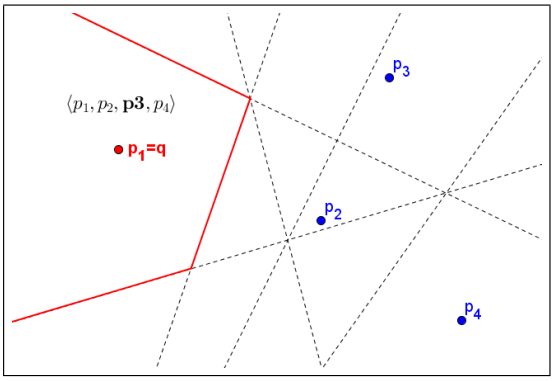


Figure 4.15: $M2^{th}FN(q)$=$\{p_3\}$

4.5.3 Bichromatic k Farthest Neighbour (BkFN)

Bichromatic k farthest neighbour query is similar similar to the BkNN query in that the object type that issues the query is different from the object type that answers the query. Like other bichromatic queries, the value of k is within the range of $1 \leq k \leq m$. However, unlike the BkNN query, the generator points are searched from the right-most generator point in the sequence. Hence, the answer to a BkFN query is all generator points in the sequence of a Voronoi cell where the query point is located, starting from m^{th} position until $the(m + 1 - k)^{th}$ position.

Let $Vh(P)$, $q \notin P$ and R_{seq} is the Voronoi cell with a certain sequence seq that contains q. BkFN(q) can be identified as all generator points in i-position on seq where $m \geq i \geq (m + 1 - k)$ and $1 \leq k \leq m$.

The algorithm for BkFN with HSVD structure on nearest neighbour framework can be described as follows.

***Algorithm 11**: BkFN with HSVD algorithm*

Data: k, q
Result: $A = \{a_1, a_2, ..., a_k\}$ list of k facility points that satisfy $BkFN(q)$

```
1  initialize(A);
2  m ← numOf(P)
3  if (k ≤ m)then
4  |  r ← getCell(q); /* Get Voronoi cell contains q                    */
5  |  r.seq ← inverse(r.seq);
   |  /* reverse the order of seq                                       */
6  |  for i = 1 to (k) do
7  |  |  A.add(getObject(r.seq[i]));/* Get list of Generator points of r */
8  |  end
9  else
10 |  exit;/* k is bigger than number of facility points               */
11 end
```

Algorithm 11 shows the BkFN query processing with the HSVD structure on the nearest neighbours framework which is very similar to the BkNN algorithm, except with the additional *inverse()* function. The first step is to find the right Voronoi cell that contains the query point q (line 4) followed by inversion of the sequence (line 5). After that, the algorithm starts to retrieve the generator points from the first to the k previous generator points (line 6-8) on the sequence. The algorithm will not stop until k generator points have been retrieved. Since this is a bichromatic query, the value of k is valid in $1 \leq k \leq m$.

Figure 4.16 shows an example of a BkFN query. Let $P = \{p_1, p_2, p_3, p_4\}$ be facility points that will construct the HSVD $Vh(P)$. $O = \{o_1, o_2, ..., o_n\}$ are non-generator point objects and are distributed randomly on the map. Query point q is one of the non-generator point objects $q \in O$. Assume that this query point is located in Voronoi cell R_{seq1}, and $seq1 = \langle p_1, p_2, p_3, p_4 \rangle$ is the sequence with certain distance order and $k = 2$. In farthest neighbour queries, the first generator point is the last generator point in the sequence *seq*; therefore, this query will retrieve two generator points from the right. With this process, the answer for B2FN(q) will be $\langle p_4, p_3 \rangle$.

4.5.4 Bichromatic k^{th} Farthest Neighbour (Bk^{th}FN)

Bichromatic k^{th} farthest neighbour query is a special case of a BkFN query where the answer to this query is only a generator point in k^{th} position from the sequence of a Voronoi cell where the query point q is located. The k^{th} position from the right can be

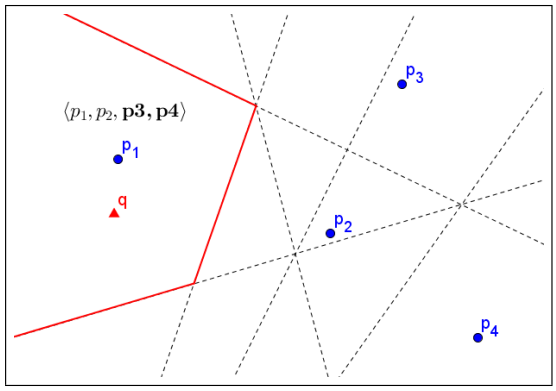


Figure 4.16: $B2FN(q)$=$\{p_4, p_3\}$

written as $(m+1-k)^{th}$ position, so the answer to this query is the generator point in $(m+1-k)^{th}$ position in the sequence of the Voronoi cell where the query point q is located.

Let $Vh(P)$, $q \notin P$ and R_{seq} be the Voronoi cell with a certain sequence seq that contains q. A Bk^{th}FN(q) query can be identified as the generator points in i^{th}-position on seq where $i = (m+1-k)$ and $1 \leq k \leq m$.

The algorithm based on the HSVD structure on the nearest neighbours framework to answer Bk^{th}FN query is shown in algorithm 12.

Algorithm 12: Bkth FN with HSVD algorithm

Data: k, q
Result: $A = \{a\}$, facility point that satisfy $Bk^{th}FN(q)$

```
1 m ← numOf(P)
2 if (1 ≤ k ≤ m) then
3 |   r ← getCell(q); /* Get Voronoi cell contains q                    */
4 |   r.seq ← inverse(r.seq);
  |   /* reverse the order of seq                                       */
5 |   A.add(getObject(r.seq[k])); /* Get kth generator points from r    */
6 else
7 |   exit;/* k is bigger than number of facility points                */
8 end
```

In Algorithm 12, region identification to determine the right Voronoi cell that contains query point q has to be done in the first place to comply with nearest neighbour framework (line 3) and is followed by sequence inversion (line 4). The next step is to find the generator point at k^{th} position on the sequence of a Voronoi cell that contains a query point (line 5). The algorithm will finish after the generator point has been retrieved.

Figure 4.17 shows an example of a Bk^{th}FN query. This example uses the same condition as does the BkFN example. Let $P = \{p_1, p_2, p_3, p_4\}$ be facility points that will construct the HSVD $Vh(P)$. $O = \{o_1, o_2, ..., o_n\}$ are non-generator point objects and are distributed

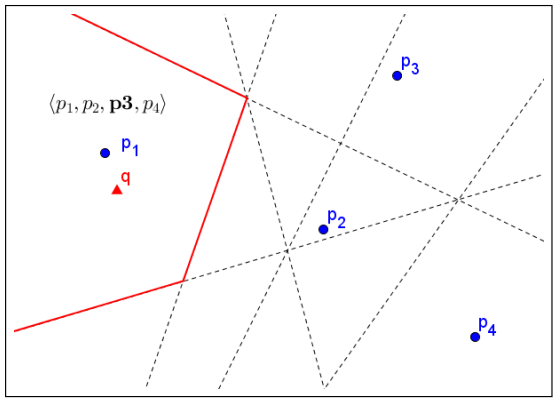


Figure 4.17: $B2^{th}FN(q)$=$\{p_3\}$

randomly on the map. Query point q is one of the non-generator point objects $q \in O$. Assume that this query point is located in Voronoi cell R_{seq1}, and $seq1 = \langle p_1, p_2, p_3, p_4 \rangle$ is the sequence with a certain distance order and $k = 2$. To answer this query, the algorithm has to retrieve the second generator point from the right in the sequence of a Voronoi cell where the query point is located. Therefore, B2thFN(q) = $\{p_3\}$.

4.6 Evaluation

We conducted our experiments on a pre-computed HSVD with ten generator points and various numbers of non-generator points. This evaluation is meant to obtain the behaviour of the algorithm under certain conditions. We generate the generator points and non-generator points randomly by using Matlab 2013b. The experiments are conducted on Windows 8.1 with i7-3520M CPU@2.90GHz and 8GB RAM. We implemented the algorithms on Java NetBeans IDE 7.3.1 with MySQL Server 4 as the database server. We conducted four experiments to obtain: (1) the performance of monochromatic queries, (2) the performance of bichromatic queries, and (3) the impact of number of non-generator points in bichromatic queries, and (4) the impact of number of generator points in both monochromatic and bichromatic queries. We run each type of query with the same k for 10,000 times consecutively in order to obtain the best overall system behaviour under repetitive query requests.

Figure 4.18 shows the cells that need to be read in order to obtain a cell that contains query point q in processing monochromatic queries with different k value. As we can see from this figure, the number of cells that need to be read for four variations of queries that include variation of k, k^{th} , farthest or nearest in monochromatic queries remain constant

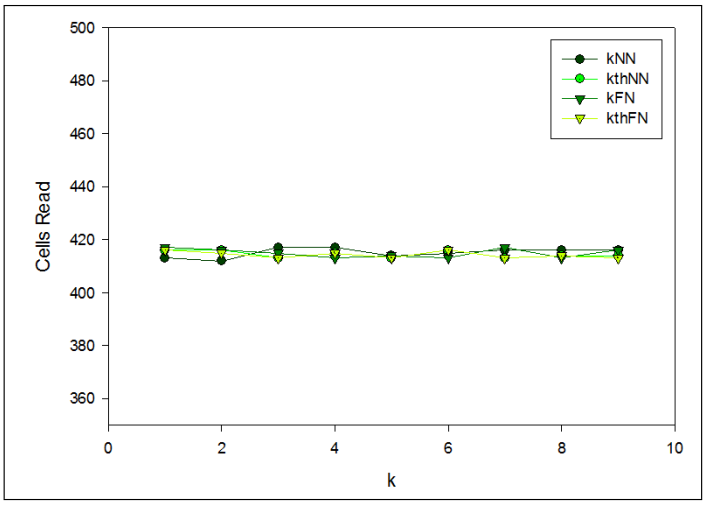


Figure 4.18: Voronoi cells that need to be read on monochromatic queries

for all values of k. This is because the Voronoi cells are stored without using any indexes and the method for finding the correct cells is only the sequential search. Hence, the cost of finding the right cells will remain similar (constant) for all monochromatic query variations and for all values of k.

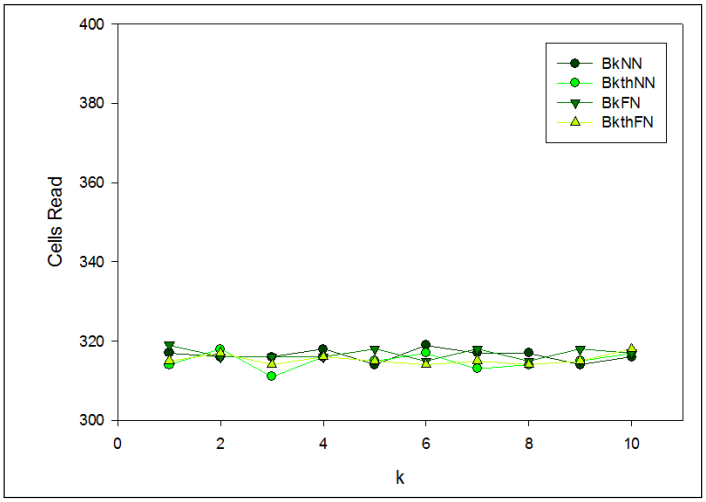


Figure 4.19: Voronoi cells that need to be read on bichromatic queries with 1000 non-generator points

Figure 4.19 shows the number of cells that need to be read before the right cell that contains query point q can be found in four types of bichromatic queries which are BkNN, Bk^{th}NN, BkFN and Bk^{th}FN. The query point is chosen randomly from 1000 non-generator points that are distributed randomly. From this graph, we can see that the number of Voronoi cells that need to be processed remain similar and constant for all query types and all values of k. Because no index is available, the cells need to be read sequentially.

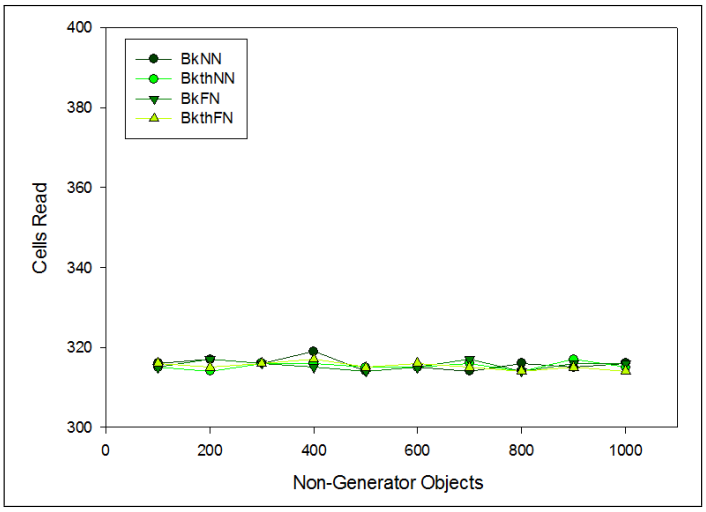


Figure 4.20: Voronoi cells that need to be read on Bichromatic queries with 1000 non-generator points

In Figure 4.20, we investigate the effect of a number non-generator points, ranging from 200 to 1000 non-generator points, where the query point is chosen randomly from these points. As we can see, the number of non-generator points in BNN and BFN does not influence the system performance, since the number of Voronoi cells that need to be read remains the same for all numbers of non-generator points.

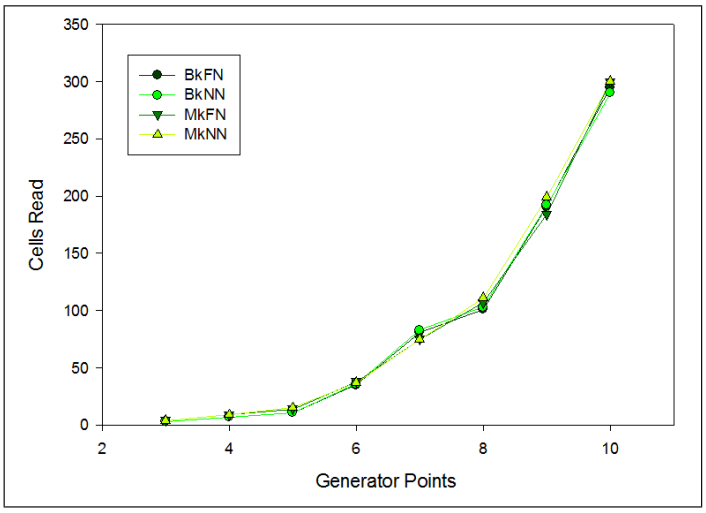


Figure 4.21: Voronoi cells to be read with different Generator Points

Figure 4.21 shows the different numbers of Voronoi cells that need to be read from different generator points. We evaluate the impact of the number of generator points on overall system performance. This figure shows that the number of cells to be read increases with the increment number of generator points, since the number of available Voronoi cells in the HSVD depends on the number of generator points.

From these experiments, we can conclude that the performance of nearest neighbour queries with HSVD is always constant since the processing cost depends on the number of Voronoi cells in HSVD which can be estimated using m^4 on Chapter 3. Therefore, nearest neighbour queries will have the same processing cost as that of other different types (nearest or farthest), any different k value, and any different non-generator points if using the same HSVD structure.

4.7 Chapter Summary

In this chapter, we presented nearest neighbour query processing with a region-based approach. We proposed a generic nearest neighbour framework with HSVD, where this framework can be applied to solve several variations of nearest neighbour queries: Monochromatic k Nearest Neighbours queries (MkNN), Monochromatic k^{th} Nearest Neighbour queries (Mk^{th}NN), Bichromatic k Nearest Neighbours queries (BkNN), Bichromatic k^{th} Nearest Neighbour queries (Bk^{th}NN), Monochromatic k Farthest Neighbours queries (MkFN), Monochromatic k^{th} Farthest Neighbours queries (Mk^{th}FN), Bichromatic k Farthest Neighbours queries (BkFN) and Bichromatic k^{th} Farthest Neighbour queries (Bk^{th}FN).

Our analysis shows that the performance of nearest neighbour framework with HSVD will not be affected by the number of k, various numbers of non-generator points, query types and repetitive queries if using the same predefined HSVD structure. However this performance depends on the number of Voronoi cells generated from HSVD since no indexes are applied to the HSVD structure.

Chapter 5

Reverse Nearest Neighbour Region with HSVD

5.1 Overview

Reverse nearest neighbour query (RNN) is another version of nearest neighbour query introduced by (Stanoi et al., 2000) where the aim is to obtain all objects that consider the query point as one of their nearest neighbours. We have discussed various methods for solving reverse nearest neighbours queries in Section 2.2.3 and the region-based approach in spatial query processing in Section 2.3.

In literature, RNN queries can be solved either by the point-to-point or the region-based method. In the point-to-point method, each candidate checks whether the query point q is one of its nearest neighbours (Katayama and Satoh, 1997; Wang et al., 2000, 2005). In region-based method, either of two approaches can be used. The first one is by using the approximate region to prune unnecessary candidates (Wu et al., 2008b; Tran et al., 2010). This method must be followed by an objects verification step to obtain the correct objects for the query. The second approach is by using the true region method, where the aim is to obtain the region where the correct objects are located (Cheema et al., 2011; Nghiem, Maulana, Waluyo, Green and Taniar, 2013; Adhinugraha et al., 2013). However, this method also has three major problems: (1) the regions from this method can only be used for a specific query point and k value; hence, if the query point issues other queries with different value of k, or another query point issues a query, the region created from the previous query will become invalid, (2) the same region has to be

recreated if the same query point issues the same query with the same k value, (3) none of these methods can be used to identify reverse farthest regions.

In this chapter, we will apply HSVD properties to two variations of reverse queries, which are RNN queries and RFN queries along with the variations of these queries. We propose an index structure for cells retrieval called **Cell Level Index (CLI)** that can be applied to both RNN and RFN queries. We also apply this method to other variations of reverse queries, which are k^{th} sequence and group reverse queries.

The overall contributions of this chapter are as follows:

1. The introduction of a region-based method for RNN and RFN queries with HSVD.
2. A proposed cells retrieval structure called **Cell Level Index (CLI)** to avoid unnecessary cells retrieval during spatial query processing.
3. The application of a CLI structure on HSVD to solve both Reverse Nearest and Reverse Farthest queries, as well as their variants in k,kth and group reverse queries.

5.2 Queries Taxonomy, Notations and Definitions

5.2.1 Queries Taxonomy

In the reverse queries, we define the taxonomy based on how the index is read in order to obtain the influence region of a query point at certain k values. The taxonomy of reverse queries is described as follows:

1. Reverse Nearest Neighbours Queries (RNN)
   (a) Reverse k Nearest Neighbours Queries (RkNN)
   (b) Reverse kth Nearest Neighbours Queries (RkthNN)
2. Reverse Farthest Neighbours Queries (RFN)
   (a) Reverse k Farthest Neighbours Queries (RkFN)
   (b) Reverse kth Farthest Neighbours Queries (RkthFN)
3. Group Reverse Queries
   (a) Group Reverse Nearest Neighbours Queries (GRNN)
   (b) Group Reverse Farthest Neighbours Queries (GRFN)

In RNN, the CLI index is read from the lowest to the highest to represent the nearest region to the farthest, while in RFN, the CLI index is read from the highest to the lowest to represent the farthest region to the nearest. However, in group reverse queries, the CLI

index can be read only for the first query point, and the other query points will be used to refine the region of the first query point.

5.2.2 Notations

- $P = \{p_1, p_2, ..., p_m\}$ is a set of facility points, where these points are also generators of HSVD.
- $Vh(P)$ is HSVD from P.
- m is the number of facility points. $m = |P|$.
- $U = \{u_1, u_2, ..., u_j\}$ is set of users that need the service from facility points P. in HSVD structure, U is set of non-generator point objects.
- q is query point. In reverse query, $q \in P$
- $Q = \{q_1, q_2, ..., q_k\}$ is the set of query points in group reverse queries, where $k = |Q|$ and $Q \subseteq P$.
- $seq = \langle p_1, p_2, ..., p_m \rangle$ is a sequence of facility points
- $R_{seq} = R\langle p_1, p_2, ..., p_m \rangle$ is a region/Voronoi cell where p_1 is the nearest facility point followed by $p_2, p_3, ...$ until p_m as the farthest.
- $seq[j]$ is a facility point at position of j from a distance sequence. $0 \leq j < (m-1)$.
- k is the number of objects of interest which query node q wants to retrieve.
- first k-sequence is the first k numbers of generator points from a seq.

  If $seq = \langle p_5, p_4, p_2, p_3, p_1 \rangle$, first 2-sequence is $\langle p_5, p_4 \rangle$
- last k-sequence is the last k numbers of generator points from a seq.

  If $seq = \langle p_5, p_4, p_2, p_3, p_1 \rangle$, last 2-sequence is $\langle p_1, p_3 \rangle$
- k^{th}-sequence is the k^{th} nearest generator points from a seq.

  If $seq = \langle p_5, p_4, p_2, p_3, p_1 \rangle$, 4^{th}-sequence is p_3
- $p^{[k]}$ indicates a generator point p at k^{th} position on distance sequence.
- $R(p^{[k]})$ refers to all Voronoi cells that have generator point p at k^{th} position on the distance sequence.
- $\langle [*,]p_i[,*] \rangle$ refers to any generator points except p_i on the sequence.

  $\langle p_3, *, *, * \rangle$ is the Voronoi cells with 4 generator points that have p_3 at the first position on the sequence.

  $\langle *, *, p_3, * \rangle$ is the Voronoi cells with 4 generator points that have p_3 at the third position on the sequence.

- $C(Q,k)$ is the first k sequence combinations of set Q, where $Q \in P$ and $k = |Q|$.
  If $Q = \{p_1, p_2, p_3\}$ and $k = 3$, $C(Q,k) = \{\langle p_1, p_2, p_3, ...\rangle, \langle p_1, p_3, p_2, ...\rangle,$
  $\langle p_2, p_1, p_3, ...\rangle, \langle p_2, p_3, p_1, ...\rangle, \langle p_3, p_1, p_2, ...\rangle, \langle p_3, p_2, p_1, ...\rangle\}$.

5.2.3 Definitions

In the context of reverse nearest neighbour queries, there are two variations to how the objects are retrieved, either starting from the nearest, called *reverse nearest neighbour* queries, or from the farthest, called *reverse farthest neighbour* queries. Assume that there are two distinct sets of objects. Let $P = \{p_1, p_2, ..., p_m\}$ is a set of facility points and $U = \{u_1, u_2, ..., u_j\}$ is a set of *users* that need the service from facility points. The reverse nearest query is issued from a facility point to obtain a set of users that considers this facility as the nearest, so that $q \in P$.

Definition 5.2.1. Reverse Nearest Neighbour (RNN) query is a method used to find users in U that consider query point q as their nearest facility point.

$$RNN(q) = \{u | q = NN(u), u \in U\}$$

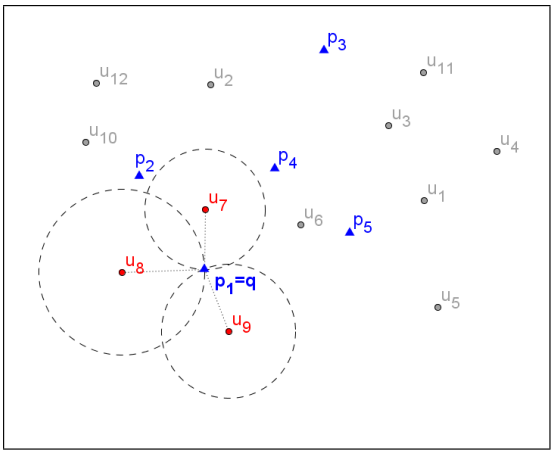


Figure 5.1: $RNN(q)$=$\{u_7, u_8, u_9\}$

Figure 5.1 shows an example of RNN query. Assume that the facility point p_1 is the query point. A reverse nearest neighbour query from p_1 means finding all objects of interest that consider the query point as the nearest. From this figure, only objects u_7, u_8 and u_9 consider p_1 as the nearest.

Definition 5.2.2. Reverse k Nearest Neighbour(RkNN) query is a method to find users in U that consider the query point q as one of their k nearest facility points.

$$RkNN(q) = \{u | q \in kNN(u), u \in U, 1 \leq k \leq |P|\}$$

e

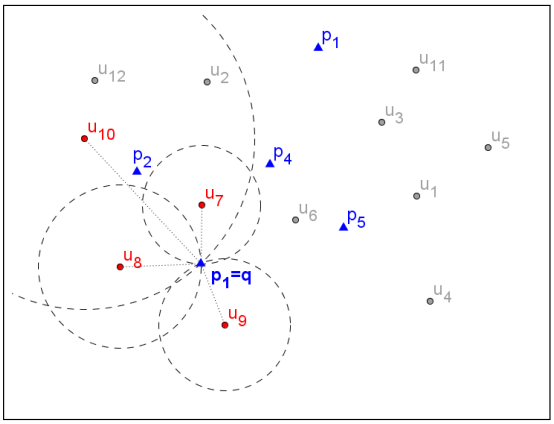


Figure 5.2: $R2NN(q)$=$\{u_7, u_8, u_9, u_{10}\}$

An example of RkNN can be seen on figure 5.2, where the query point is p_1 and $k = 2$. The aim of this query is to find all users u that consider p_1 as one of their 2NN facility points. Hence, all users that consider p_1 as the first nearest or second nearest will be considered as R2NN(p_1).

Definition 5.2.3. Reverse k^{th} Nearest Neighbour (Rk^{th}NN) query is the method of finding users in U that consider the query point q as their k^{th} nearest facility point.

$$Rk^{th}NN(q) = \{u | q = k^{th}NN(u), u \in U, 1 \leq k \leq |P|\}$$

Reverse k^{th} nearest neighbour is a specific form of RkNN query, where the aim is to find the objects of interest that consider the query point as the k^{th} nearest facility point.

An example of Rk^{th}NN can be seen in Figure 5.3, where the query point is p_1 and $k = 2$. This query is meant to find all users u that consider p_1 as their 2^{nd} nearest facility point. In this figure, only user u_{10} considers p_1 as its 2^{nd} nearest facility point, hence R2^{th}NN(p_1) = $\{u_{10}\}$.

Definition 5.2.4. Group Reverse Nearest Neighbour (GRNN) query is the method of finding users in U that consider k number of query points in set of Q as their k nearest facility points.

$$GRNN(Q) = \{u | Q = kNN(u), u \in U, k = |Q|\}$$

Group reverse nearest neighbour query is a special modification of the RNN query, where the query is issued from k number of query points in set $Q = \{q_1, q_2, ..., q_k\}$. The

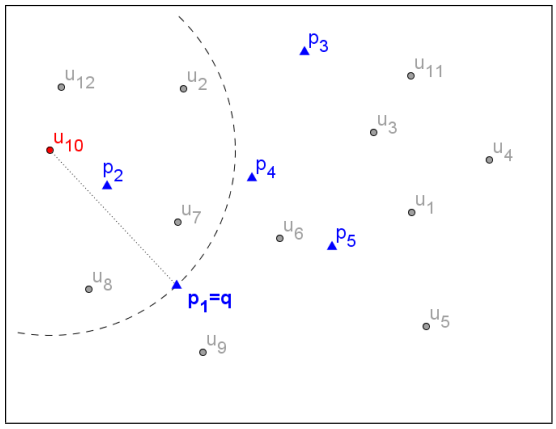


Figure 5.3: $R2^{th}NN(q)$=$\{u_{10}\}$

aim is to find all objects of interest that consider these query points as their k nearest facility points. An example of this query can be seen in Figure 5.4. In this example, the query is issued from p_3, p_4, p_5, and the objective is to find all users u that consider these query points as their kNN.

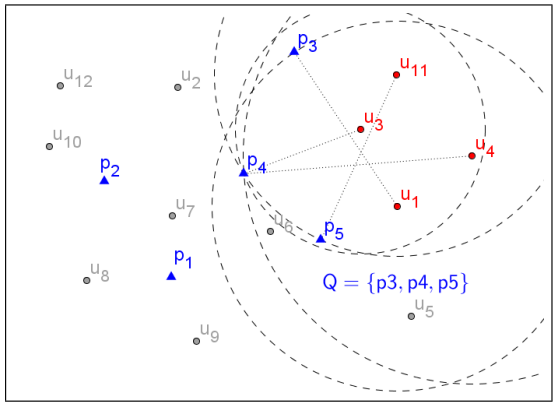


Figure 5.4: $GRNN(Q)$=$\{u_1, u_3, u_4, u_{11}\}$

Definition 5.2.5. Reverse Farthest Neighbour (RFN) query is a method of finding users in U that consider the query point q as their farthest facility point.

$$RFN(q) = \{u | q = FN(u), u \in U\}$$

An example of RFN query is given in Figure 5.5. Here, objects u_6, u_7, u_8, u_9 consider query point p_1 as the farthest facility point.

Definition 5.2.6. Reverse k Farthest Neighbour (RkFN) query is a generalization of RFN query where the aim is to find users in U that consider query point q as one of their k farthest facility points.

$$RkFN(q) = \{u | q \in kFN(u), u \in U, 1 \leq k \leq |P|\}$$

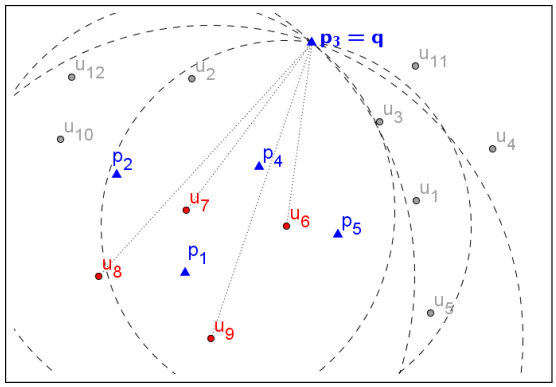


Figure 5.5: $RFN(q)$=$\{u_6, u_7, u_8, u_9\}$

Figure 5.6 shows the example of RkFN query. The query point is p_3 and the value of $k = 2$, so the aim of this query is to find all users that consider the query point p_3 as one of the two farthest facility points. From this example, we can see that users u_6, u_7, u_8, u_9 consider p_3 as the farthest facility point and objects u_5, u_{10}, u_{12} consider p_3 as the second farthtest facility point. Hence, $R2FN(p_3) = \{u_5, u_6, u_7, u_8, u_9, u_{10}, u_{12}\}$.

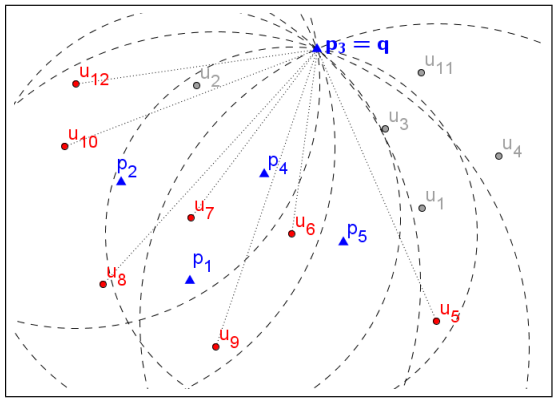


Figure 5.6: $R2FN(q)$=$\{u_5, u_6, u_7, u_8, u_9, u_{10}, u_{12}\}$

Definition 5.2.7. Reverse k^{th} Farthest Neighbour (Rk^{th}FN) query is the method used to find users in U that consider query point q as their k^{th} farthest facility point.

$$Rk^{th}FN(q) = \{u | q = k^{th}FN(u), u \in U, 1 \leq k \leq |P|\}$$

Figure 5.7 gives an example of $R2^{th}FN(q)$, where the query point is p_3. In this example, all users that consider query point p_3 as their second farthest facility point are $\{u_5, u_{10}, u_{12}\}$.

Definition 5.2.8. Group Reverse Farthest Neighbour(GRFN) query is the method to find users in U that consider k numbers of query points in set of $Q = \{q_1, q_2, ..., q_k\}$ as their k farthest facility points.

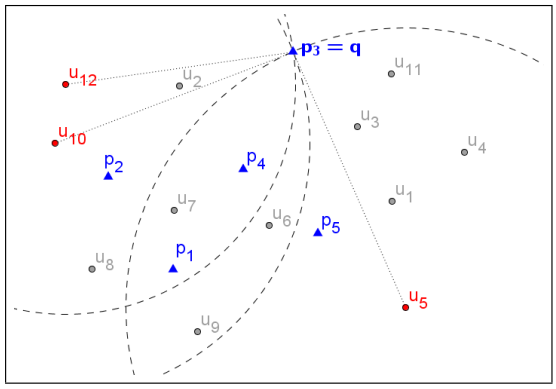


Figure 5.7: $R2^{th}FN(q)$=$\{u_5, u_{10}, u_{12}\}$

$$GRFN(Q) = \{u|Q = kFN(u), u \in U, k = |Q|\}$$

An example of GRFN is shown in Figure 5.8. In this example, the set of query point is $Q = \{p_1, p_2\}$ and since there are two query points, the value of $k = 2$. Hence, GRFN(Q) is meant to find all users U that consider the set of query point in Q as their two farthest facility points. Therefore, GRFN(Q)=$\{u_1, u_3, u_4, u_{11}\}$.

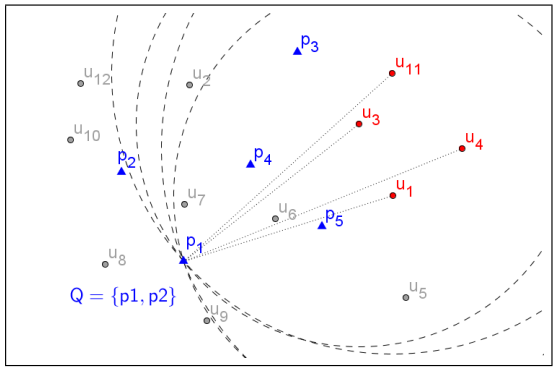


Figure 5.8: $GRFN(Q)$=$\{u_1, u_3, u_4, u_{11}\}$

5.3 Cell Level Index (CLI)

In this part, we introduce a new indexing structure for the HSVD structure called **cell level index (CLI)**. This structure has level identification for each generator point, and each level will refer to all corresponding Voronoi cells. Each Voronoi cell will refer to non-generator points objects, so when a Voronoi cell is loaded, all non-generator points objects can be identified easily. To the best of our knowledge, there are no index structures that can be used to point to the exact cells from a generator point of Voronoi diagram and depends on its position (Level) in the sequence as well, hence CLI was proposed.

Reverse nearest neighbours is a method of finding non-facility points objects that consider the query point as their nearest facility point. The RNN queries can be solved effectively by using a region-based approach (Cheema et al., 2011; Adhinugraha et al., 2013) where the aim is to find the region that contains objects to answer the query instead of checking each non-facility point one by one.

In solving reverse nearest problems using the HSVD structure, the facility points P will be considered as the generator points and users U will be considered as non-generator point objects, since the HSVD structure will be constructed based on P. A RNN query using HSVD is meant to find the Voronoi cells that contains non-generator point objects where these objects consider the query point as the nearest facility point.

Since each cell in HSVD has a unique sequence, which indicates the distance order of facility points from a certain Voronoi cell, the RNN region can also be identified by using the HSVD structure. However, in order to find appropriate cells for a particular RNN query, all cells need to be read.

In Chapter 3, the number of Voronoi cells from m facility points can be estimated using equation 3.9.3 at the maximum number of m^4. Hence by using HSVD, the number of cells that need to be read in order to process an RNN query for any value of k will be constant at m^4 cells. While this might be not an issue for higher values of k (i.e $k > (m/2)$) in RkNN query, this will be a serious problem for lower k value or Rk^{th}NN query, when the number of cells needed to solve the query is very low.

The Voronoi cells needed to answer RNN query depends on the position of the query point in the sequence of a Voronoi cell. In this thesis, the position of query point in a sequence will be referred as **'Level'**. For example, to obtain the region of $R1NN(p_1)$, we have to obtain all cells that have p_1 in *level* 1. To obtain the region of $R2NN(p_1)$, we have to obtain all cells that have p_1 in *level* 1 and 2, since $R2NN(p_1)$ means all non-facility points objects that consider p_1 as one of their two nearest facility points. We find that *level* is the main part that needs to be considered in order to determine whether a Voronoi cell can be chosen to answer an RNN query.

In this index structure, each generator point will have m level and each level of a facility point will have c Voronoi cells, where $0 \leq c < m^4$. A level of generator points might not have any cells if there is no sequence with this generator point at a certain level; hence, c can be 0. A Voronoi cell can be referred by m level of generator points, since a

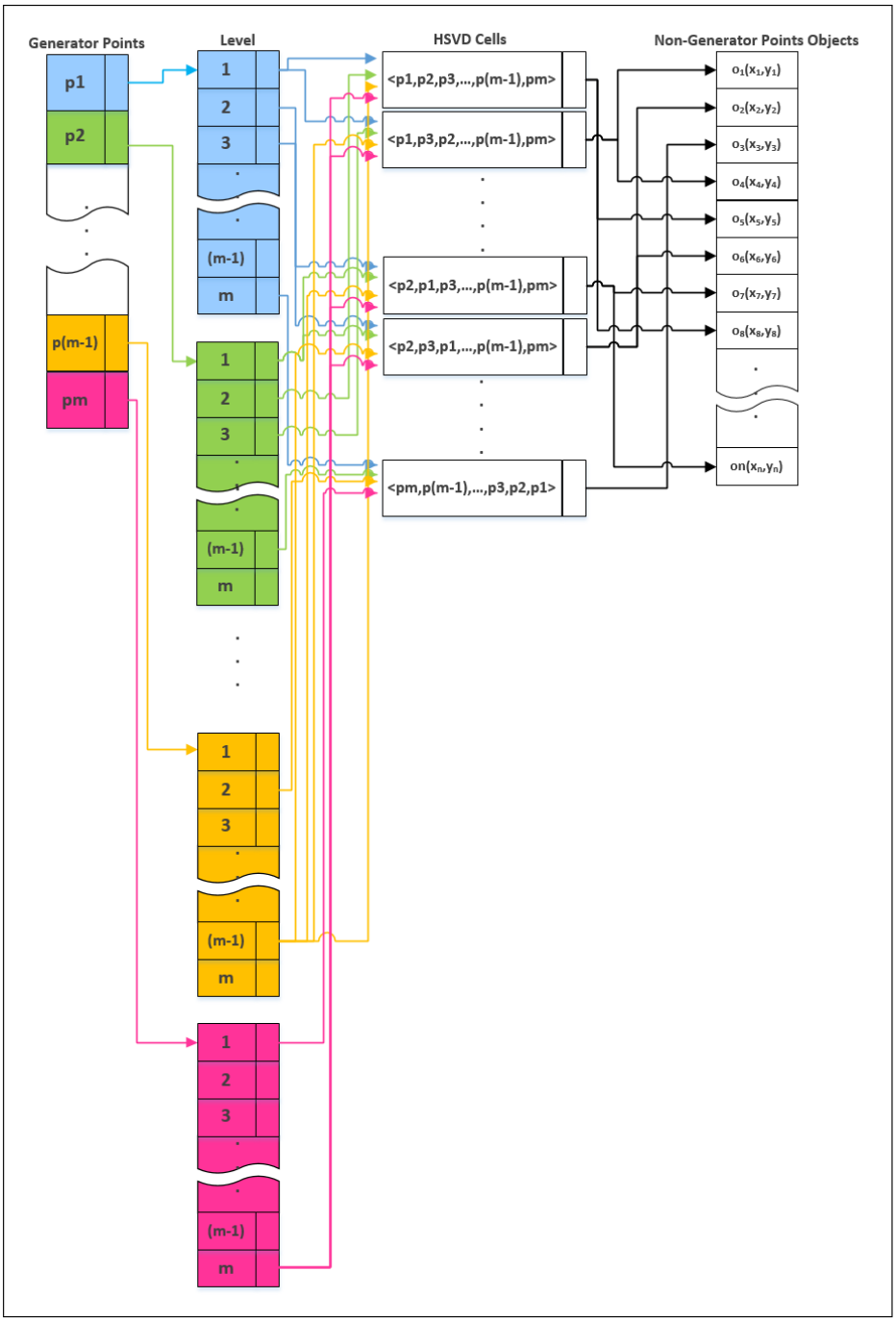
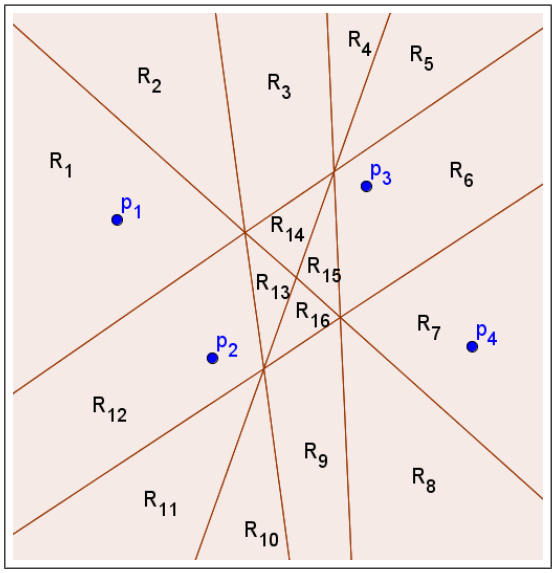


Figure 5.9: HSVD Index Structure

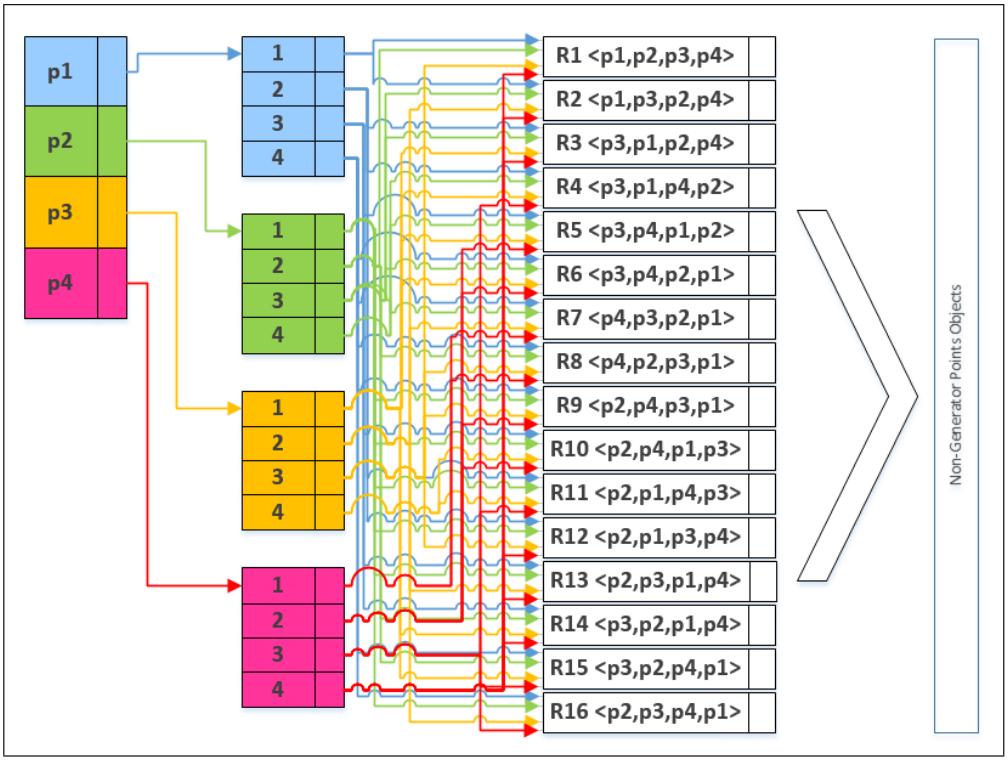
Voronoi cell in HSVD will always have a sequence from m generator points. Each cell may contain several numbers of non-generator point objects, and each object can be referred only from a single Voronoi cell, since no objects can appear in more than one Voronoi cell. The complete structure of this index is shown in Figure 5.9.

For example, a Voronoi cell with sequence $\langle p_1, p_2, p_3, p_4, p_5 \rangle$ can be referred from generator point p_1 in *level* 1, generator point p_2 in *level* 2, generator point p_3 in *level* 3, generator point p_4 in *level* 4 and generator point p_5 in *level* 5. By using CLI with the

HSVD structure, we can minimize the number of cells needed to be read in order to identify the right Voronoi cell for RNN queries.



(a) $Vh(p_1, p_2, p_3, p_4)$



(b) CLI structure of $Vh(p_1, p_2, p_3, p_4)$

Figure 5.10: Highest Order Voronoi diagram and its corresponding CLI

Figure 5.10 shows an example of HSVD constructed from four generator points (Figure 5.10a) and its corresponding CLI structure (Figure 5.10b). The number of estimated possible Voronoi cells in equation 3.9.3 from four generator points is 18 Voronoi cells, while the number of real Voronoi cells in within boundary is 16. These cells are mapped

to CLI structure and each generator point has 4 level cells because there are four generator points. Each level of generator points might or might not have the Voronoi cells. Each Voronoi cell will have a set of non-generator point objects in it, although we do not map these objects in this structure. The complete structure is shown in Figure 5.10b.

5.4 Reverse Nearest Neighbour with CLI Structure

Reverse nearest neighbour queries can be answered using either the region-based approach or point-to-point approach. The region-based approach has a significant advantage since it does not require an objects verification step to filter the candidates. In this section, we will describe the region-based approach used to solve RNN queries by using Cell-Level-Index (CLI). There are two variations of queries: (1) RkNN queries and (2) Rk^{th}NN queries.

5.4.1 Reverse k Nearest Neighbour

In this thesis, Reverse k Nearest Neighbour queries require finding non-generator point objects that consider a query point q as one of their k nearest generator point. These objects can be identified in all Voronoi cells that have query point q as the first k position in their sequences. In *a CLI* structure, all Voronoi cells that have the objects to answer the query are located in *level* 1 to *level* k.

$$RkNN(q) = \bigsqcup_{level=1}^{level=k} R(q^{[level]}) \tag{5.4.1}$$

***Algorithm 13**: RkNN on HSVD-based with CLI Structure*

Data: k, q
Result: $R = \{r_1, r_2, ..., r_n\}$ list of Voronoi cells that satisfy $RkNN(q)$

```
1  initialize(R);
2  m ← numOf(P)
3  if (k ≤ m)then
4  |  for lv = 1 to k do
5  |  |  for cl = 1 to sizeOf(q.lv.cells) do
6  |  |  |  R.add(q.lv.cells[cl]);/* load all Cells                    */
7  |  |  end
8  |  end
9  else
10 |  exit;/* k is bigger than number of facility points               */
11 end
```

Algorithm 13 explains RkNN query processing using CLI. In RkNN queries, the objective is to obtain all Voronoi cells where the query point is located in *level* k or less (line 4-8). For each level (lv), all Voronoi cells(*cells*) will be added to the result set (line 6). Since all non-generator point objects have been assigned to appropriate Voronoi cells, these objects can be identified straight from the result set.

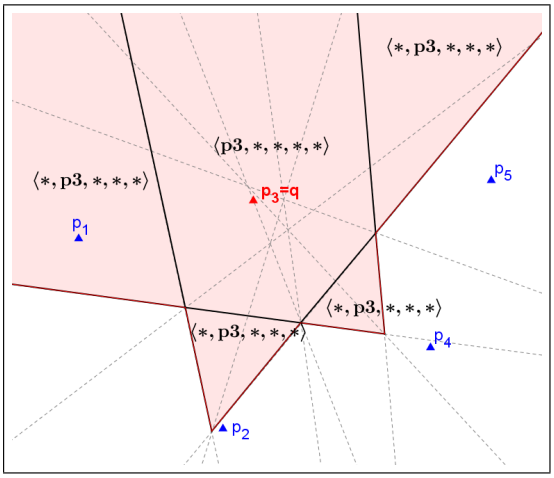


Figure 5.11: $R2NN(p_3)$ with CLI

Figure 5.11 shows how to identify the $R2NN(p_3)$ region from HSVD. The region of this query consists of all Voronoi cells where generator point p_3 is in *level* 2 or less. All Voronoi cells with p_3 in *level* 2 are denoted in $\langle *, p_3, *, *, * \rangle$ symbol, where "$*$" means any generator point which is not p_3. Meanwhile, all Voronoi cells with p_3 in *level* 1 are denoted in $\langle p_3, *, *, *, * \rangle$. From equation 5.4.1, the region for $R2NN(p_3)$ can be written as

$$R2NN(p_3) = R\langle p_3, *, *, *, * \rangle \sqcup R\langle *, p_3, *, *, * \rangle$$

Figure 5.12 shows 10 generator points and 1000 non-generator points objects distribution that will be used for the simulation. The simulation is built in a Java Net Beans environment. Figure 5.13 shows the $R3NN(p_{51})$ region from the simulation.

5.4.2 Reverse k^{th} Nearest Neighbour

Reverse k^{th} nearest neighbours is the method of finding all non-generator point objects that consider the query point as their k^{th} nearest generator point. In an HSVD, these objects are located in the region where the query point is the k^{th} position in the distance sequence. The Rk^{th}NN query can be formalized as

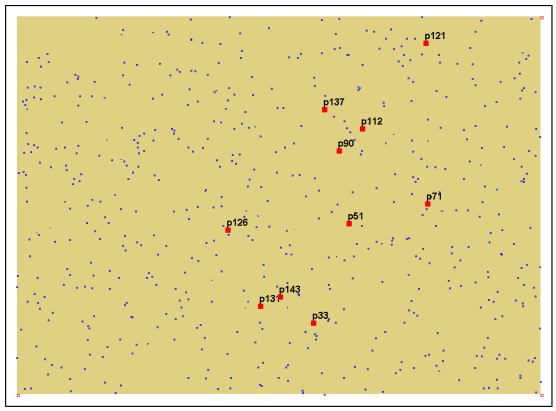


Figure 5.12: Objects distribution from simulation

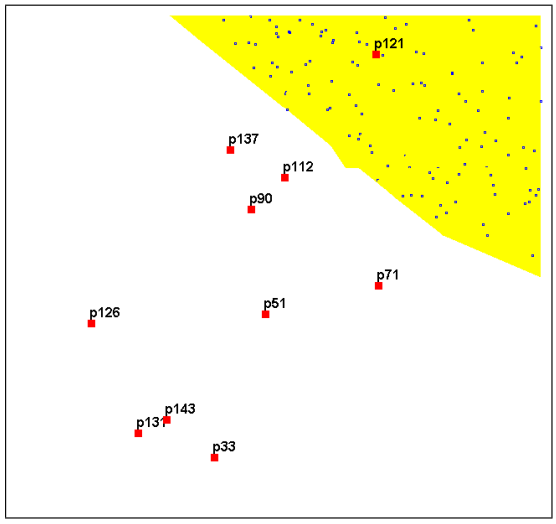


Figure 5.13: $R3NN(p51)$ Region and Objects in it from simulation

$$Rk^{th}NN(q) = R(q^{[k]})\rangle \tag{5.4.2}$$

***Algorithm 14**: RkthNN on HSVD-based with CLI Structure*

Data: k, q
Result: $R = \{r_1, r_2, ..., r_n\}$ list of Voronoi cells that satisfy $RkthNN(q)$

```
1  initialize(R);
2  lv ← k;
3  m ← numOf(P)
4  if (k ≤ m)then
5  |  for cl = 1 to sizeOf(q.lv.cells) do
6  |  |  R.add(q.lv.cells[cl]);/* load all Cells                    */
7  |  end
8  else
9  |  exit;/* k is bigger than number of facility points            */
10 end
```

Algorithm 14 shows how an Rk^{th}NN query is answered with HSVD. This algorithm is very similar with $RkNN$ algorithm, except the aim of this algorithm is to obtain only the Voronoi cells where query point q is on *level* k (line 5-7). The algorithm will stop after all Voronoi cells in this level have been loaded into the result set (R). All non-generator point objects can be identified from this result set.

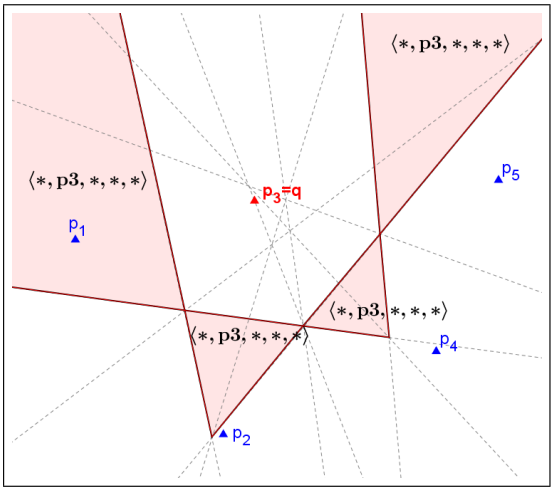


Figure 5.14: $R2^{th}NN(p_3)$ with CLI

Figure 5.14 shows an example of an $Rk^{th}NN$ query than can be solved by using HSVD structure. Assume that the HSVD is constructed from five generator points $P = \{p_1, p_2, p_3, p_4, p_5\}$, the query point is $q = p_3$ and $k = 2$. The $R2^{th}NN(p_3)$ can be defined as all Voronoi cells where p_3 at *level* 2 in the sequence. From this figure, we can identify the nine Voronoi cells needed to answer this query. Therefore, the region of $R2^{th}NN(p_3)$ can be written as

$$R2^{th}NN(p_3) = \bigsqcup R\langle *, p_3, *, *, * \rangle.$$

Another example of $Rk^{th}NN$ query processing by using the HSVD structure is shown in Figure 5.15. This figure demonstrates the region and non-generator-point objects in it from $R3^{th}NN(p_{121})$. Similar to simulation in Figure 5.13, the HSVD is constructed from 10 generator points and the number of existing non-generator points are 1000.

5.5 Reverse Farthest Neighbour with CLI Structure

Reverse farthest neighbour approach is the method used to find non-generator-point objects that consider a query point q as their farthest generator point. These objects can

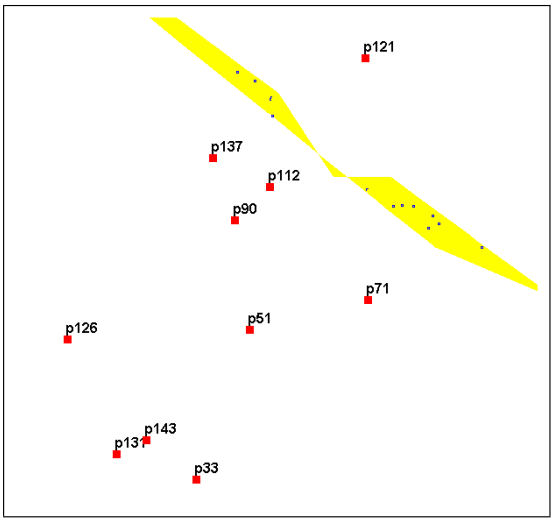


Figure 5.15: $R3^{th}NN(p121)$ Region and Objects in it from simulation

be identified in all Voronoi cells that have query point q at the last position in their sequences. In this section, we will describe the method used to identify the region as well as non-generator-point objects in it for RkFN and Rk^{th}FN queries. We will solve these problems with a CLI and HSVD structure.

5.5.1 Reverse k Farthest Neighbour

To answer RkFN queries based on the HSVD structure with *CLI* structure, all Voronoi cells that have the objects to answer the query are located in *level* m to *level* $m+1-k$.

$$RkNN(q) = \bigsqcup_{level=m+1-k}^{level=m} R(q^{[level]}) \tag{5.5.1}$$

***Algorithm 15**: RkFN on HSVD-based with CLI Structure*

Data: k, q
Result: $R = \{r_1, r_2, ..., r_n\}$ list of Voronoi cells that satisfy $RkFN(q)$

```
1  initialize(R);
2  m ← numOf(P)
3  if (k ≤ m)then
4  |  for lv = m down to (m+1-k) do
5  |  |  for cl = 1 to sizeOf(q.lv.cells) do
6  |  |  |  R.add(q.lv.cells[cl]);/* load all Cells                    */
7  |  |  end
8  |  end
9  else
10 |  exit;/* k is bigger than number of facility points               */
11 end
```

Algorithm 15 explains how to identify an $RkFN$ region from the HSVD structure. This algorithm works in a similar way to the algorithm for an $RkNN$ query, except this algorithm starts from the farthest *level* and moves towards the nearest (line 4) *level* of query point. On each *level*, all corresponding Voronoi cells will be loaded. The algorithm will stop once all Voronoi cells on each level have been loaded, and all non-generator-point objects can be referred from the selected Voronoi cells.

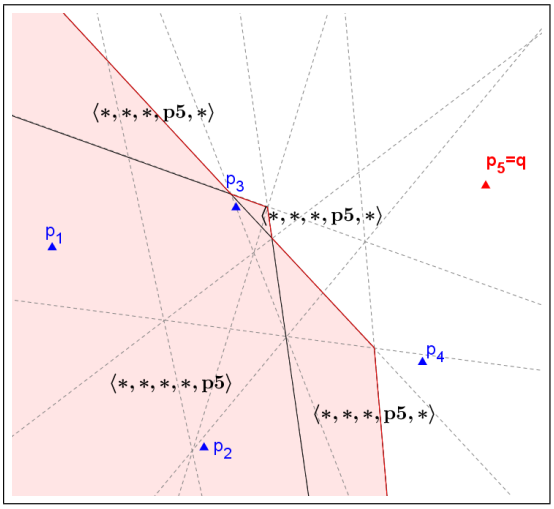


Figure 5.16: $R2FN(p_5)$

Figure 5.16 gives an example of an $RkFN$ query. Assume that the HSVD is constructed from five generator points $P = \{p_1, p_2, p_3, p_4, p_5\}$, the query point $q = p_5$ and $k = 2$. $R2FN(p_5)$ is meant to find all Voronoi cells where p_5 is between *level* 1 and *level* 2. The region of this query is the shaded area. Therefore, the region of R2FN(p_5) can be defined as

$$R2FN(p_5) = R\langle *, *, *, *, p_5\rangle \sqcup R\langle *, *, *, p_5, *\rangle$$

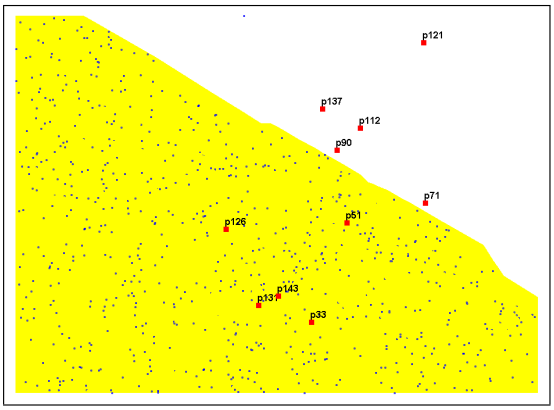


Figure 5.17: $R3FN(p121)$ Region and Objects in it from simulation

Figure 5.17 is an example of an $RkFN$ where the HSVD is generated from ten generator points, $k = 3$ and $q = p_{121}$. The number of non-generator-point objects is 1000 and these objects are distributed uniformly on the map.

5.5.2 Reverse k^{th} Farthest Neighbour

Reverse k^{th} farthest neighbour query is the method of finding the non-generator-point objects that consider the query point as their k^{th} farthest generator point. These objects can be identified in all Voronoi cells where the query point is at the last k^{th} position on the sequence. In the HSVD structure, the Voronoi cells can be identified where the query point is on *level* k^{th} from the last, or equivalent to *level* $m + 1 - k$ in $Rk^{th}NN$ query.

$$Rk^{th}FN(q) = R(q^{[m+1-k]}) \tag{5.5.2}$$

***Algorithm 16**: RkthFN on HSVD-based with CLI Structure*

```
   Data: k, q
   Result: R = {r_1, r_2, ..., r_n} list of Voronoi cells that satisfy RkthFN(q)
 1 initialize(R);
 2 m ← numOf(P)
 3 lv ← m + 1 − k;
 4 if (k ≤ m)then
 5 |  for cl = 1 to sizeOf(q.lv.cells) do
 6 |  |  R.add(q.lv.cells[cl]);/* load all Cells                    */
 7 |  end
 8 else
 9 |  exit;/* k is bigger than number of facility points           */
10 end
```

Algorithm 16 shows the process of $Rk^{th}FN$ query processing. This method chooses the Voronoi cells that have query point q at the k^{th} last *level* (line 3), and then loads all Voronoi cells on this level (line 5-7). The non-generator-point objects can be identified from the loaded Voronoi cells since the non-generator-point objects can be referred from Voronoi cells in the CLI structure.

Figure 5.18 shows an example of this query $R2^{th}FN(p_5)$. Assume that the predefined HSVD structure is constructed from five generator points, and CLI is constructed from HSVD. The region of $R2^{th}FN(p_5)$ can be identified by choosing all the Voronoi cells where query point p_5 is located on *level 4* in CLI structure. The cells that have generator point

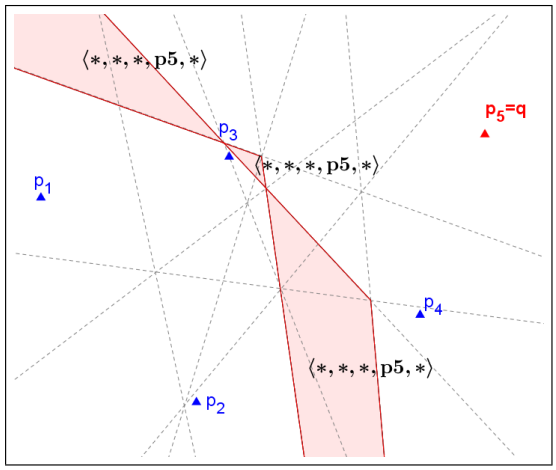


Figure 5.18: $R2^{th}FN(p_5)$

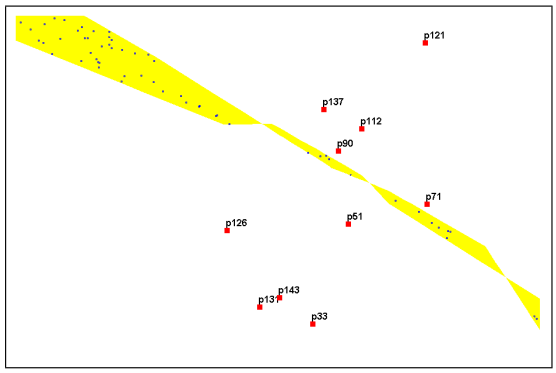


Figure 5.19: $R3^{th}FN(p131)$ Region and Objects in it from simulation

p_5 at level 4 will have the sequence pattern as $\langle *, *, *, p_5, * \rangle$, and the region of this query will be the union of all Voronoi cells with this pattern, as can be shown in the figure.

Another example of this query type is shown in Figure 5.19, where the HSVD structure is constructed from ten generator points and there are 1000 users distributed randomly. The query point is p_{131} and $k = 3$. The region of $R3^{th}FN(p_{131})$ is shaded yellow and the dots represent non-generator-point objects that consider p_{131} as their 3^{th} farthest facility point.

5.6 Group Reverse Queries

Group reverse query is a reverse query that is issued from a set of query points. The main aim of group reverse query is to find the region and non-generator-point objects in it that consider the set of query points as their nearest generator points. Group reverse query has recently attracted researchers' attention and has great potential for use in further applications (Zhu et al., 2014). We divide the group reverse queries into two categories:

(1) group reverse nearest neighbour queries (GRNN) and (2) group reverse farthest neighbour queries (GRFN). In group reverse queries, the value of k is equal to the number of query points in Q. Let $Q = \{q_1, q_2, ..., q_k\}$, then $k = |Q|$, since the region of group reverse query is the region where all non-generator-point objects consider the set of query points as their k nearest or farthest generator points.

5.6.1 Group Reverse Nearest Neighbour (GRNN)

Let $P = \{p_1, p_2, p_3, ..., p_m\}$ is a set of generator points from highest order Voronoi diagram $V(P)$. Group reverse nearest neighbour from a set of query point $Q = \{q_1, q_2, q_3, ..., q_k\} \subseteq P$ is a method of finding the region where all non-generator-point objects U will consider this set of query points as their k nearest generator points.

$$GRNN(Q) = \{u \mid Q = kNN(u), u \in U, k = |Q|, Q \subseteq P\} \tag{5.6.1}$$

When a non-generator-point object u considers a query point q as the nearest generator point, object u is part of RNN of q. When this object u also considers another generator point p as the two nearest generator point so that $2NN(u) = \{q, p\}$, this means that object u is also the reverse nearest neighbours of $\{q, p\}$. Since there are two generator points, the value of k depends on the number of generator points involved in this query.

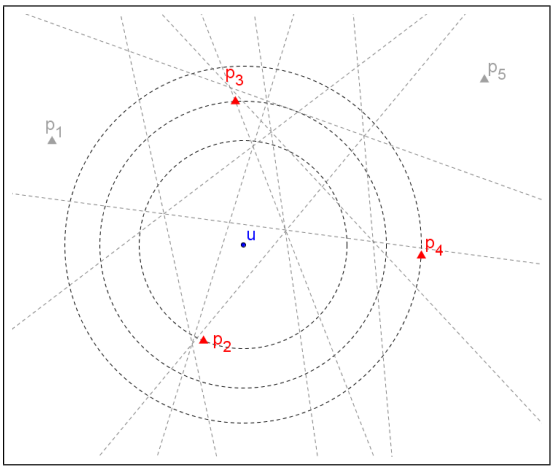


Figure 5.20: $3NN(u) = \{p_2, p_3, p_4\}$

In answering GRNN query from a set of query points Q with HSVD, the set of query points Q is a subset of all facility points $P = \{p_1, p_2, p_3, ..., p_m\}$, so $Q \subseteq P$. The highest order Voronoi diagram $Vh(P)$ is constructed from all facility points P, so that P is also

a generator point and query points in Q are also generator points. Since all query points have to be considered as the nearest generator points, the value of k will depend on the number of query points, so $k = |Q|$. Hence, in region of GRNN query, each query point will be one of k nearest generator point for any non-generator-point objects U in this region as can be seen in figure 5.20.

In Group Reverse Nearest queries, the set query point $Q = \{q_1, q_2, q_3, ..., q_k\}$ consists of multiple query points, so in order to cover all the query points as k nearest generator points, the region of GRNN query must be the subregion of any RkNN(q_i), where

$$(GRNN(Q) \sqsubseteq RkNN(q_1)) \wedge (GRNN(Q) \sqsubseteq RkNN(q_2)) \wedge ... \wedge (GRNN(Q) \sqsubseteq RkNN(q_k)) \tag{5.6.2}$$

Since the region of $GRNN(Q)$ is a subregion of all $RkNN(q_i)$, the region of $GRNN(Q)$ is constructed from the intersection of all regions $RkNN(q_i)$, where can be formulated as:

$$GRNN(Q) = RkNN(q_1) \sqcap RkNN(q_2) \sqcap ... \sqcap RkNN(q_k) \tag{5.6.3}$$

The process of identifying the region of GRNN(Q) in equation 5.6.3 with HSVD can be simplified by performing $RkNN(q_i)$ for k times, and intersecting all $RkNN$ regions.

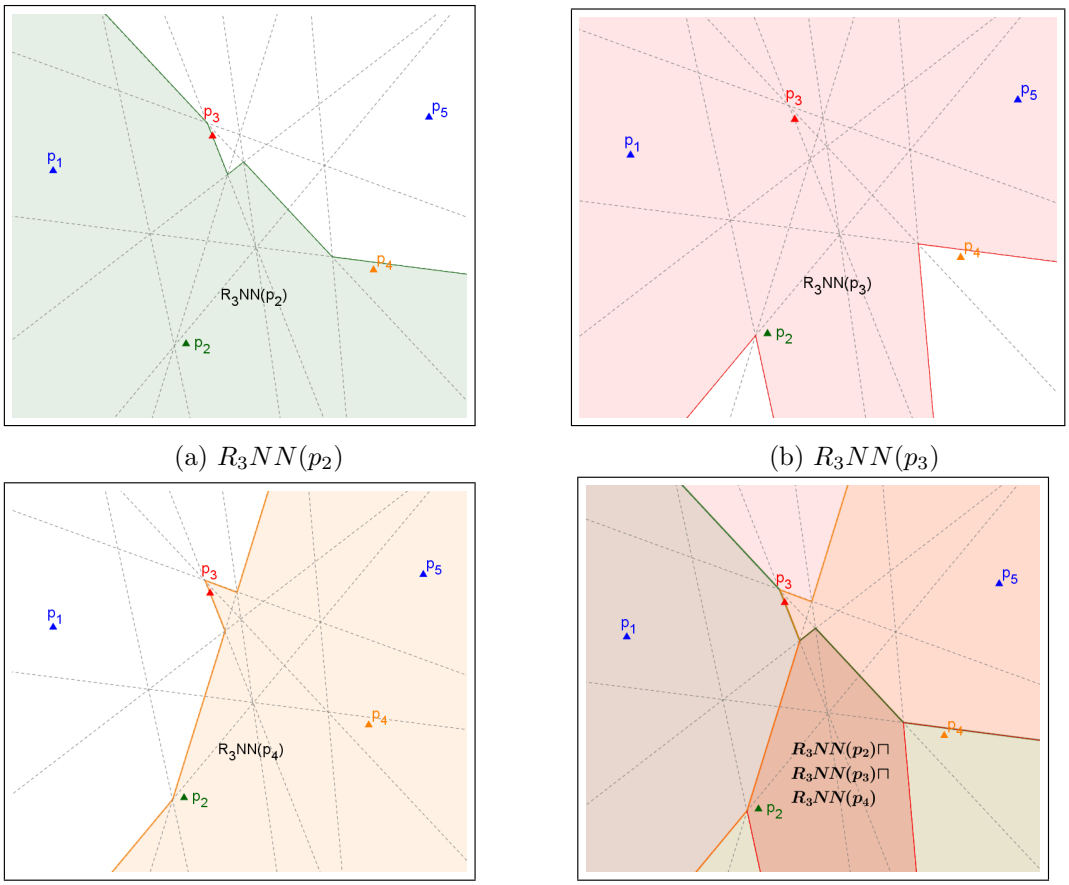
An example of this method can be seen in Figure 5.21. In this example, assume that the query point $Q = \{p_2, p_3, p_4\}$. Since the region of GRNN(Q) is the intersection of all $R3NN(p)$, to obtain this region, we have to perform $R3NN(p)$ three times (Figure 5.21a,5.21b,5.21c), and intersect all $R3NN$ regions (Figure 5.21d).

In the region of GRNN(Q), all query points must be considered as the nearest generator points, regardless of the distance sequence. Hence, the distance sequence of this region can consist of any k combinations of k query points at the first left sequence. The k combination of possible distance sequences from a set of query points Q will be $C(Q, k)$ as $\langle q_1, q_2, ..., q_k, [*]\rangle$, $\langle q_2, q_1, ..., q_k, [*]\rangle$, ..., $\langle q_k, q_{(k-1)}, ..., q_1, [*]\rangle$

From this explanation, the region of GRNN(Q) can be expressed as

$$GRNN(Q) = \bigsqcup R\langle C(Q, k)\rangle \tag{5.6.4}$$

To minimize the number of cells that need to be read in equation 5.6.3, we will only search the possible cells in the first $RkNN(q_i)$ region, where $q_i \in Q$, because



(a) $R_3NN(p_2)$ (b) $R_3NN(p_3)$

(c) $R_3NN(p_4)$ (d) $GRNN(\{p_2, p_3, p_4\})$

Figure 5.21: $GRNN(Q), Q = \{p_2, p_3, p_4\}$

$GRNN(Q) \subseteq RkNN(q_i)$. We will call this first region as *candidate region*, and then followed by refinement to get all the cells that have a combination of Q on their first k-sequences. This method is implemented using the following algorithm.

Algorithm 17 is arranged based on equation 5.6.4, where the GRNN region can be identified by the union of all Voronoi cells that have any k combination of Q as the first-k in their distance sequences. Since the GRNN region is also a subregion of any region of RkNN(q_i) as in equation 5.6.2, this algorithm starts by choosing a *candidate region* RkNN(q_1) (line 5, V). The next step is to find cells that have any k-combinations as k-first sequence (line 8-10) and choose these cells as the result set for this query. The non-generator-point objects can be referred from the regions of $GRNN(Q)$ itself.

This algorithm is illustrated in Figure 5.22. Let $P = \{p_1, p_2, p_3, p_4, p_5\}$, $Q = \{p_2, p_3, p_4\}$ and $k = 3$. The first step in finding the region of GRNN(Q) is by finding the *candidate region* R3NN(p_2) (green-shaded area). The next step is to apply a refinement process by selecting all Voronoi cells that have any k-combinations of Q as their

***Algorithm 17**: GRNN on HSVD-based with CLI Structure*

Data: $Q = \{q_1, q_2, ..., q_k\}$
Result: $R = \{r_1, r_2, ..., r_n\}$ list of Voronoi cells that satisfy $GRNN(q)$

```
1  initialize(R);
2  m ← numOf(P)
3  k ← numof(Q)
4  if (k ≤ m)then
5  |  V ← RkNN(k, Q[1]);/* get RkNN region of first query point        */
6  |  for i = 1 to sizeOf(V) do
   |  |  /* Check all available Voronoi cells                          */
7  |  |  s ← getSequence(V[i]);
8  |  |  if subSet(Q,k,s) then
   |  |  |  /* if set of Q is in first-k of s                         */
9  |  |  |  R.add(V[i]);
10 |  |  end
11 |  end
12 else
13 |  exit;/* k is bigger than number of facility points              */
14 end
```

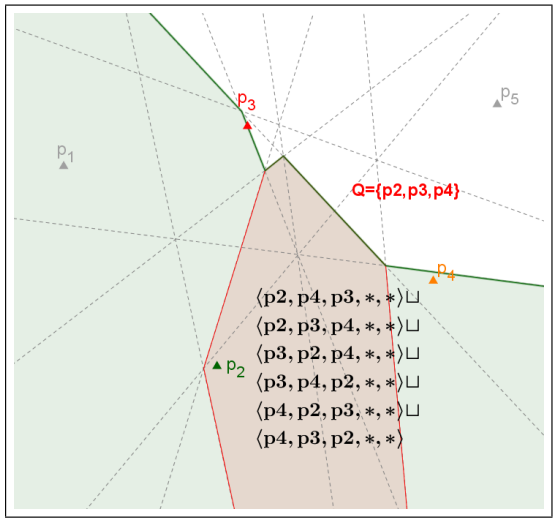


Figure 5.22: $GRNN(p_2, p_3, p_4)$

k-first sequences. Hence, the region of $GRNN(Q)$ is shown as a red-shaded area and can be formally written as

$$GRNN(p_2, p_3, p_4) = R\langle p_2, p_3, p_4, *, *\rangle \sqcup R\langle p_2, p_4, p_3, *, *\rangle \sqcup R\langle p_3, p_2, p_4, *, *\rangle \sqcup$$
$$R\langle p_3, p_4, p_2, *, *\rangle \sqcup R\langle p_4, p_2, p_3, *, *\rangle \sqcup R\langle p_4, p_3, p_2, *, *\rangle$$

Another example of a GRNN query is shown through our simulation, where the HSVD is constructed from ten generator points and 1000 non-generator-point objects are generated randomly as shown in Figure 5.23. In this figure, the query point set

$Q = \{p_{51}, p_{71}, p_{90}, p_{112}\}$ and the region of a GRNN query consists of all non-generator-point objects that consider these query points as their four nearest generator points.

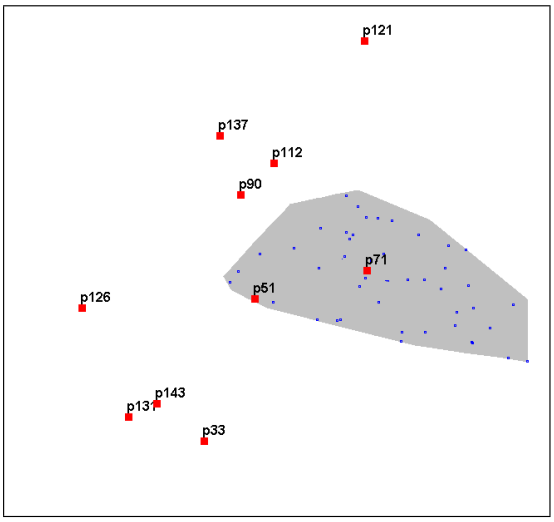


Figure 5.23: $GRNN(p_{51}, p_{71}, p_{90}, p_{112})$ Region and Objects in it from simulation

5.6.2 Group Reverse Farthest Neighbours (GRFN)

Let $P = \{p_1, p_2, p_3, ..., p_m\}$ be a set of generator points from a highest order Voronoi diagram $V(P)$. GRFN from a set of query point $Q = \{q_1, q_2, q_3, ..., q_k\}, Q \subseteq P$ is a method of finding the region where all non-generator points in U will consider this set of query points as their k farthest generator points.

$$GRFN(Q) = \{u \mid Q = kFN(u), k = |Q|, Q \subseteq P\} \tag{5.6.5}$$

Similar to the GRNN query, the value of k depends on the number of query points in Q, since all of the query points must be considered as nearest generator points from objects u. Figure 5.24 shows an example of the three farthest generator points from object u, where object u is a non-generator-point object. In this example, generator point p_3, p_4, p_5 are three farthest generator points from object u, and object u is also part of R3FN(p_3), part of R3FN(p_4) and part of R3FN(p_5).

The region of RkFN(q) contains non-generator-point objects that consider a query point as their k farthest generator point. The region of $GRFN(Q)$ contains non-generator-point objects that consider other query points beside q_1 as their k farthest generator points. Hence, a region of $GRFN(Q)$ must be a subregion of RkFN(q_i), where

$$(GRFN(Q) \sqsubseteq RkFN(q_1)) \wedge (GRFN(Q) \sqsubseteq RkFN(q_2)) \wedge ... \wedge (GRFN(Q) \sqsubseteq RkFN(q_k)) \tag{5.6.6}$$

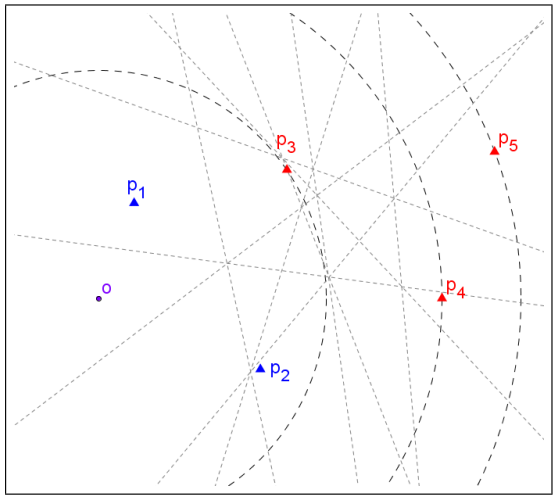


Figure 5.24: $3FN(o) = \{p_3, p_4, p_5\}$

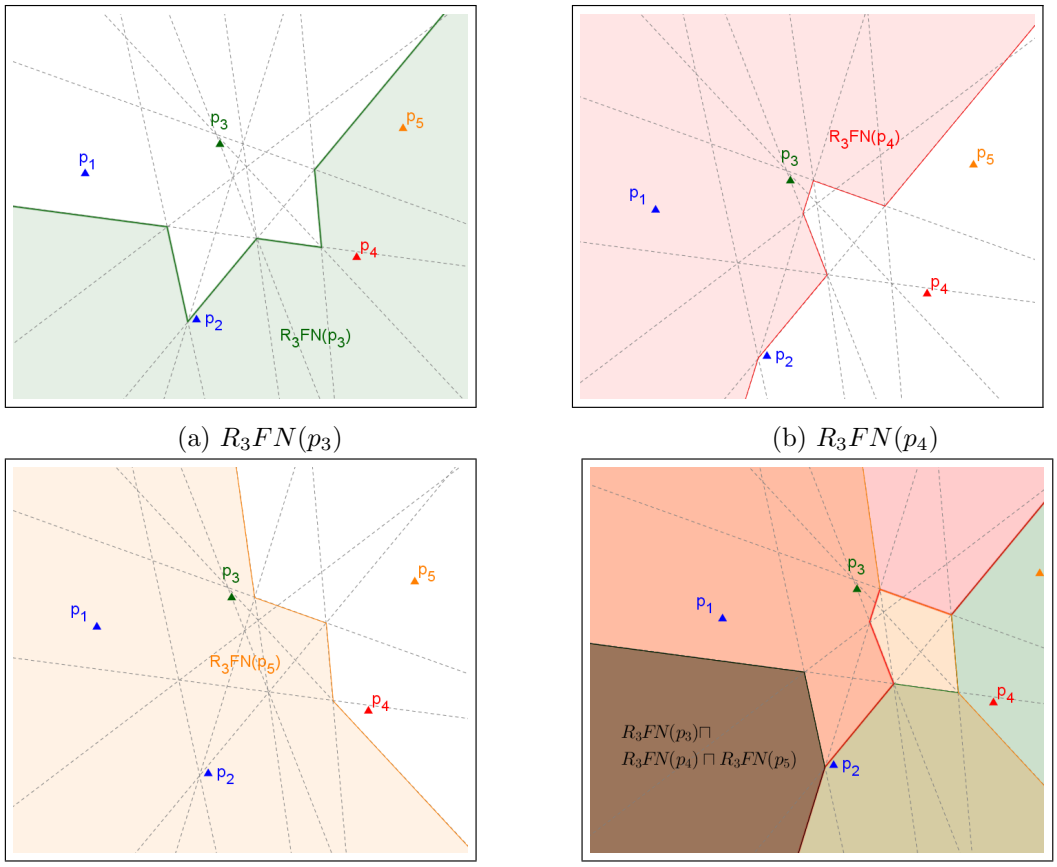
Since the region of $GRFN(Q)$ is a subregion of the entire region of RkFN(q_i), the region of $GRNN(Q)$ can be constructed from intersection of region of $R_kFN(q_i)$ that can be expressed as:

$$GRFN(Q) = RkFN(q_1) \sqcap RkFN(q_2) \sqcap ... \sqcap RkFN(q_k) \tag{5.6.7}$$

In this method, the region of GRFN(Q) can be identified by performing RkFN(q_i) k-times and intersecting with all $RkFN$ regions to obtain the GRFN(Q) region.

An example of the process used to identify the region of GRFN in equation 5.6.7 with HSVD can be seen in Figure 5.25. In this example, the HSVD is constructed from $P = \{p_1, p_2, p_3, p_4, p_5\}$. The set of query points $Q = \{p_3, p_4, p_5\}$ and $k = 3$. The region of $GRFN(Q)$ is defined as the intersection of all regions of $R3FN$, so that $GRFN(Q) = R3FN(p_3) \sqcap R3FN(p_4) \sqcap R3FN(p_5)$.

In identifying the region of a GRFN query with HSVD, it is important to minimize the number of cells that need to be read. The region of GRFN in equation 5.6.7 will involve too many unnecessary cells, since each query point will perform a RkFN query and obtain its own region before the final region can be obtained. Since the final GRFN(Q) region is always a subregion of each region of RkFN(q_i), the process of identifying GRFN region



(a) $R_3FN(p_3)$ (b) $R_3FN(p_4)$

(c) $R_3FN(p_5)$ (d) $GRFN(\{p_3, p_4, p_5\})$

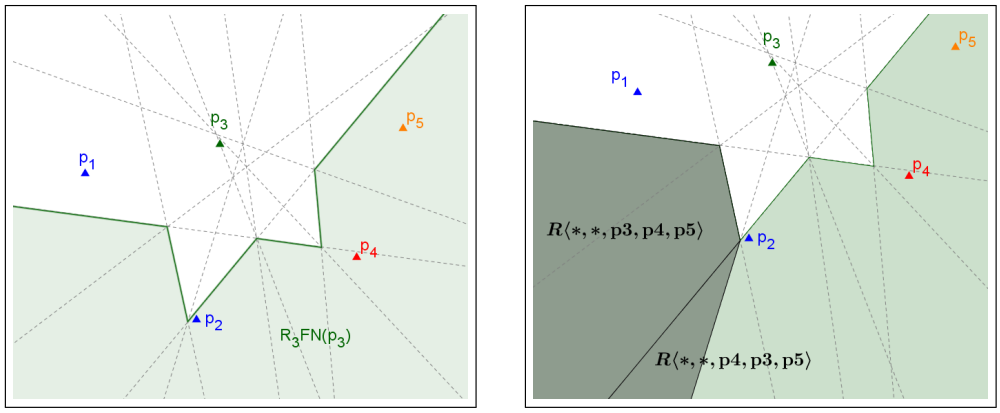
Figure 5.25: $GRFN(Q), Q = \{p_3, p_4, p_5\}$

can be initially focused on a *candidate region* of RkFN(q_i), and the refinement process will be done within this region. An example of this process can be seen in Figure 5.26. The initial step is shown in figure 5.26a, where an initial region is set from a RkNN query. The next step is to refine and prune any unnecessary cells so that the region of GRFN(Q) can be identified. At the end, the region of GRFN is the union of all Voronoi cells that have combination Q as the last k-sequence (Figure 5.27), and can be expressed as

$$GRFN(Q) = \bigsqcup R\ inverse(\langle C(Q,k)\rangle) \tag{5.6.8}$$

From this equation, we can define an algorithm to identify the region of GRFN, based on HSVD and CLI structure. This algorithm receives Q as the input, and produces the region for GRFN(Q).

Algorithm 18 explains how to identify the region of GRFN by using the HSVD and CLI structures. The first step is to identify the initial region that can be obtained from a region of RkFN(q_1) (line 5). The next step is to refine the region from line 5 so that only



(a) Initial (b) Refinement

Figure 5.26: $GRFN(Q), Q = \{p_3, p_4, p_5\}$

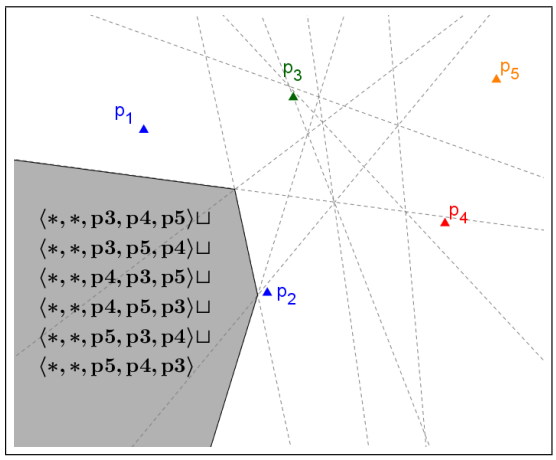


Figure 5.27: $GRFN(p_3, p_4, p_5)$

those Voronoi cells that have Q in the last k sequences will be chosen (line 9-11). To get the last k in the same manner as RkNN process, we apply $inverseSequence()$ function to reverse the sequences. The algorithm will stop after the initial region has been refined.

By using this algorithm, the region of GRFN(Q) in Figure 5.27 can be formulated as

$$GRFN(p_3, p_4, p_5) = R\langle *, *, p_3, p_4, p_5\rangle \sqcup R\langle *, *, p_3, p_5, p_4\rangle \sqcup R\langle *, *, p_4, p_3, p_5\rangle \sqcup$$
$$R\langle *, *, p_4, p_5, p_3\rangle \sqcup R\langle *, *, p_5, p_3, p_4\rangle \sqcup R\langle *, *, p_5, p_4, p_3\rangle$$

Another example of a GRFN query is shown through our simulation, where the HSVD is constructed from ten generator points and 1000 non-generator-point objects are generated randomly as shown in Figure 5.28. In this figure, the query point set $Q = \{p_{71}, p_{121}, p_{137}\}$ and the region of a GRFN query consists of all non-generator-point objects that consider these query points as their three farthest generator points.

***Algorithm 18**: GRFN on HSVD-based with CLI Structure*

Data: $Q = \{q_1, q_2, ..., q_k\}$
Result: $R = \{r_1, r_2, ..., r_n\}$ list of Voronoi cells that satisfy $GRFN(q)$

```
1  initialize(R);
2  m ← numOf(P)
3  k ← numof(Q)
4  if (k ≤ m)then
5  |  V ← RkFN(k, Q[1]);/* get RkFN region of first query point        */
6  |  for i = 1 to sizeOf(V) do
   |  |  /* Check all available Voronoi cells                          */
7  |  |  s ← getSequence(V[i]);
8  |  |  s ← inverseSequence(s) /* Inverse the order of sequence       */
9  |  |  if subSet(Q,k,s) then
   |  |  |  /* if set of Q is in first k of s                         */
10 |  |  |  R.add(V[i]);
11 |  |  end
12 |  end
13 else
14 |  exit;/* k is bigger than number of facility points              */
15 end
```

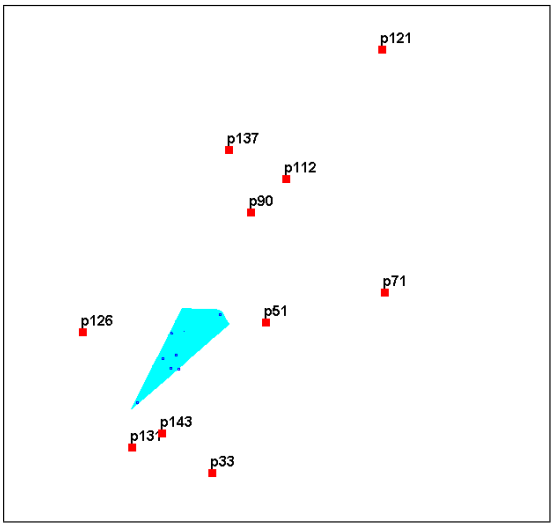


Figure 5.28: $GRFN(p71, p121, p137)$ Regions and Objects in it from simulation

5.7 Discussion

Spatial reverse queries can be classified by the way objects are sorted according to the distance, either from the nearest to the farthest or from the farthest to the nearest. Reverse k nearest neighbour is the method of finding the regions and users in those regions that consider a facility point as one of their k nearest facility points, and this method starts the search process from the nearest to the farthest. Reverse k farthest neighbours on the other hand, is the method of finding the regions and users in those regions that consider the

facility point as one of their k farthest facility points, and this method starts the searching process from the farthest to the nearest.

In this chapter, we utilize a highest order Voronoi diagram to solve spatial reverse queries. This diagram has an important spatial feature: sequence information to all generator points that can be used to solve both RkNN and RkFN queries. However, the HSVD structure does not provide an efficient retrieval structure. Therefore, this diagram must be used in a sequential manner to answer any reverse queries. To overcome this problem, we propose an index structure for HSVD that provides an effective retrieval structure for reverse queries called **cell level index (CLI)**. This index is built to identify cells that have a generator point at a certain level. This index can easily obtain the cells needed for any k values in all reverse query variations including RNN, RFN, GRNN and GRFN query processing.

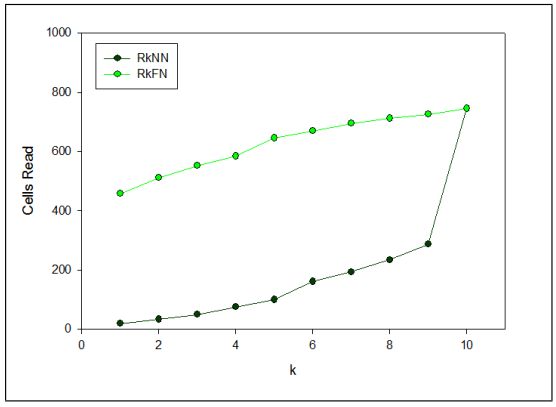


Figure 5.29: Cells read for RkNN and RkFN with CLI

Figure 5.29 shows the number of cells read for RkNN and RkFN. In this figure, the number of cells read is also the number of cells needed to answer the query since the CLI index refers to the correct cells directly so that no unnecessary cells are read on the CLI index.

Meanwhile, Figure 5.30 shows the number of cells that need to be read for an $Rk^{th}NN$ query and an $Rk^{th}FN$ query by using the CLI structure. The number of cells represents the number of cells needed to answer the query for a specific value of k, and no unnecessary reading is performed.

From our experiments, it is clear that the CLI can perform better in identifying the region of spatial reverse query since this structure can identify the cells for any generator points on any level.

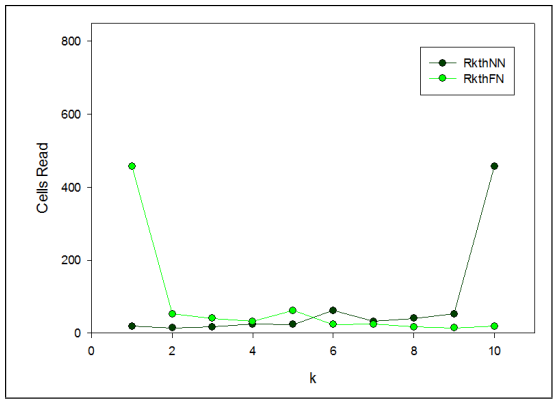


Figure 5.30: Cells read for RkthNN and RkthFN with CLI

5.8 Chapter Summary

In this chapter, we show how to answer the reverse queries in spatial databases and their variations using the HOVD structure. Since HSVD does not have a retrieval structure, in this chapter we propose a retrieval index to avoid reading all cells called, **cell level index (CLI)**. We applied this index structure to solve common spatial reverse queries and their variations, which are RkNN, RkFN, Rk^{th}NN, Rk^{th}FN, GRNN and GRFN queries.

Chapter 6

Polychromatic Spatial Queries with HSVD

6.1 Overview

In spatial query processing, there are two common categories based on the number of objects types involved in the query, named *monochromatic* and *bichromatic* queries. In a monochromatic query, both query point and the result from the query use the same type of object. In a bichromatic query, the object that issues the query is different from the result sets. Both monochromatic and bichromatic queries have been intensively discussed in Chapter 2, Chapter 4 and Chapter 5.

In this chapter, we will discuss a region-based **polychromatic query** processing, where the number of object types involved in the spatial queries is greater than two. Polychromatic query is very common in our daily lives, although this type of query has not attracted substantial research (Zhao et al., 2009). The worst case for polychromatic approach is when the number of object types involved in the query is the same as the number of available objects.

Consider the following everyday example in nearest neighbours problems. Assume that a student needs to find the nearest restaurant in the area of Monash University in Clayton. However, he also needs to find nearby petrol station, as shown in Figure 6.1. In this example, the student expects to obtain a list of the nearest restaurants and also the nearest petrol stations around his current position so that he can determine which way he should go.

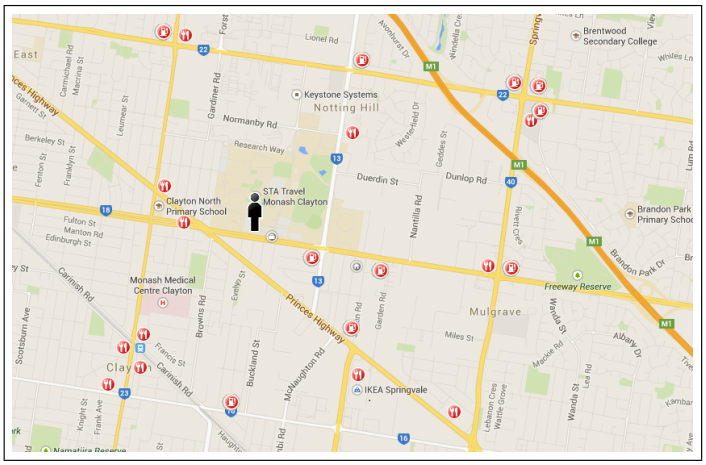


Figure 6.1: Polychromatic query with travel distance priority

Consider another example of polychromatic query in reverse nearest neighbours problems. Assume that there are several facility types related to educational facilities, which are Primary School (PS), Public Library (PL), Hospital (H) and users (U). PS, PL and H are all facility points which are needed by users U who lived in a particular residential area. Examples polychromatic queries that can be issued are: "Which Primary Schools and users consider a Public Library as the nearest library?", "Which region/residential area that consider Primary School ps, Public Library pl and Hospital h as their 3 nearest facility points?"

Polychromatic queries can also be extended into *hierarchical queries*, where the facility points can be classified into classes/subclasses, and the query can be issued according desired class/subclass. Consider another daily activity related to hierarchical queries. Assume that there are various types of restaurants in a suburb. The restaurants can be classified as "European" and "Asian" restaurants, where European restaurants can be classified more specifically as "Italian","French" and "Greek" restaurants, while Asian restaurants can be classified more specifically as "Chinese","Indian", and "Japanese" restaurants. A customer is looking for $five$ restaurants nearest to his current location. If the customer does not specify a particular type of restaurant, the result of the query will be any five restaurants close to his location. If he specifies five Asian restaurants, the result of the query could be five Chinese, Indian or Japanese restaurants nearest to his current location. Furthermore, the customer can also change the query to obtain a more specific result, such as finding five Japanese restaurants around his current location. *Hierarchical query* is very useful when a user needs to refine a query to obtain more specific results.

There are many possible applications of polychromatic queries, not limited to a spatial area. However, in this chapter we will limit our discussion to nearest neighbour queries and reverse queries in spatial databases. In this chapter, we contribute to answering polychromatic spatial queries in the following ways. We:

1. Define the taxonomy of a polychromatic query for spatial databases.
2. Propose methods for solving polychromatic queries by using HSVD.
3. Propose the *hierarchical queries* for the concept of polychromatic queries.
4. Propose methods to solve the *hierarchical queries* in polychromatic queries.

6.2 Queries Taxonomy, Notations and Definitions

6.2.1 Queries Taxonomy

In this chapter, we define the taxonomy of polychromatic queries based on how the generator point types are classified. In polychromatic queries, the facility types are not grouped while in hierarchical queries, the facility types are grouped into certain groups, and each group may form a larger group. The queries must be processed differently to accommodate the needs of each type.

In polychromatic queries, query processing depends on the nature of the query itself. In this chapter, the polychromatic nearest neighbour and polychromatic farthest neighbour queries will be processed based on nearest neighbours framework, while polychromatic reverse query processing will read all cells since the CLI index cannot be applied in this situation.

The taxonomy of polychromatic queries is described as follows:

1. Polychromatic Queries
   (a) Polychromatic Nearest Neighbours Queries
      i. Polychromatic k Nearest Neighbours Queries
      ii. Polychromatic all-k Nearest Neighbours Queries
   (b) Polychromatic Farthest Neighbours Queries
      i. Polychromatic k Farthest Neighbours Queries
      ii. Polychromatic all-k Farthest Neighbours Queries
   (c) Polychromatic Reverse Queries

i. Polychromatic Reverse k Nearest Neighbours Queries

ii. Polychromatic Reverse k Farthest Neighbours Queries

2. Hierarchical Queries
   (a) Hierarchical k Nearest Neighbours Queries
   (b) Hierarchical k Farthest Neighbours Queries
   (c) Hierarchical Reverse k Nearest Neighbours Queries
   (d) Hierarchical Reverse k Farthest Neighbours Queries

In hierarchical queries, both hierarchical nearest neighbour and hierarchical farthest neighbours will still apply the nearest neighbours framework. However, in hierarchical RNN and hierarchical RFN, the CLI index cannot be used as well. Hence, all cells need to be read in order to solve these problems.

6.2.2 Notations

- $T = \{t_1, t_2, ..., t_i\}$ is the list of available facility types.
- $P(t_i) = \{p_1^{ti}, p_2^{ti}, ..., p_x^{ti}\}$ is the set of facility points type t_i.
- $P = \bigcup_1^i |P(t_i)|$ is the set of available facility points.
- $Vh(P)$ is Highest Order Voronoi diagram from P.
- m is the number of facility points. $m = \sum_1^i |P(t_i)|$.
- $U = \{u_1, u_2, ..., u_j\}$ is set of **users** that need the service from facility points P.
- q is query point. $q \in P$ or $q \in U$
- $T_q = \{Tq_1, Tq_2, ..., Tq_y\}, T_q \subseteq T$ be the set of **facilities type query**.
- $A = \{a_1^{Tq1}, a_2^{Tq1}, ...a_k^{Tq_1}, a_1^{Tq_2}, a_2^{Tq_2}, ..., a_k^{Tq_2}, ..., a_1^{Tq_y}, ..., a_k^{Tq_y}\}$, $a^{Tq_y} \in P(t_y)$ is the answer for polychromatic query.
- $seq = \langle p_1, p_2, ..., p_m \rangle$ is a distance sequence of facility points
- $R_{seq} = R\langle p_1, p_2, ..., p_m \rangle$ is a region/Voronoi cell where p_1 is the nearest facility point followed by $p_2, p_3, ...$ until p_m as the farthest.
- $seq[j]$ is a facility point at position of j from a distance sequence. $0 \leq j < (m-1)$.
- k is the number of objects of interest which query node q wants to retrieve.
- first k-sequence is the first k numbers of generator points from a seq.

  If $seq = \langle p_5, p_4, p_2, p_3, p_1 \rangle$, the first two sequence is $\langle p_5, p_4 \rangle$
- last k-sequence is the last k numbers of generator points from a seq.

  If $seq = \langle p_5, p_4, p_2, p_3, p_1 \rangle$, the last two sequence is $\langle p_1, p_3 \rangle$

6.2.3 Definitions

In the context of polychromatic spatial queries, the facility points are classified into several different types, and the query is issued according to the expected types that need to be retrieved. Let $T = \{t_1, t_2, ..., t_i\}$ be the list of available facility types, $P(t_i) = \{p_1^{ti}, p_2^{ti}, ..., p_x^{ti}\}$ be the set of facility points type t_i. The number of available facility points from all sets can be defined as $m = \sum_1^i |P(t_i)|$. The users that need the service from all facility points is $U = \{u_1, u_2, ..., u_j\}$. In a polychromatic query, since a type of facility point can also provide service to other types of facility points, the queries can be issued either from users in U or facility points in $P(t_i)$.

The answer to a polychromatic query is the set of k facility points A from expected facility types T_q. Let $T_q = \{Tq_1, Tq_2, ..., Tq_y\}, T_q \subseteq T$ be the set of expected facilities type, and $A = \{a_1^{Tq1}, a_2^{Tq1}, ...a_k^{Tq_1}, a_1^{Tq_2}, a_2^{Tq_2}, ..., a_k^{Tq_2}, ..., a_1^{Tq_y}, ..., a_k^{Tq_y}\}$, $a^{Tq_y} \in P(t_y)$. The query in polychromatic queries can be issued either from a facility point or non-facility point, $q \in U$ or $q \in P(t_i)$, depending on the type of the queries, either Nearest Neighbours queries or Reverse Nearest Neighbours queries. The following definitions explain the possible variations in polychromatic queries in this thesis. However, these variations may be extended to support other purposes.

Definition 6.2.1. Polychromatic k Nearest Neighbours (PkNN(q, k, T_q)) query is the method of finding k nearest facilities for a set of facility types T_q from a query point. The query is issued from user $q \in U$.

Figure 6.2 shows an example of a polychromatic query from query point q. Assume that there are three facility types, which are blue circle, green triangle and red diamond. User q issues a query which intended to find the two nearest facility points from either green triangle or red diamond types. This query is expected to retrieve 2 nearest facility points from q, either from the green triangle type or the red diamond. In this example, this query retrieved 1 red diamond and 1 green triangle.

Definition 6.2.2. Polychromatic all-k Nearest Neighbours (Pall-kNN(q, k, T_q)) query is the method used to find k nearest facilities for each facility type in T_q from query point q where $q \in U$.

Figure 6.3 shows an example of $Pall - kNN(q, 2, T_q)$ where the query is issued from a user q to obtain k nearest facility points for each type. Assume that there are three facility

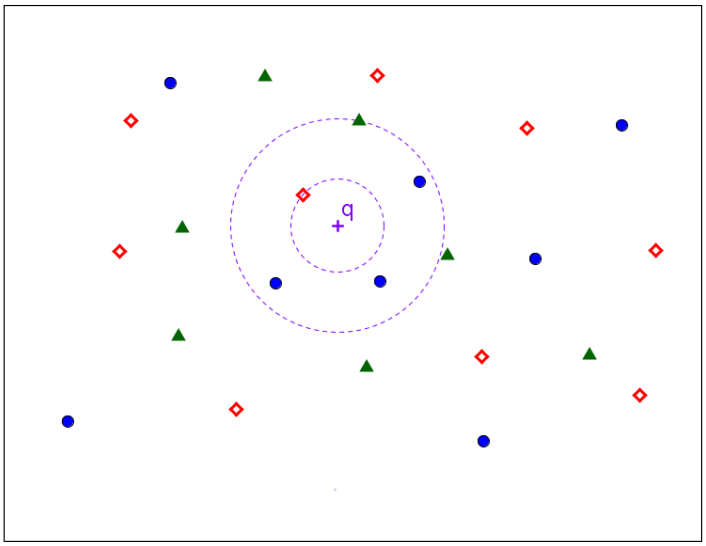


Figure 6.2: $PkNN(q, 2, T_q)$

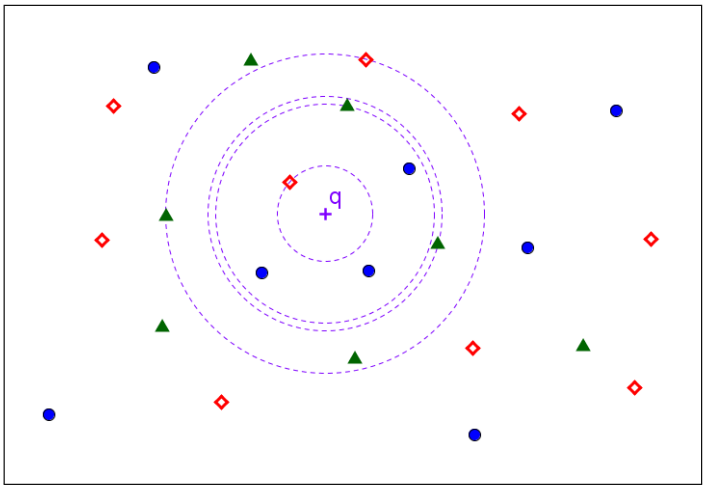


Figure 6.3: $Pall - kNN(q, 2, T_q)$

types, which are blue circle, red diamond and green triangle. This query is intended to find two nearest red-diamond facility points and two nearest green-triangle facility points around q, so a total of four facility points need to be retrieved.

In short, polychromatic kNN is a query that can be issued from either generator points or non-generator points, and this query will consider a set of generator points as the answer.

Definition 6.2.3. Polychromatic k Farthest Neighbours (PkFN(q, k, T_q)) query is the method of finding k farthest facilities for a set of facility types T_q from a query point. The query is issued by user $q \in U$.

Figure 6.5 shows an example of $PkFN(q, 2, T_q)$. Assume that there are three types of facility points which are blue circle, green triangle and red diamond. The query is issued

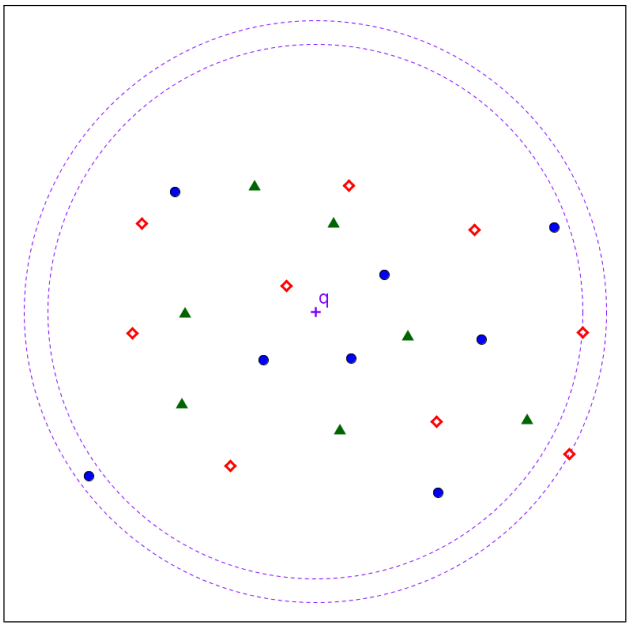


Figure 6.4: $PkFN(q, 2, T_q)$

by a user q to find the two farthest facility points from either the red diamond type or green triangle type. In this example, the query retrieves two red diamond facility points as the two farthest facility points from q, and no green triangles are selected.

Definition 6.2.4. Polychromatic all-k Farthest Neighbours (Pall-kFN(q, k, T_q)) query is the method used to find k farthest facilities for each facility type in T_q from query point q where $q \in U$.

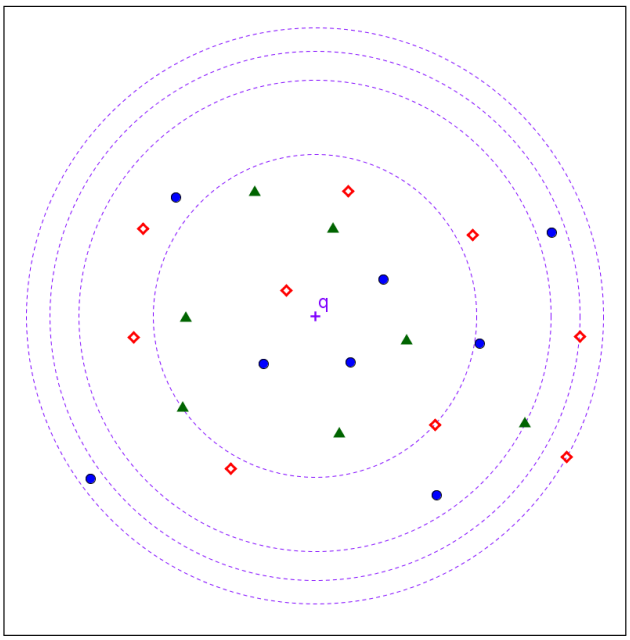


Figure 6.5: $P\ all - kFN(q, 2, T_q)$

Figure 6.5 shows an example of $Pall - kFN(q, 2, T_q)$. Assume that there are three types of facility points which are blue circle, green triangle and red diamond. The query is issued by user q to find the two farthest facility points for each facility type. In this example, this query is expected to retrieve two red diamond facility points and two green triangle facility points.

Definition 6.2.5. Polychromatic Reverse k Nearest Neighbours (PRkNN(q, k, T_q)) is the method of finding the region and users in it that consider query point q as one of their polychromatic k nearest neighbours $PkNN(u, k, T_q)$. In this query, the query is issued from a query point q where $q \in P(T_{qi]})$ and $k \geq |T_q|$.

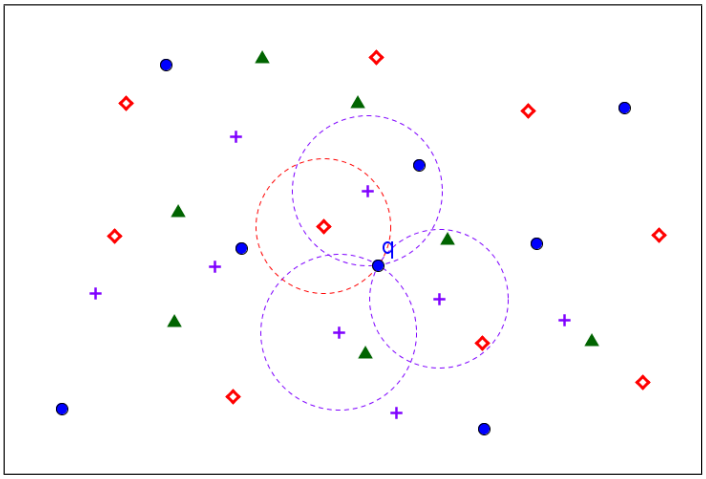


Figure 6.6: $PRkNN(q, 2, T_q)$

Figure 6.6 shows an example of a PRkNN query. In this example, assume that there are three types of facility points which are blue circle (BC), green triangle (GT) and red diamond (RD), and users are represented by purple crosses. A blue circle facility point q issues a polychromatic reverse query to identify the users that consider the query point q as their two nearest facility points compared with other blue circles or green triangles, but ignores red diamonds. Hence, $T_q = \{BC, GT\}$. In this example, the query returns three users that consider query points q as one of their $PkNN(u, 2, T_q)$.

Definition 6.2.6. Polychromatic Reverse k Farthest Neighbours (PRkFN(q, k, T_q)) is the method of finding the region and users in it that consider query point q as one of their polychromatic k farthest neighbours $PkNN(u, k, T_q)$. In this query, the query is issued from a query point q where $q \in P(T_{qi]})$ and $k \geq |T_q|$.

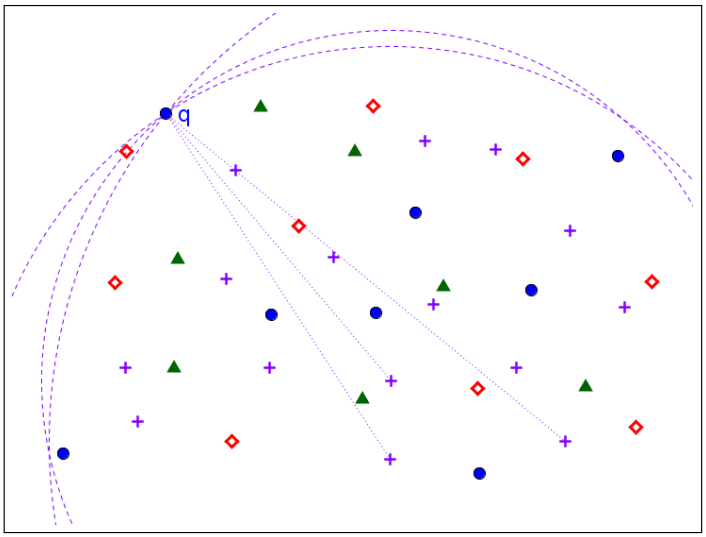


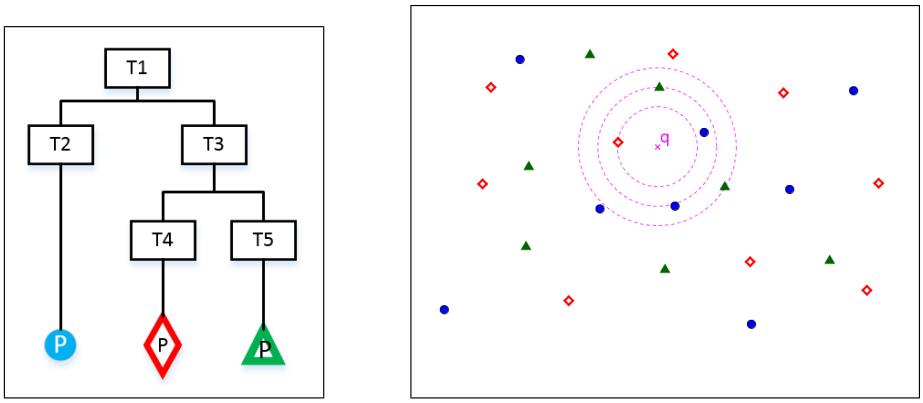
Figure 6.7: $PRkFN(q, 2, T_q)$

Figure 6.7 shows an example of a PRkFN query. In this example, assume that there are three types of facility points which are blue circle (BC), green triangle (GT) and red diamonds (RD), and users are represented by purple crosses. A blue circle issues a query $PRkFN(q, 2, T_q), T_q = \{BC, RD)\}$. This query is intended to find all users that consider this query point as one of their two nearest facility points either BC or RD, while GT is excluded.

Definition 6.2.7. Hierarchical k Nearest Neighbours Query(HkNN(q, k, T_q)) is a method to find k nearest facility points type T_q or all other subtypes of T_q, where the query is issued from a user $q \in U$.

In this type of query, the types of facility points are constructed in a hierarchical manner, so that a type can have a parent type and also sub-types, and the hierarchical structure is not a balanced structure.

Figure 6.8 shows an example of hierarchical k nearest neighbour query. Let $T1$ is a general type of facility points, where $T1$ has 2 more specific types, named $T2$ and $T3$. Blue circles are all facility points type $T2$. Meanwhile facility type $T3$ has more specific types named $T4$ and $T5$, where red diamonds are facility points type $T4$ and green triangles are facility points type $T5$ as shown in Figure 6.8a. Figure 6.8b shows an example of $HkNN(q, 3, T3)$. Assume that the query is issued by user $q \in U$, and the query is intended to find three nearest facility points type $T3$ from q. Since $T3$ is a parent for $T4$ and $T5$, this query will perform polychromatic kNN query for facility types $T4$ and $T5$.



(a) Hierarchy of Facility Types (b) $HkNN(q, 3, T3), q \in U$

Figure 6.8: $HkNN$ query example

Definition 6.2.8. Hierarchical k Farthest Neighbours Query(HkFN(q, k, T_q)) is a method used to find k farthest facility points type T_q or all other subtypes of T_q, where the query is issued from a user $q \in U$.

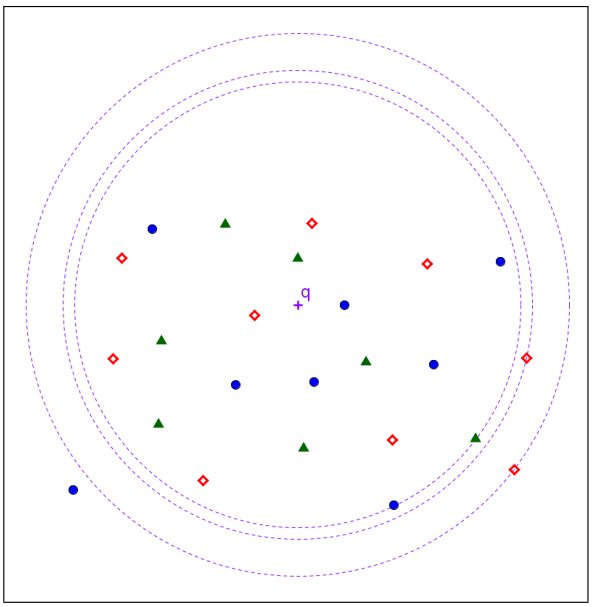


Figure 6.9: $HkFN(q, 3, T3), q \in U$

Figure 6.9 shows an example of a HkFN query. This query uses the hierarchical structure as shown in Figure 6.8a. In this example, a user $q \in U$ issues $HkFN(q, 3, T3)$ and this query is intended to find the three farthest facility points type $T3$ from q. Since $T3$ is a parent for $T4$ and $T5$, this query will also perform polychromatic kFN query for facility types $T4$ and $T5$.

Definition 6.2.9. Hierarchical Reverse k Nearest Neighbour Query(HRkNN(q, k, T_q)) is a method used find the region and users in it that consider query point q as one of their

k nearest facility points of subtypes T_q. This query is issued from a facility point q of type T_q.

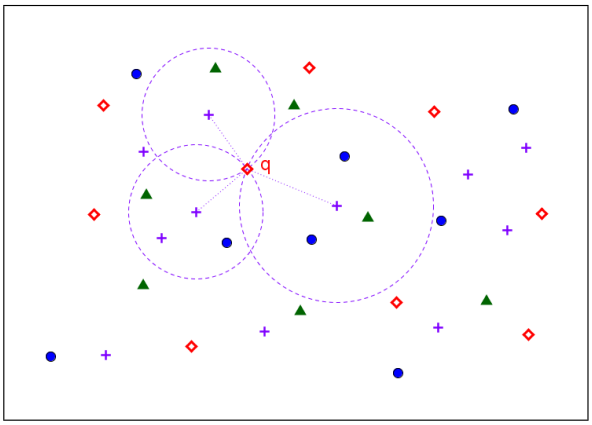


Figure 6.10: $HRkNN(q, 2, T3), q \in U$

Figure 6.10 shows an example of HRkNN query. This query uses the same hierarchical structure as that shown in Figure 6.8a. In this example, the query is issued from a red diamond facility point, and this query is intended to find the users that consider query point q as one of their two nearest T3 facility points. Since $T3 = \{RD, GT\}$, the query will ignore BC as the competitors. Hence, only users that consider query point q as one of their hierarchical 2NN of $T3$ will be considered.

Definition 6.2.10. Hierarchical Reverse k Farthest Neighbour Query(HRkFN(q, k, T_q)) is a method used to find the region and users in it that consider query point q as one of their k farthest facility points of subtypes T_q. This query is issued from a facility point q of type T_q.

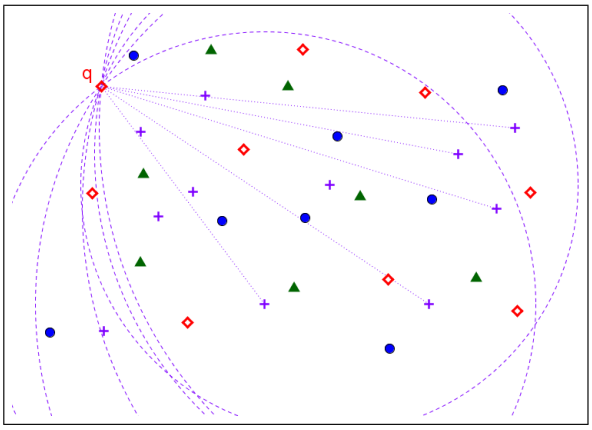


Figure 6.11: $HRkFN(q, 2, T3), q \in U$

Figure 6.10 shows an example of HRkFN query which uses the same hierarchical structure as that shown in Figure 6.8a. The query is issued from a red diamond query point q and is intended to find all users that consider this query point as one of their *two* farthest T3 facility points. From the hierarchical structure, this query will ignore any blue circle facility points as the competitors.

6.3 Polychromatic Query Processing

In this section, we will discuss a region-based polychromatic query processing. A polychromatic query is a type of query where the objects types retrieved from this query are more than one. In polychromatic query, there are more than one facility types and the query can be issued either from a facility point that provides service to other facility points or from the user that needs service from facility points. In solving polychromatic queries with an HSVD, these facility points are also generator points. Hence, the HSVD structure is constructed with multiple types of facility points. In this section, we will cover several variations of polychromatic queries which are :

1. Polychromatic Nearest Neighbours
   (a) Polychromatic k Nearest Neigbours
   (b) Polychromatic all-k Nearest Neigbours
2. Polychromatic Farthest Neighbours
   (a) Polychromatic k Farthest Neigbours
   (b) Polychromatic all-k Farthest Neigbours
3. Polychromatic Reverse Queries
   (a) Polychromatic Reverse k Nearest Neigbours
   (b) Polychromatic Reverse k Farthest Neigbours

The following subsection will discuss how to adapt HSVD to support polychromatic spatial query. We will describe the data structures needed to support this type of query, and also the algorithm to solve polychromatic spatial query based on HSVD.

6.3.1 Data Structure

In a polychromatic query, the facility points consist of multiple facility types, and a spatial query will involve more than two facility types. Since the polychromatic query will be

solved by using an HSVD, this diagram will be constructed from various types of generator points. However, since all generator points will be used to construct the diagram, we will treat all facility points as the same generator points. To distinguish this type of generator point from the others, we add an attribute to each generator point to indicate the generator point type.

Structure	**Attributes**	**Description**
Facility Point	name	Facility point name
	location	Facility point coordinate
	fType	Facility point type

Table 6.1: Polychromatic Facility Point Structure

Table 6.1 shows the basic data structure for a facility point where we add a facility type as an additional attribute to the generator point to identify the facility type. However, the HSVD structure and the method of HSVD construction will remain the same. All the properties of the HSVD structure as mentioned in Chapter 3 will also remain the same.

To process polychromatic queries, only those facility points that have type in T_q will proceed; all other types will be ignored. To simplify the process, we will use the $trim(seq, T_q)$ function to keep all facility points with type T_q and remove all other facility points from the sequence. The illustration of this function is shown in Figure 6.12. In $trim()$ function, we also use function $contains([list\_attr], attr)$ to check if an attribute $attr$ exists in the list of attribute $[list\_attr]$. The contains function will return *True* if $attr$ is listed in $list\_attr$. Otherwise, this function will return *False* and the $attr$ will not be taken. The detailed trim algorithm is shown in algorithm 19.

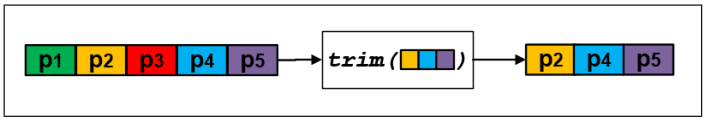


Figure 6.12: Function $trim()$

The Example

To demonstrate the polychromatic queries, in this section we will use the same HSVD structure for all variations of polychromatic queries to simplify the explanations. The HSVD structure is constructed from eight generator points from three facility types, which are *blue circle (BC)*, *green triangle (GT)*, and *red diamond (RD)*. The list of facility points and their facility types are listed in Table 6.3. This example is shown in Figure 6.13.

Algorithm 19: Trim function

Data: seq, T_q
Result: $tseq$ trimmed sequence

```
initialize(tseq);
for i = 1 to sizeOf(seq) do
    if contains(T_q, seq[i].fType) then
        tseq.add(seq[i]);
    end
end
return tseq;
```

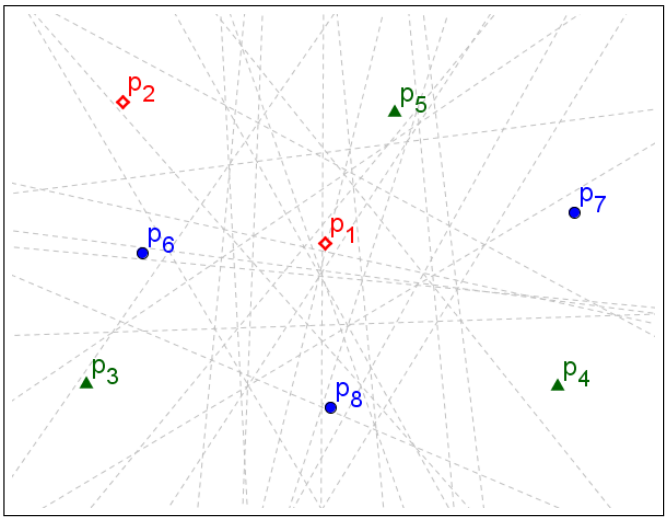


Figure 6.13: Polychromatic HSVD

Points	(x,y)	fType
p_1	(x, y)	RD
p_2	(x, y)	RD
p_3	(x, y)	GT
p_4	(x, y)	GT
p_5	(x, y)	GT
p_6	(x, y)	BC
p_7	(x, y)	BC
p_8	(x, y)	BC

Table 6.2: Polychromatic Facility Point Structure

In this example, we predefine the facility type queries $T_q = \{BC, GT\}$. The query will be issued from a query point q, where the type of query point will be determined from the type of query itself.

6.3.2 Polychromatic k Nearest Neighbours (PkNN)

Polychromatic k nearest neighbour method is the method of finding k nearest facility point type T_q, where T_q is the list of expected facility points. In this query, the nearest facility points must have the type listed in T_q. This query is issued from a user $q \in U$.

Consider this daily activity as a demonstration of a polychromatic query. Assume that a customer who just moved to a new suburb needs to go to a supermarket for his groceries. There are many supermarket franchises in that suburb, although this customer wants to consider only three supermarkets which are Coles[1], Woolworths[2] and Aldi[3]. This customer needs to find the nearest supermarket from his home and does care which of these three is the nearest supermarket. With polychromatic k nearest neighbour query, the customer can assign these franchises as the expected types and the query will return the first nearest supermarket from listed types as the result.

In the query processing, PkNN query uses the same algorithm as the kNN algorithm with an additional step to identify the facility type in the sequence. The complete algorithm for polychromatic k nearest neighbour query is as follows:

Algorithm 20: Polychromatic k-NN Query with HSVD

Data: k, q, T_q
Result: $A = \{a_1, a_2, ..., a_k\}$ list of k facility points that satisfy $P\_kNN(q)$

```
1 initialize(A);
2 r ← getCell(q); /* Get Voronoi cell contains q                    */
3 tseq ← trim(r.seq, T_q);/* trim the sequence based on T_q          */
4 if k > sizeOf(tseq) then
5 |   k ← sizeOf(tseq);
6 end
7 for i = 1 to k do
8 |   A.add(getObject(tseq[i]));
9 end
```

Algorithm 20 explains how to a solve PkNN query. This algorithm is based on algorithm 7 for bichromatic kNN query with the HSVD structure. The algorithm starts by identifying the cell that contains the query point q (line 3), followed by trimming the sequence with T_q (line 4). If the value of k is bigger than the remaining generator points in the sequence after the trimming process, all generator points will be taken as the answer (line 5-7), otherwise the first k generator points will be taken (line 8-10).

[1] coles.com.au
[2] woolworths.com.au
[3] aldi.com.au

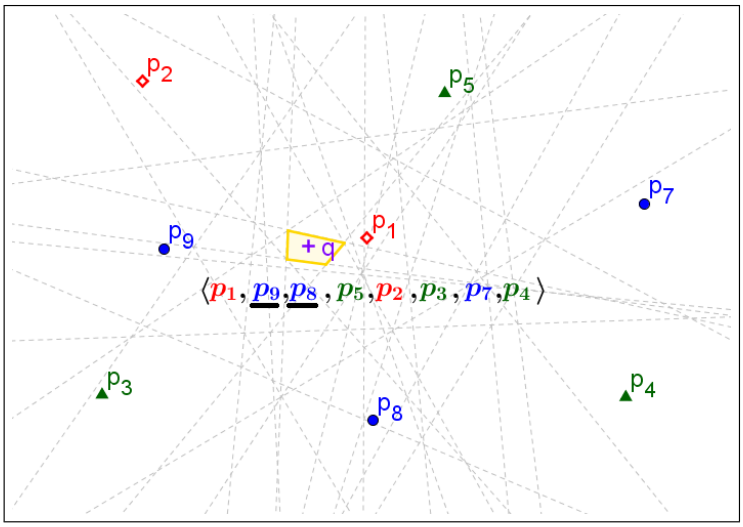


Figure 6.14: $PkNN(q, 2, T_q), T_q = \{BC, GT\}$

Figure 6.14 shows an example of PkNN based on the HSVD structure. The query is issued from a user q to find the two nearest facility points, either with type BC or GT. The cell that contains the query point is indicated by the yellow polygon, and the sequence in this cell is $\langle p_1, p_9, p_8, p_5, p_2, p_3, p_7, p_4 \rangle$. Since $RD \notin T_q$, all facility points type RD will be ignored, and facility point p_1 will not be considered as the first nearest facility point from q. The answer to this query is indicated by the underlining in the sequence, where the first nearest facility point for q will be p_9 followed by p_8, because these facility points type are listed in T_q.

6.3.3 Polychromatic all-k Nearest Neighbours (Pall-kNN)

Polychromatic all-k nearest neighbour method is the method of finding the k nearest facility point on each type in T_q, where T_q is the list of expected facility points. This query is issued by a user $q \in U$.

The real application for this type of query can be demonstrated by the previous scenario with the supermarket. Assume that this customer knows that each supermarket has its own advantages because each supermarket will not attach the lowest prices to all of its items. In order to obtain all cheapest prices for all the items needed, this customer needs to find the first nearest supermarket for each type. This query will return the list of nearest supermarket for each franchise as the result.

In the query processing, a Pall-kNN query uses an algorithm similar to the PkNN algorithm with the additional step of repeating kNN search for each facility type. The algorithm of Pall-kNN is described as follows:

Algorithm 21: Polychromatic all-kNN Query with HSVD

Data: k, q, T_q
Result: $A = \{a_1, a_2, ..., a_k\}$ list of k facility points that satisfy $P\_kNN(q)$

```
1  initialize(A);
2  r ← getCell(q); /* Get Voronoi cell contains q                    */
3  for j = 1 to sizeOf(T_q) do
4      count ← 0
5      i ← 1
6      while (count < k)and(i ≤ m) do
7          if (T_q[j] = r.seq[i].fType) then
               /* if the facility type is part of T_q, p is taken    */
8              A.add(getObject(r.seq[i]));
9              count++;
10         end
11         i++;
12     end
13 end
```

Algorithm 21 explains how to solve Pall-kNN query. This algorithm is based on algorithm 20 for PkNN. The algorithm starts by identifying the cell that contains the query point q (line 3). In Pall-kNN, the PkNN process (line 7-13) has to be repeated j times, where $j = |T_q|$ to get k facility points for each type (line 4). The algorithm will stop when each facility type has retrieved k facility points, or no more facility points can be processed in the distance sequence.

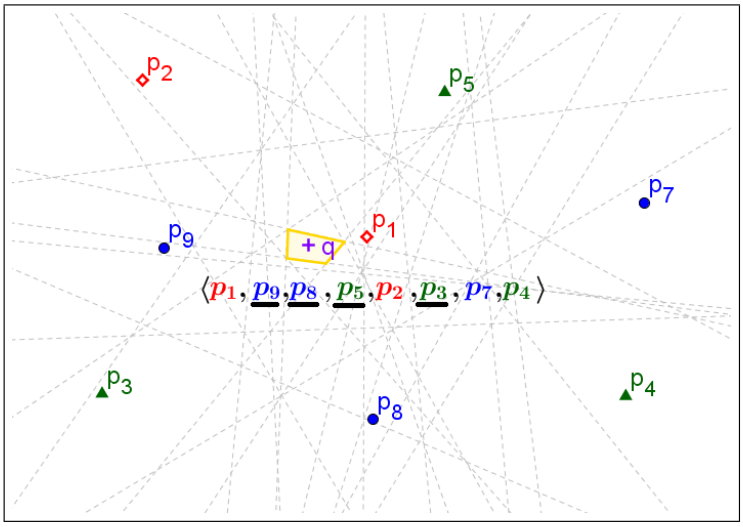


Figure 6.15: $P\ all - kNN(q, 2, T_q), T_q = \{BC, GT\}$

Figure 6.15 shows an example of polychromatic all-k nearest neighbour query based on the HSVD structure. The query is issued from a user q to find two nearest facility points for all facility point types in T_q . The cell that contains the query point is indicated by the yellow polygon, and the sequence in this cell is $\langle p_1, p_9, p_8, p_5, p_2, p_3, p_7, p_4 \rangle$. Since $RD \notin T_q$, all facility points type RD will be ignored, and facility point p_1 will not be considered as the first nearest facility point from q. The answer to this query is indicated by the underline in the sequence. In this example, the answer to this query will be $\{p_9, p_8, p_5, p_3\}, where\{p_9, p_8\}$ are two nearest facilities type BC and $\{p_5, p_3\}$ are two nearest facilities type GT.

6.3.4 Polychromatic k Farthest Neighbours (PkFN)

Polychromatic k farthest neighbour method is the method of finding k farthest facility point type T_q, where T_q is the list of expected facility points. In this query, the farthest facility points must have the type listed in T_q. This query is issued by user $q \in U$.

Consider the following example in our daily lives. Assume that a backpacker wants to find the farthest local public transport stations in a state from his current location, so he can enjoy local public transport as far as he can. However, he considers only buses or train stations. In Victoria, the public transport fare depends on the zone instead of distance, so for a tourist, it is best to embark on the longest journey instead of shorter ones. Since this backpacker does not mind the type of public transport he uses, the query will return either a train station or bus station as the farthest station in the state.

In the query processing, a PkFN query use the same algorithm as the BkFN algorithm with the additional step of identifying the facility type in the sequence. The complete algorithm for a PkFN query is as follows:

Algorithm 22 shows how to solve $PkFN$ query. This algorithm is based on algorithm 20 for the PkNN query with an additional inversion process for the sequence. The algorithm starts by identifying the cell that contains the query point q (line 3) and then trimming (line 4) and inversion to facilitate the farthest process (line 5). If the value of k is greater than the number of remaining generator points in the sequence, all remaining generator points will be taken (line 6-8). Otherwise, only first k generator points from the inverted sequence will be taken (line 9-11).

Algorithm 22: Polychromatic k-FN Query with HSVD

Data: k, q, T_q
Result: $A = \{a_1, a_2, ..., a_k\}$ list of k facility points that satisfy $P\_kNN(q)$

```
1  initialize(A);
2  r ← getCell(q); /* Get Voronoi cell contains q                    */
3  tseq ← trim(r.seq, T_q);/* trim the sequence based on T_q         */
4  tseq ← inverse(tseq);/* invert the sequence                       */
5  if k > sizeOf(tseq) then
6  |  k ← sizeOf(tseq);
7  end
8  for i = 1 to k do
9  |  A.add(getObject(tseq[i]));
10 end
```

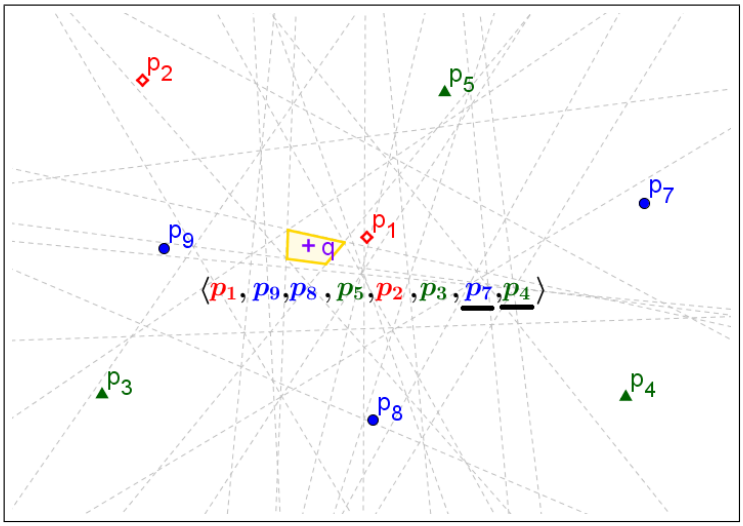


Figure 6.16: $PkFN(q, 2, T_q), T_q = \{BC, GT\}$

Figure 6.16 shows an example of a PkFN query. We still use the same example as for the previous type of query. Assume that the query is issued from a yellow polygon where $T_q = \{BC, GT\}$. The query is intended to find the two farthest facility points either type BC or GT. In this example, the query retrieves $\{p_4, p_7\}$ as the two farthest facility points from q.

6.3.5 Polychromatic all-k Farthest Neighbours(Pall-kFN)

Polychromatic all-k farthest neighbour method is the method of finding k farthest facility points for each facility type in T_q, where T_q is the list of expected facility points. Similar to the previous polychromatic queries, this query is issued from a user $q \in U$.

Consider the backpacker example in the PkFN case. Assume that the backpacker wants to find the farthest local public transport stations for trains and also buses in a

state from his current location. Unlike $PkFN$ that will obtain only one farthest facility from two expected facility types, this query will retrieve one facility for each facility type. So, the backpacker can try two public modes of transport with different types.

In the query processing, a Pall-kFN query use the same algorithm as the PkFN algorithm with an additional step to repeat the process j times, where $j = |T_q|$. The complete algorithm for a Pall-kFN query is described in algorithm 23.

Algorithm 23: Polychromatic all-kFN Query with HSVD

```
   Data: k, q, T_q
   Result: A = {a_1, a_2, ..., a_k} list of k facility points that satisfy P_kNN(q)
 1 initialize(A);
 2 r ← getCell(q); /* Get Voronoi cell contains q                    */
 3 r.seq ← inverse(r.seq);
 4 for j = 1 to sizeOf(T_q) do
 5 |   count ← 0
 6 |   i ← 1
 7 |   while (count < k)and(i ≤ m) do
 8 |   |   if (T_q[j] = r.seq[i].fType) then
   |   |   |   /* if the facility type is part of T_q, p is taken    */
 9 |   |   |   A.add(getObject(r.seq[i]));
10 |   |   |   count++;
11 |   |   end
12 |   |   i++;
13 |   end
14 end
```

Algorithm 23 explains how to solve a Pall-kFN query. This algorithm is based on algorithm 22 for PkFN with an additional repetition for T_q. The algorithm starts by identifying the cell that contains the query point q (line 3), followed by the *inverse()* function (line 4). To obtain k facility points for each facility type, we have to repeat the $PkFN$ process (line 5). The $PkFN$ is performed by applying *contains()* function to check if the type of a generator point on the sequence matched with T_q (line 8-11). If *contains()* function returns *True*, then this facility point is part of the answer. The algorithm will stop if k number of facility points have been retrieved.

Figure 6.17 shows an example of a Pall-kFN query. We still use the same example as for the previous type of query. Assume that the query is issued from a yellow polygon where $T_q = \{BC, GT\}$. The query is intended to find the two farthest facility points for each type BC or GT. For the GT type, the query retrieves $\{p_4, p_3\}$, while for BC type, the query retrieves $\{p_7, p_8\}$.

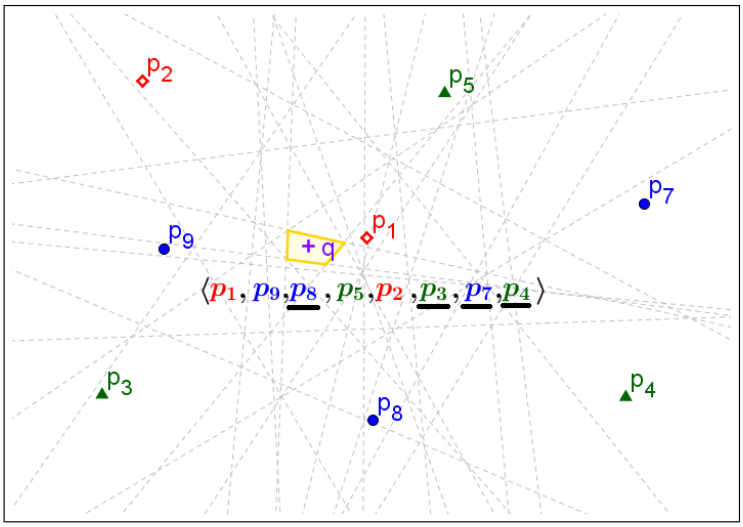


Figure 6.17: $Pall - kFN(q, 2, T_q), T_q = \{BC, GT\}$

6.3.6 Polychromatic Reverse k Nearest Neighbours (PRkNN)

Polychromatic reverse k nearest neighbour method is the method of finding the region and users in the region that consider the query point as their PkNN on type T_q. This query is issued from a generator point, where the type of this generator point must be listed in T_q.

Consider the following example to illustrate $PRkNN$ query in our daily lives. Assume that a business owner opens a new Chinese restaurant in a particular suburb and he wants to identify his potential customers as well as his competitors. Since his restaurant is a Chinese restaurant, he will not consider all types of restaurants as his competitors. Instead, he considers only other Chinese restaurants, Indian restaurants and Japanese restaurants as his direct competitors. To identify his region as well as the potential customers, a polychromatic reverse k nearest neighbour method is the best possible solution for this problem.

The PRkNN cannot be solved by using the CLI index structure as described in Chapter 5 because the level of each generator point can be changed according to T_q. Hence, at the moment, the PRkNN can only be solved by reading all available cells in HSVD.

Algorithm 24 explains the PRkNN query processing. Since there are no indexes or retrieval structures that can be used to identify the correct cells to be taken, this algorithm has to read all the cells to find the suitable cells for this query (line 2). Each cell needs to be trimmed (line 4) and then $inPosition(seq, q)$ function returns the position of the query point in the sequence. If the position is less than or equal to k, then this cell is

Algorithm 24: *PRkNN* with HSVD structure

Data: q, k, T_q
Result: $R = \{r_1, r_2, ..., r_n\}$ list of Voronoi cells that satisfy $P\_RkNN(q, k, T_q)$

```
1 initialize(R);
2 for i = 1 to sizeOf(Vh) do
3     r ← Vh[i];/* get the cell                                          */
4     r.seq ← trim(r.seq, T_q);/* remove other generator points type except T_q
          */
5     if (inPosition(r.seq, q) ≤ k) then
          /* if q is in position ≤ k on the trimmed sequence            */
6         R.add(r); /* get this cell                                    */
7     end
8 end
```

taken as part of the reverse region. The algorithm will be stopped after all cells have been examined.

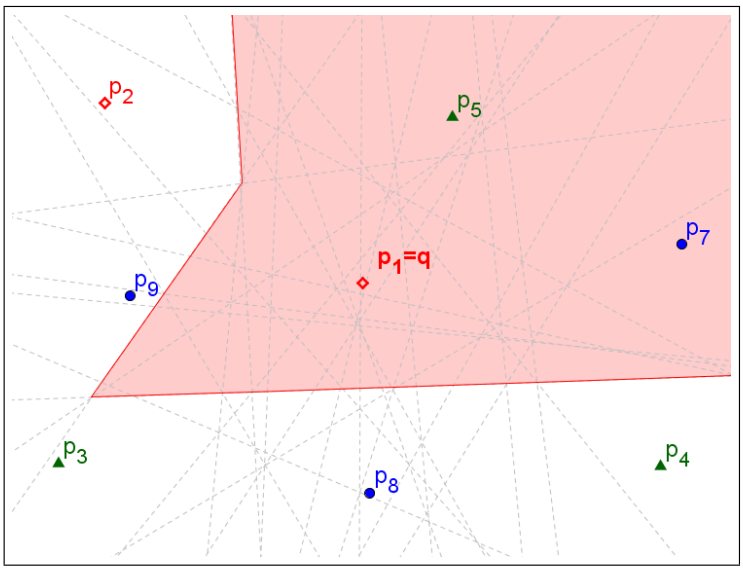


Figure 6.18: $PRkNN(q, 2, T_q), T_q = \{BC, RD\}$

Figure 6.18 shows an example of a PRkNN query. This example uses the same case as stated previously. In this example, the query is issued from a facility point $q = p_1$. This query is intended to find the region where the users will consider the query point as one of their nearest neighbours by ignoring all GT type facility points because this type is not listed in T_q. As we can see from this figure, p_5 is included in the region. However, since GT is ignored, all facility points with this type will not be considered as the competitors.

6.3.7 Polychromatic Reverse k Farthest Neighbours (PRkFN)

Polychromatic reverse k farthest neighbour method is the method of finding the region and users in the region that consider the query point as their PkFN on type T_q. This query is issued from a generator point, where this generator point type must be listed in T_q.

Consider the following example to illustrate $PRkFN$ query in our daily lives. Assume that the city council needs to identify the region for the location of a new landfill and this location must be the farthest region from the residential points and city water supply reservoir. The city identified the largest reservoir as the query point, while other reservoirs and residential points are supported facility types. The query is intended to find the farthest region from the main reservoir in the city that can be used for new landfill. This query can be solved by using polychromatic reverse k farthest neighbour method.

Similar to $PRkNN$, a polychromatic reverse k farthest neighbour query cannot be solved by using the CLI index structure as described in Chapter 5 because the level of each generator point can be changed according to T_q. Hence, at the moment, the $PRkFN$ can be solved only by reading all available cells in HSVD.

Algorithm 25: $PRkFN$ with HSVD structure

Data: q, k, T_q
Result: $R = \{r_1, r_2, ..., r_n\}$ list of Voronoi cells that satisfy $P\_RkFN(q, k, T_q)$

```
1 initialize(R);
2 for i = 1 to sizeOf(Vh) do
3 |   r ← Vh[i];/* get the cell                                          */
4 |   r ← trim(r, T_q);/* remove other generator points type except T_q  */
5 |   r.seq ← inverse(r.seq);/* inverse the sequence                     */
6 |   if (inPosition(r.seq, q) ≤ k) then
  |   |   /* if q is in position ≤ k on the trimmed sequence            */
7 |   |   R.add(r); /* get this cell                                    */
8 |   end
9 end
```

Algorithm 25 explains the PRkFN query processing. Similar to the PRkNN query, this algorithm performs in a sequential manner, where all cells of HSVD have to be read since no indexes can be applied in this algorithm (line 2). Each cell needs to be trimmed (line 4) and inverted (line 5) to support farthest queries. Function $inPosition(seq, q)$ will find the position of query point q in the sequence seq. If the position is less than or equal to k, then this cell will be considered as part of the answer; otherwise, the cell will be ignored.

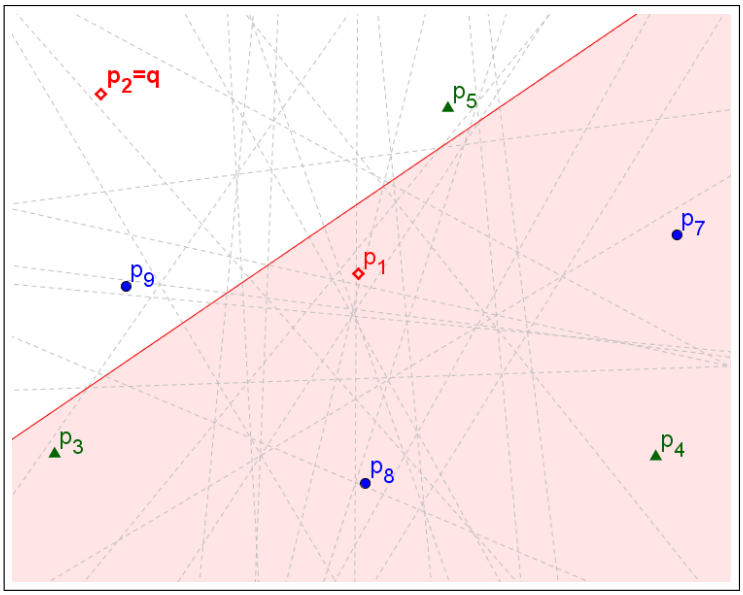


Figure 6.19: $PRkFN(q, 2, T_q), T_q = \{BC, RD\}$

Figure 6.19 shows an example of a PRkFN query, where the query point is $q = p_2$, $k = 2$ and $T_q = \{BC, RD\}$. This query will ignore all facility points of type GT. Hence, in this example, the answer to this query is shown in the shaded area.

6.4 Hierarchical Query Processing

In this section, we will discuss another variation of the polychromatic query, called *hierarchical* query. The hierarchical query is a type of query where the facility types involved in the query exceed two, and these types are constructed in a hierarchical manner. To the best of our knowledge, there are no current researches deal with this particular problem. In this type of query, the facility points are considered as generator points in HSVD, where in constructing the HSVD, we omit the facility types.

Hierarchical problem can appear in our daily lives. For example, assume that a customer is looking for Chinese restaurants nearest to his current location, but he doesn't want to travel for farther than 2Km. Since he cannot find any Chinese restaurants within his desired distance, he will accept any Asian restaurant as long as this restaurant is located within a 2Km radius. There are so many countries in Asia and it is not possible to find each type of restaurant one-by-one. Therefore, it would be convenient if he could just issue a query for nearest Asian restaurants.

In this section, we will cover some of the variations of hierarchical queries that can be found in our daily lives which to the best of our knowledge, no-one has discussed previously. The variations of hierarchical queries are:

1. Hierarchical k nearest neighbour queries
2. Hierarchical k farthest neighbour queries
3. Hierarchical reverse k nearest neighbour queries
4. Hierarchical reverse k farthest neighbour queries

The following subsection will discuss how to adapt the HSVD structure to different kinds of hierarchical queries. We will discuss hierarchical queries processing in nearest neighbour problems, farthest neighbours problems, reverse nearest neighbour problems and reverse farthest neighbours problems.

6.4.1 Hierarchical Structure

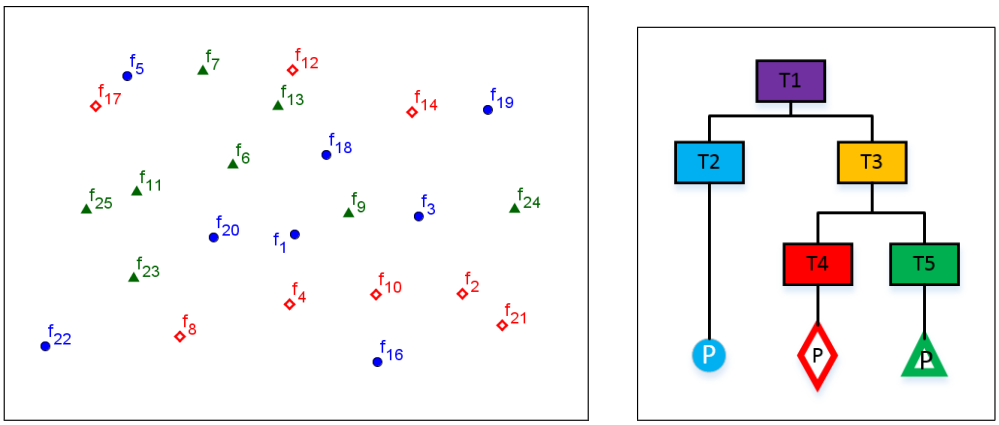
To accommodate a hierarchical structure in an HSVD, we modify the generator point structure to support hierarchical types. An additional attribute for the facility type is added, similar to polychromatic problems. The hierarchical structure for each facility point is stored by using a *list of sequence types* as shown in Table 6.3.

Structure	**Attributes**	**Description**
Facility Point	name	Facility point name
	(x, y)	Facility point coordinate
	$HType$	List of Facility point type

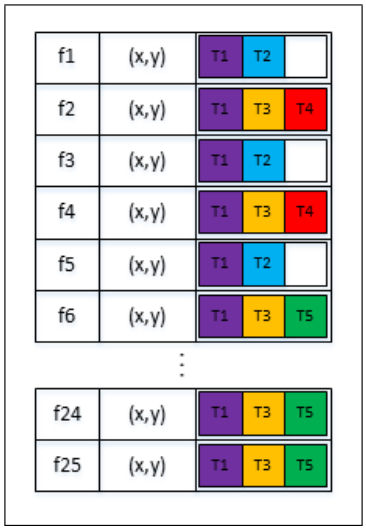
Table 6.3: Hierarchical Facility Point Structure

An example of data structure representation is shown in Figure 6.20. Figure 6.20a shows the multiple types of facility points that are distributed in the map. The hierarchical types are shown in Figure 6.20b. In this figure, we can see that $T1$ is the general type, and all facility points will be included in this type. $T2$ is the subtype of $T1$, and only blue circle facility points are considered in this type. Since $T2$ is subtype of $T1$, all blue points facility points will have $\{T1, T2\}$ as their types in hierarchical order. The new structure is created to provide query by hierarchal type of objects, and provide direct pointer to the objects based on a type, rather than performing point-to-point verification. With the same concept, all the red diamond types are considered as $\{T1, T3, T4\}$ types, while all

green triangle types will be considered as $\{T1, T3, T5\}$ types. The data structure for the facility points in hierarchical order can be seen in Figure 6.20c.



(a) Facility Points (b) Hierarchical Types



(c) Hierarchical Data Structure

Figure 6.20: Example of Hierarchical representation

In hierarchical queries, the attribute to store the facility type is different from the polychromatic queries, so we cannot use the $trim()$ function in polychromatic. Therefore, we use another trimming function for the hierarchical structure called $htrim(sequence, T_q)$ function. This function still has the same input, which is the sequence and T_q. However, since the content of the facility types attribute is different, we have to make a small adjustment to the function.

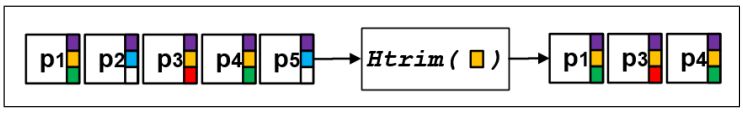


Figure 6.21: function $htrim(sequence, T_q) \rightarrow sequence$

Algorithm 26: Hierarchical Trim (*htrim*) function

```
Data: seq, T_q
Result: hseq hierarchical trimmed sequence
initialize(hseq);
for i = 1 to sizeOf(seq) do
    if contains(seq[i].HType, T_q) then
        hseq.add(seq[i]);
    end
end
return hseq;
```

The hierarchical trimming function is shown in algorithm 26. We define the function as $htrim(sequence, T_q) \rightarrow sequence$. This function accepts the distance sequence of a Voronoi cell and T_q, and removes all facility points that do not have type T_q in their hierarchical types structure. The illustration of this function is presented in Figure 6.21. In this function, we still use $contains([list\_attr], attr)$ function to check whether an attribute $attr$ exists in the list of attribute $[list\_attr]$. In polychromatic queries, T_q consists of a list of types, while in hierarchical queries, the facility type in each facility point consists of list of types. Therefore, we have to swap the input for the *contains*() function in hierarchical trimming.

The Example

To illustrate the hierarchical process for each query type, we will use the same example from polychromatic queries in Figure 6.13 with an additional hierarchical structure as shown in Figure 6.22. In this example, $T1$ is general type, hence if $T_q = \{T1\}$, the hierarchical query will use all available facility points. The hierarchical order is stored in each facility point. The HSVD is constructed from these facility points and the HSVD structure will remain the same as the HSVD structure for polychromatic queries.

6.4.2 Hierarchical k Nearest Neighbours (HkNN)

Hierarchical k nearest neighbour method is a method of finding k nearest facility points type T_q and all of its subtypes. The query is issued by a user $q \in U$. HkNN query is very similar to PkNN query, except that the type is constructed in hierarchical order, and a specific retrieval method is needed to obtain the answer.

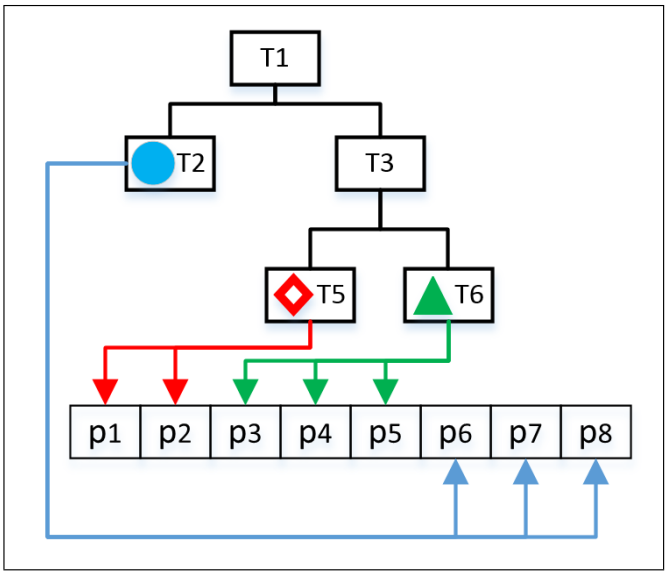


Figure 6.22: The example of Hierarchical Structure

HkNN queries are common in our daily activities. However, to the best of our knowledge, this problem has not been solved recently. Assume that a customer is going to find a restaurant during a holiday, and he would prefer a fast food restaurant, and he chooses McDonald's. However, he cannot find any restaurants within a certain range. He knows that there are many different franchises of fast food restaurants, and it will be very time consuming if he has to perform the same query for each different franchise. Since all of these franchises are classified as fast food restaurants, it would be easier if he could query any nearest fast food restaurants and all of its subcategories, without considering any other types of restaurants. In this example, the problem can be easily solved by using HkNN query.

The HkNN query is very similar to PkNN query. The main difference is how to determine whether a facility point in a sequence must be considered or must be ignored. In HkNN query, we use $htrim()$ function to remove all facility points that do not belong to type T_q

Algorithm 27 explains the HkNN query processing. This algorithm is very similar to PkNN, and the main difference is only in the trimming function. The first step is to obtain the cell that contains the query point (line 2), and then followed by hierarchical trimming (line 3). If the remaining facility point is less than the value of k, the value of k will be set to the number of remaining facility points (line 4-6). The algorithm will stop after k nearest facility points type T_q have been retrieved (line 7-9).

Algorithm 27: Hierarchical k-NN Query with HSVD

Data: k, q, T_q
Result: $A = \{a_1, a_2, ..., a_k\}$ list of k facility points that satisfy $P\_kNN(q)$

```
1 initialize(A);
2 r ← getCell(q); /* Get Voronoi cell contains q                    */
3 r.seq ← htrim(r.seq, T_q); /* trim the sequence                   */
4 if (k > sizeOf(r.seq)) then
5 |  k ← sizeOf(r.seq);
6 end
7 for i = 1 to k do
8 |  A.add(getObject(r.seq[i]));
9 end
```

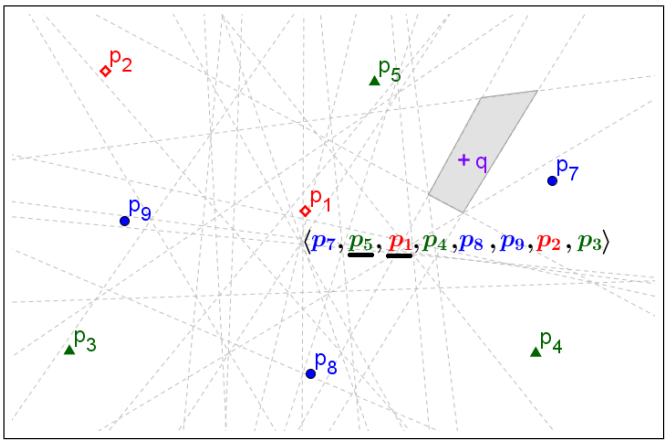


Figure 6.23: $H\_kNN(q, 2, T_q), T_q = T3$

Figure 6.23 demonstrates the HkNN query using predefined example in subsection 6.3.1 with an additional hierarchical structure in subsection 6.4.1. In this example, assume that a user issues a query $H\_kNN(q, 2, T3)$, where this query is intended to find all facility points that have facility type $T3$ or any other subtypes of $T3$. Therefore, all facility point types $T4$ and $T5$ will be taken, since these types are subtype of $T3$. All facility points type $T2$ will be ignored, since $T2$ is not a subtype of $T3$. In this example, the answer to this query in nearest neighbours order is $\{p_5, p_1\}$.

6.4.3 Hierarchical k Farthest Neighbours (HkFN)

Hierarchical k farthest neighbour method is a method of finding k farthest facility points type T_q and all of its subtypes. The query is issued by a user $q \in U$. A hierarchical k farthest neighbour query is very similar to polychromatic k farthest neighbour query, except that the type is constructed in hierarchical order, and a specific retrieval method is needed to obtain the answer.

The main difference between HkFN query and PkFN query is how to determine whether a facility point in a sequence must be considered or must be ignored. In this query, we use the $htrim()$ function to remove all facility points that do not belong to type T_q. To support the farthest query, we also have to inverse the sequence so that we can process the sequence in reverse order.

Algorithm 28: Hierarchical k-FN Query with HSVD

Data: k, q, T_q
Result: $A = \{a_1, a_2, ..., a_k\}$ list of k facility points that satisfy $P\_kNN(q)$

```
1  initialize(A);
2  m ← P /* get number of facility points                    */
3  r ← getCell(q); /* Get Voronoi cell contains q            */
4  r.seq ← htrim(r.seq, T_q); /* trim the sequence           */
5  r.seq ← inverse(r.seq);/* inverse the sequence            */
6  if (k > sizeOf(r.seq)) then
7  |  k ← sizeOf(r.seq);
8  end
9  for i = 1 to k do
10 |  A.add(getObject(r.seq[i]));
11 end
```

Algorithm 28 explains the HkFN query processing. Similar to PkFN query processing, this algorithm begins by finding the cell that contains the query point (line 3). The next step is to perform hierarchical trimming function to remove all facility points that do not belong to type T_q (line 4), followed by reversing the sequence to support the farthest query (line 5). If the remaining facility points are less than k, the value of k will be set to the remaining number of facility points (line 6-8). The algorithm will stop after k farthest facility points have been retrieved (line 9-11).

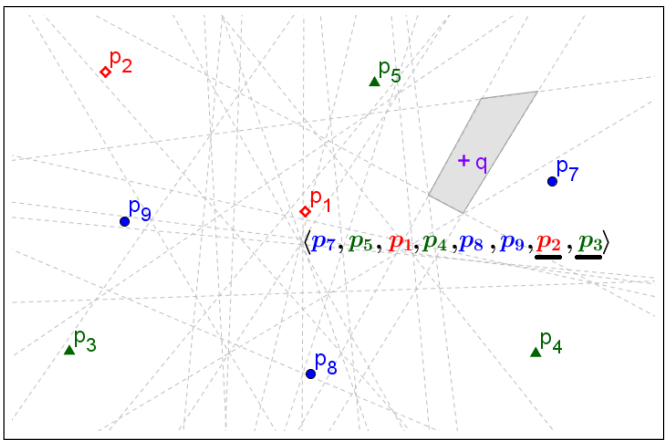


Figure 6.24: $H\_kFN(q, 2, T_q), T_q = T3$

Figure 6.24 shows an example of HkFN query. We use predefined example in subsection 6.3.1 with an additional hierarchical structure in subsection 6.4.1. In this example, assumed the same user issues a query $H\_kFN(q, 2, T3)$, where this query is intended to find any k farthest facility points that have facility type $T3$ or any other subtypes of $T3$. Therefore, all facility points type $T4$ and $T5$ will be taken, since these types are subtypes of $T3$. All facility points type $T2$ will be ignored, since $T2$ is not a subtype of $T3$. In this example, the answer to this query in farthest neighbours order are $\{p_3, p_2\}$.

6.4.4 Hierarchical Reverse k Nearest Neighbours (HRkNN)

Hierarchical structure can also be applied in RNN queries, called hierarchical reverse k nearest neighbour queries. This type of query is intended to find the region and users in it that consider the query point is one of their HkNN on type T_q.

Consider the following commonplace application. Assume that an owner of a McDonald's franchise would like to know the region of his nearest potential customers, influenced by other McDonald's, KFC, Red Roaster, and all other restaurants. Since he considers only other fast food restaurants as his direct competitors, he will ignore all other types of restaurants. The expected region for this query should be influenced only by other fast food restaurants, and the potential customers in this region will consider his restaurant as the nearest fast food restaurant, compared with other fast food restaurants.

The method used to solve HRkNN query is very similar to PRkNN query, except for the trimming process. The applied trimming process, $htirm()$ function, is based on the hierarchical structure of each facility point, where this function is made to remove all facility points from the sequence if these facility points do not belong to type T_q.

Algorithm 29: $HRkNN$ with HSVD structure

Data: q, k, T_q
Result: $R = \{r_1, r_2, ..., r_n\}$ list of Voronoi cells that satisfy $H\_RkNN(q, k, T_q)$

```
1 initialize(R);
2 for i = 1 to sizeOf(Vh) do
3 |   r ← Vh[i];/* get the cell                                   */
4 |   r.seq ← htrim(r.seq, T_q);/* trim the sequence              */
5 |   if (inPosition(r.seq, q) ≤ k) then
  |   |   /* if q is in position ≤ k on the trimmed sequence      */
6 |   |   R.add(r); /* get this cell                             */
7 |   end
8 end
```

Algorithm 29 shows the process of HRkNN query processing. Similar to the PRkNN method, no retrieval structure is applied in this query, so all cells have to be read in order to answer the query (line 2). The first step in checking the cell is to perform the hierarchical trimming function on each cell (line 4) to remove all facility points that do not belong to type T_q. The $inPosition()$ function is used to determine whether the query point position is less than or equal to k. If the position of q is less than or equal to k, this cell will be considered as part of the answer. The algorithm will stop if all cells have been processed, and non-generator-point types U can be retrieved from selected cells.

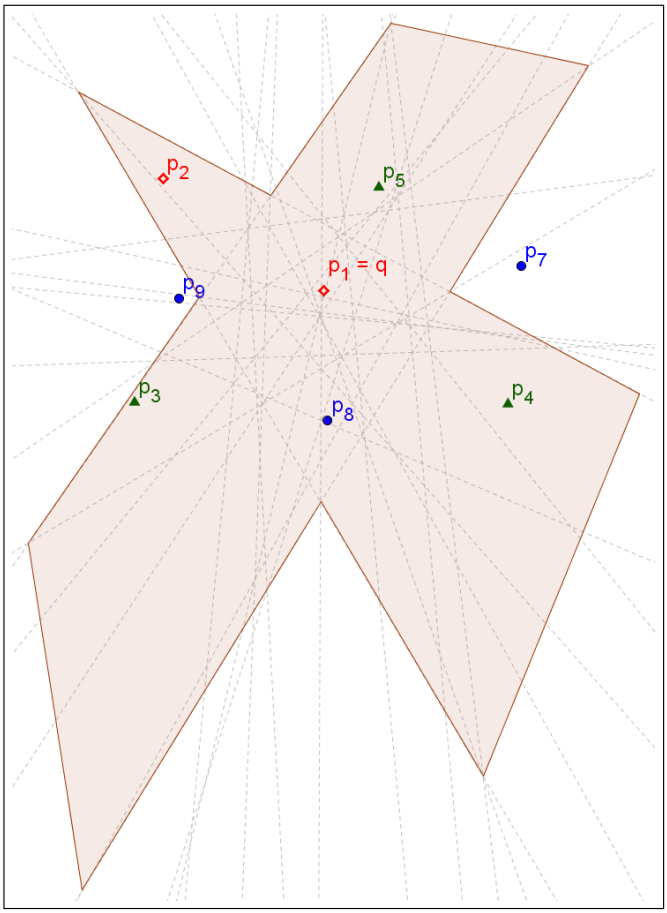


Figure 6.25: $HRkNN(p_1, 2, T_q), T_q = T3$

Figure 6.25 shows an example of HkNN query. This example uses the same example as for the previous query type as shown in Figure 6.13 . The HSVD structure is constructed by all eight facility points, which all have hierarchical types. The query is issued from p_1, and is intended to obtain the region and all users in the region that consider the query point as part of their hierarchical two nearest facility points under $T3$ type. From the hierarchical structure, $T3$ has subtype $T4$ and $T5$. Therefore the regions will be

influenced only by facility points type $T4$ and $T5$. The region for this query is the shaded area in this example.

6.4.5 Hierarchical Reverse k Farthest Neighbours (HRkFN)

Hierarchical reverse k farthest neighbour method is aimed at finding the region and users in it that consider the query point is one of their HkFN on type T_q. The method used to solve HRkFN query is very similar to PRkFN query, except for the trimming process. The applied trimming process, $htirm()$ function, is based on a hierarchical structure for each facility point. This function is made to remove all facility points from the sequence if these facility points do not belong to type T_q.

Algorithm 30: $HRkFN$ with HSVD structure

Data: q, k, T_q
Result: $R = \{r_1, r_2, ..., r_n\}$ list of Voronoi cells that satisfy $H\_RkFN(q, k, T_q)$

```
1 initialize(R);
2 for i = 1 to sizeOf(Vh) do
3 |   r ← Vh[i];/* get the cell                                   */
4 |   r.seq ← htrim(r.seq, T_q);/* trim the sequence              */
5 |   r.seq ← inverse(r.seq);/* inverse the sequence              */
6 |   if (inPosition(r.seq, q) ≤ k) then
  |   |   /* if q is in position ≤ k on the trimmed sequence      */
7 |   |   R.add(r); /* get this cell                             */
8 |   end
9 end
```

Algorithm 30 shows the method used to solve HRkFN query. Similar to PRkFN query, no retrieval structure can be used to improve the retrieval process. Therefore, the retrieval can be done only by reading all available cells (line 2). The hierarchical trimming function is applied to each cell to remove all facility points that do not belong to type T_q (line 4), followed by sequence inversion to support farthest query (line 5). The next step is to determine the position of the query point on the remaining sequences. If the position is less than or equal than k, then the cell will be added to the result set. Otherwise the cells will be ignored. The algorithm will stop once all cells have been processed.

Figure 6.26 shows an example of HRkFN query. In this example, the HSVD structure is constructed from eight facility points from different facility types in hierarchical order. The facility point p_5 issues a query $HRkNN(p_5, 1, T3)$ to obtain the region and users in the region that consider p_5 as their farthest facility point in type $T3$.

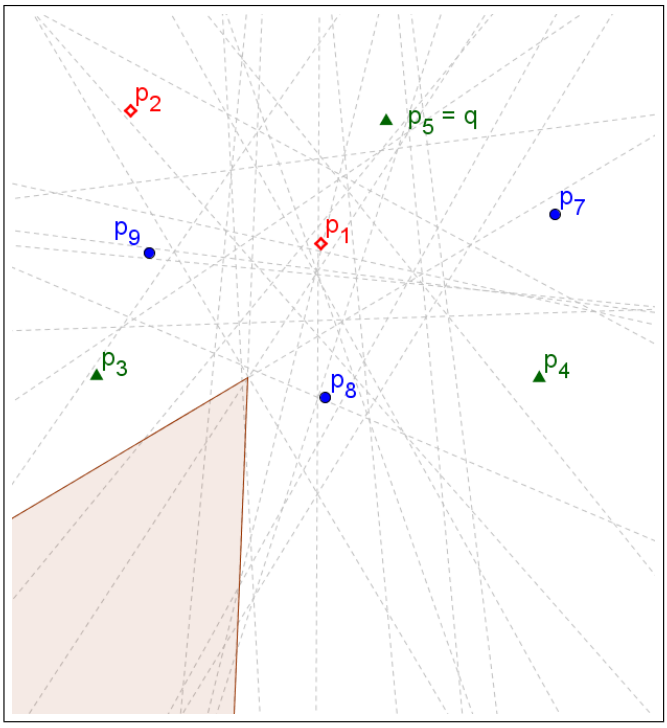


Figure 6.26: $HRkFN(p_5, 1, T_q), T_q = T3$

6.5 Discussion

In this chapter, we propose region-based **polychromatic** and **hierarchical** spatial queries processing. Polychromatic query processing is the method to apply to more than two types of facility points in the spatial query. To the best of our knowledge, only authors (Zhao et al., 2009) that use more than one type of facility points in nearest neighbour queries. They used a multi-layered of Voronoi diagram, where each facility type has its own Voronoi diagram. The answer to nearest neighbours queries can be obtained by overlaying all Voronoi diagrams. This method relies on order-1 Voronoi diagram, and to find the answer of multiple-object-types nearest neighbour queries, this method firstly will find the cells that contain this query point from all Voronoi diagrams, and the generator points for these cells are considered as the answer for $k = 1$. To get the answer for $k > 1$, this method will perform cell expansion to find the k objects by doing point-to-point calculation in each cell for each type. The cost for this method depends on the number of types and the number of objects. Therefore the processing time is increasing sharply with the increasing object types, and this method only effective for $k = 1$.

In our method, polychromatic query is done by using predefined HSVD, therefore the query cost will remain constant for any number of object types and any number of k,

where the main cost is to find the right cell that contains query point. for any number of object types and any number of k, where the main cost is to find the right cell that contains query point.

We distinguish the performance analysis of the queries based on the cost estimation to answer the queries. The classifications are:

1. k Nearest/Farthest Neighbours

   In this type of query, the performance of these queries for both polychromatic and hierarchical queries depends on the number of cells available in HSVD. Similar to the kNN/kFN query in Chapter 4, the main cost is to find the cell that contains the query point. Since the HSVD structure remains static, the cost of answering either polychromatic or hierarchical queries will remain constant for any value of k.

2. all-k Nearest/Farthest Neighbours

   In this type of query, the number of cells that need to be read in order to identify the cell that contains query point will be the same as k nearest/farthest neighbour queries. If the number of types listed in T_q is j, the sequence has to be read j times before the answer can be obtained. However, this extra efforts do not affect the entire cost, since this process does not need extra access to the cells. Therefore the cost to process this query will be similar to that of kNN/kFN queries.

3. Reverse k Nearest/Farthest Neighbours

   Unlike an RkNN/RkFN query, in both polychromatic and hierarchical queries, there is no retrieval structure that can be used to improve retrieval performance, such as the CLI index used in RkNN/RkFN query in Chapter 5. Therefore, to process polychromatic and hierarchical reverse queries, all available cells in HSVD have to be read and processed, which makes the processing performance lower than RkNN/RkFN with a CLI structure.

6.6 Chapter Summary

In this chapter, we propose region-based polychromatic query processing, and the first in proposing the hierarchical spatial queries. Polychromatic and hierarchical queries are very closely related to our daily activities, although have not attracted much attention from researchers. In this chapter, we formulate the queries definitions, structures and processes, as

well as the generalisation for the queries. We classify the polychromatic queries as: polychromatic k nearest neighbour queries ($PkNN$), polychromatic all-k nearest neighbour queries ($Pall-kNN$), polychromatic k farthest neighbour queries ($PkFN$), polychromatic all-k farhest neighbours queries ($Pall - kFN$), polychromatic reverse k nearest neighbour queries ($PRkNN$), polychromatic reverse k farthest neighbour queries ($PRkFN$), hiearchical k nearest neighbour queries (HkNN), hiearchical k farthest neighbour queries (HkFN), hiearchical reverse k nearest neighbour queries (HRkNN), and hiearchical reverse k farthest neighbour queries (HkFN).

Chapter 7

Conclusion and Future Works

7.1 Conclusions

Region-based query processing with true region in a spatial database focuses in finding a region that contains the correct objects for the query. The proposed region-based query processing method in this thesis removes the need to perform point-to-point calculation and verification step that usually occurs in spatial query processing through candidate regions. Furthermore, the proposed method also avoids the region reconstruction cost for any identical queries that are submitted to the server. This thesis started by presenting a region partitioning concept called **highest order Voronoi diagram (HSVD)** that can be used to identify the true region of various spatial queries and at the same time, avoiding point-to-point and verification phase, and avoiding region recreation for any identical queries submitted to the server. This diagram addresses the challenge of repetitive region reconstructions by providing a set of partitioned regions with distance order information that can be used to answer several spatial queries types. The proposed diagram also addresses the challenges of diverse methods in spatial query processing by introducing a **nearest neighbours framework** to generalize the nearest neighbours query processing, and also a **cell level index (CLI)** to generalize the reverse nearest neighbours query processing. The framework and index are not only capable in generalizing the query processing, but also simplify the farthest query processing for both nearest neighbours queries and reverse nearest neighbours queries. Furthermore, in reverse nearest neighbours queries, the regions do not need to be recreated for any queries with different parameters. This thesis also extended the well-known spatial queries into new level of spatial queries,

which are **polychromatic** and **hierarchical** spatial queries. The overview of this thesis is illustrated in Figure 7.1.

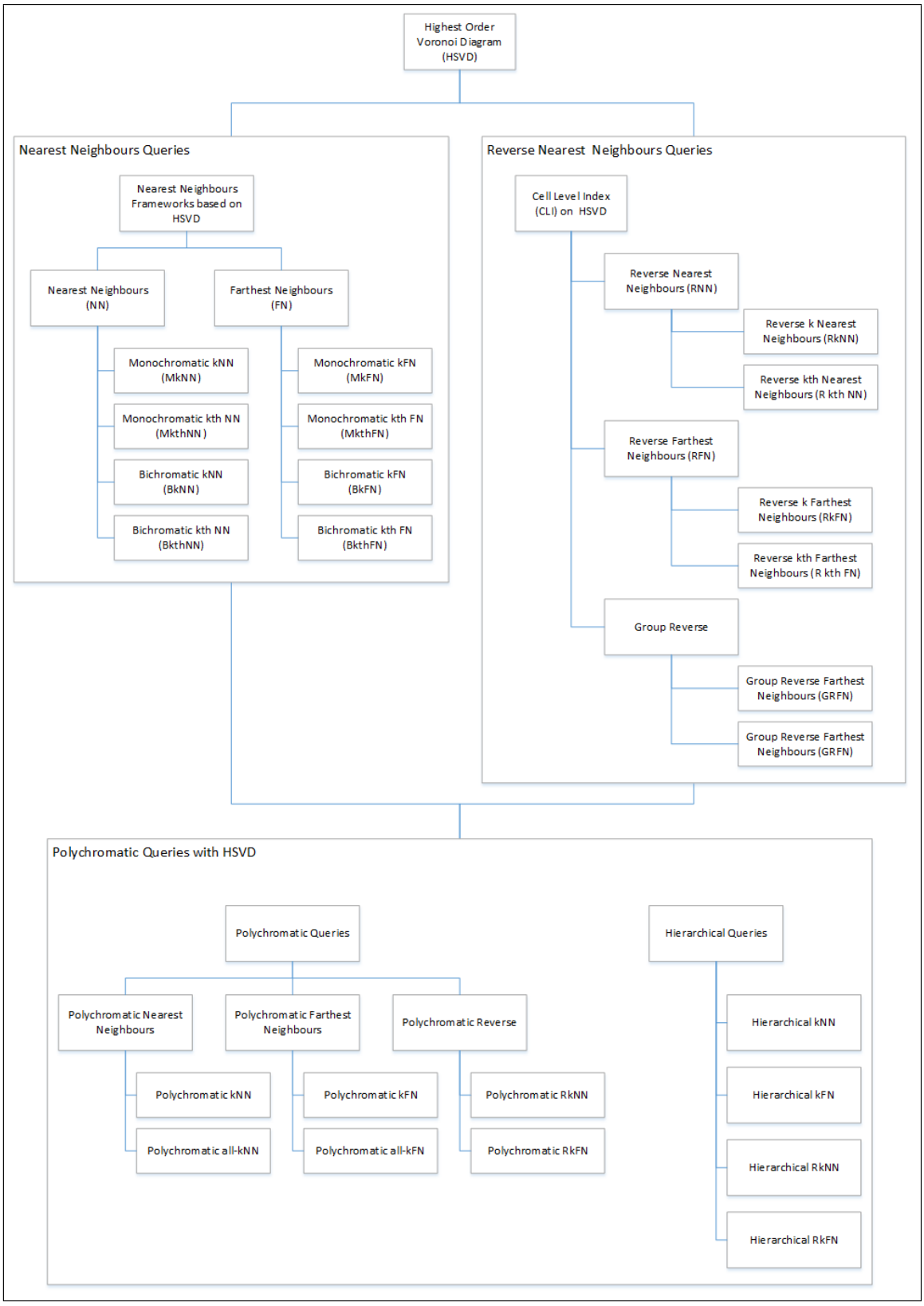


Figure 7.1: Thesis Overview

7.1.1 Highest Order Voronoi Diagram

This thesis proposes a variation of a Voronoi diagram called **highest order Voronoi diagram (HSVD)** that can be used as a model for solving various spatial queries with a region-based approach as well as its dual tessellation, the adjacency graph **(Chapter 3)**. A highest order Voronoi diagram is the generalization of Voronoi diagram where this diagram is capable of constructing an order-1 to order-m Voronoi diagram by merging the cells that have the same distance sequences order. The novel structure of this model focuses on region partitioning with detailed distance order (sequence) in each cell, where the sequences can be used to identify the distance order of all generator points. Unlike an ordinary Voronoi diagram, this diagram can be used to solve the spatial queries where $k > 1$ without any further verification phase. In spatial query processing, HSVD is considered as a predefined structure that will be used for spatial query processing. We present the definition and properties of highest order Voronoi diagram that will be useful in spatial query processing, as well as the properties of an ordinary Voronoi diagram and higher order Voronoi diagram. In constructing the highest order Voronoi diagram, we propose the FLIP method to efficiently identify the distance sequence on each Voronoi cell. The highest order Voronoi diagram has m^4 storage cost at the maximum, where m is the number of generator points. However, our evaluation shows that the average storage cost is less than $m^4/8$.

7.1.2 Nearest Neighbours Query Processing

In nearest neighbours query processing, this thesis proposes a nearest neighbours framework, a generalization to process nearest neighbours query and its variation by using a highest order Voronoi diagram (**Chapter 4**). This framework utilises the sequence generated by the FLIP algorithm to answer any type of nearest neighbour queries The generalization of nearest neighbours query processing consists of 2 main steps, which are (1) locate the cell contains the query point and (2) read the distance sequence. Furthermore, the use of the framework allows the same method applied to the nearest neighbour query be used in answering the farthest neighbour with minimum modification. The only modification is on the function used to read the sequence. Farthest neighbour query reads the sequence in the reverse order of that performed in the nearest neighbour. Hence, the

proposed nearest neighbours framework is applicable to solve monochromatic and bichromatic nearest neighbours and farthest neighbours queries. In addition, our evaluations show the proposed HSVD structure in nearest neighbours keeps the cost of processing nearest neighbours or farthest neighbours constant on any value of k.

7.1.3 Reverse Nearest Neighbours Query Processing

In reverse nearest neighbours query processing, this thesis proposes an index structure called **cell level index** (CLI) **(Chapter 5)**. This index structure is used to identify the right cells for a generator point at a certain k value for reverse nearest neighbour cells. The idea of using the index is to (1) avoid region recreation for queries; (2) simplify the reverse farthest query processing; and (3) identify the k^{th} region for both reverse nearest and reverse farthest neighbours. Furthermore, Chapter 5 also shows that the index structure will minimize the effort in processing the group reverse nearest neighbours and group reverse farthest neighbours. The maximum storage cost for this index is m^3, where m is the number of generator points for HSVD.

7.1.4 Polychromatic Query Processing

A polychromatic query is a new type of spatial query where the number of object types needs to be retrieved are more than two. Polychromatic queries occur naturally in our daily lives. However, this type of query has not attracted researchers attention. This thesis proposed a modified HSVD structure that can support polychromatic queries **(Chapter 6)**. The proposed new polychromatic queries are: polychromatic k nearest neighbour queries, polychromatic all-k nearest neighbour queries, polychromatic k farthest neighbour queries, polychromatic all-k farthest neighbour queries, polychromatic group reverse nearest neighbour queries, and polychromatic group reverse farthest neighbour queries.

Furthermore, we also extend the polychromatic queries to hierarchical queries, where the facility types are classified into certain hierarchical structures, and the queries are issued based on the hierarchical structure (**Chapter 6**). In this query type, we proposed new hierarchical queries, which are: hierarchical k nearest neighbour queries, hierarchical k farthest neighbour queries, hierarchical reverse k nearest neighbour queries and hierarchical reverse k farthest neighbour queries.

7.2 Future Works

Several promising directions for future research works are presented in this thesis. These range from direct extensions of the research to implementing the ideas in other applications.

In computational geometry, the highest order Voronoi diagram and adjacency graph are the new variants of the Voronoi diagram. This model can be further improved by reducing both construction and storage cost.

In nearest neighbours query processing, further improvement can be made by reducing the cost of finding the cells that contain the query point. Furthermore, the nearest neighbours framework can also be extended to support more queries, such as group k nearest neighbours query, and mobility support.

For reverse nearest neighbours queries, several research directions can be explored. First, to find an index structure with less storage cost. Second, to extend the support for more query types that involve region analysis.

Processing reverse polychromatic queries in polychromatic queries is still a challenge. The CLI index does not give performance advantage over the brute force method as the query process needs to read all cells in the HSVD to answer the query. Therefore, another index structure that can support both polychromatic and hierarchical queries will be a major improvement in polychromatic spatial query processing.

References

Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A. and Renz, M. (2006a). Approximate reverse k-nearest neighbor queries in general metric spaces, *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM '06, ACM, New York, NY, USA, pp. 788–789.

Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A. and Renz, M. (2006b). Efficient reverse k-nearest neighbor search in arbitrary metric spaces, *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, ACM, New York, NY, USA, pp. 515–526.

Achtert, E., Bhm, C., Krger, P., Kunath, P., Pryakhin, A. and Renz, M. (2007). Efficient reverse k-nearest neighbor estimation, *Informatik - Forschung und Entwicklung* **21**: 179–195.

Adhinugraha, K. M., Taniar, D. and Indrawan, M. (2013). Finding reverse nearest neighbors by region, *Concurrency and Computation: Practice and Experience* pp. n/a–n/a.

Adhinugraha, K., Taniar, D., Santiago, M. and Latjuba, D. (2014). Reverse nearest neighbour by region on mobile devices, *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pp. 457–464.

Agarwal, P. K., de Berg, M., Matoušek, J. and Schwarzkopf, O. (1994). Constructing levels in arrangements and higher order voronoi diagrams, *Proceedings of the Tenth Annual Symposium on Computational Geometry*, SCG '94, ACM, New York, NY, USA, pp. 67–75.

Agarwal, P. K., Matouek, J. and Suri, S. (1992). Farthest neighbors, maximum spanning trees and related problems in higher dimensions, *Computational Geometry* **1**(4): 189 – 201.

Agarwal, P., Matouek, J. and Suri, S. (1991). Farthest neighbors, maximum spanning trees and related problems in higher dimensions, *in* F. Dehne, J.-R. Sack and N. Santoro (eds), *Algorithms and Data Structures*, Vol. 519 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 105–116.

Aggarwal, A., Guibas, L., Saxe, J. and Shor, P. (1989). A linear-time algorithm for computing the voronoi diagram of a convex polygon, *Discrete Computational Geometry* **4**(1): 591–604.

Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure, *ACM Comput. Surv.* **23**(3): 345–405.

Aurenhammer, F. and Edelsbrunner, H. (1984). An optimal algorithm for constructing the weighted voronoi diagram in the plane, *Pattern Recognition* **17**(2): 251 – 257.

Aurenhammer, F. and Schwarzkopf, O. (1991). A simple on-line randomized incremental algorithm for computing higher order voronoi diagrams, *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG '91, ACM, New York, NY, USA, pp. 142–151.

Benetis, R., Jensen, C., Karciauskas, G. and Saltenis, S. (2002). Nearest neighbor and reverse nearest neighbor queries for moving objects, *Database Engineering and Applications Symposium, 2002. Proceedings. International*, pp. 44 – 53.

Benetis, R., Jensen, S., Karciauskas, G. and Saltenis, S. (2006). Nearest and reverse nearest neighbor queries for moving objects, *The VLDB Journal* **15**(3): 229–249.

Berg, M. d., Cheong, O., Kreveld, M. v. and Overmars, M. (2008). Computational geometry, algorithms and appliations. third edition.

Bohan, L. and Xiaolin, Q. (2009). Research on reverse nearest neighbor queries using ranked voronoi diagram, *Information Science and Engineering (ICISE), 2009 1st International Conference on*, pp. 951 –955.

Brassel, K. E. and Reif, D. (1979). A procedure to generate thiessen polygons, *Geographical Analysis* **11**(3): 289–303.

Brown, K. Q. (1979). Voronoi diagrams from convex hulls, *Information Processing Letters* **9**(5): 223 – 228.

Cheema, M. A., Lin, X., Zhang, W. and Zhang, Y. (2013). A safe zone based approach for monitoring moving skyline queries, *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, ACM, New York, NY, USA, pp. 275–286.

Cheema, M. A., Lin, X., Zhang, Y., Wang, W. and Zhang, W. (2009). Lazy updates: an efficient technique to continuously monitoring reverse knn, *Proc. VLDB Endow.* **2**(1): 1138–1149.

Cheema, M. A., Zhang, W., Lin, X. and Zhang, Y. (2012). Efficiently processing snapshot and continuous reverse k nearest neighbors queries, *The VLDB Journal* **21**(5): 703–728.

Cheema, M. A., Zhang, W., Lin, X., Zhang, Y. and Li, X. (2012). Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks, *The VLDB Journal* **21**(1): 69–95.

Cheema, M., Lin, X., Wang, W., Zhang, W. and Pei, J. (2010). Probabilistic reverse nearest neighbor queries on uncertain data, *Knowledge and Data Engineering, IEEE Transactions on* **22**(4): 550 –564.

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

Chen, D., Zhou, J. and Le, J. (2006). Reverse nearest neighbor search in peer-to-peer systems, *in* H. Larsen, G. Pasi, D. Ortiz-Arroyo, T. Andreasen and H. Christiansen (eds), *Flexible Query Answering Systems*, Vol. 4027 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 87–96.

Crain, I. K. (1978). The monte-carlo generation of random polygons, *Computers Geosciences* **4**(2): 131 – 141.

Cromley, R. G. and Grogan, D. (1985). A procedure for identifying and storing a thiessen diagram within a convex boundary, *Geographical Analysis* **17**(2): 167–174.

De Berg, M., Dobrindt, K. and Schwarzkopf, O. (1995). On lazy randomized incremental construction, *Discrete & Computational Geometry* **14**(1): 261–286.

Demiryurek, U. and Shahabi, C. (2012). Indexing network voronoi diagrams, *Proceedings of the 17th International Conference on Database Systems for Advanced Applications - Volume Part I*, DASFAA'12, Springer-Verlag, Berlin, Heidelberg, pp. 526–543.

Dwyer, R. (1991). Higher-dimensional voronoi diagrams in linear expected time, *Discrete Computational Geometry* **6**(1): 343–367.

Edelsbrunner, H. and Seidel, R. (1986). Voronoi diagrams and arrangements, *Discrete & Computational Geometry* **1**(1): 25–44.

Emrich, T., Kriegel, H.-P., Kröger, P., Renz, M., Xu, N. and Züfle, A. (2010). Reverse k-nearest neighbor monitoring on mobile objects, *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, ACM, New York, NY, USA, pp. 494–497.

Engel, A. (1998). *Problem Books in Mathematics*, Springer.

Figueroa, K. and Paredes, R. (2009). Approximate direct and reverse nearest neighbor queries, and the k-nearest neighbor graph, *Similarity Search and Applications, 2009. SISAP '09. Second International Workshop on*, pp. 91 –98.

Fort, M. and Sellars, J. A. (2009). Computing generalized higher-order voronoi diagrams on triangulated surfaces, *Applied Mathematics and Computation* **215**(1): 235 – 250.

Fortune, S. (1987). A sweepline algorithm for voronoi diagrams, *Algorithmica* **2**(1-4): 153–174.

Furuta T., Suzuki A., I. K. (2005). The kth nearest network voronoi diagram and its application to districting problem of ambulance systems, *Technical Report No.0501*, Center for Management Studies, Nanzan University.

Green, P. J. and Sibson, R. (1978). Computing dirichlet tessellations in the plane, *The Computer Journal* **21**(2): 168–173.

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

Huang, X., Jensen, C., Lu, H. and altenis, S. (2007). S-grid: A versatile approach to efficient query processing in spatial networks, *in* D. Papadias, D. Zhang and G. Kollios (eds), *Advances in Spatial and Temporal Databases*, Vol. 4605 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 93–111.

Iri, M., Murota, K. and Ohya, T. (1984). A fast voronoi-diagram algorithm with applications to geographical optimization problems, *System Modelling and Optimization*, Vol. 59 of *Lecture Notes in Control and Information Sciences*, Springer Berlin Heidelberg, pp. 273–288.

Kang, J., Mokbel, M., Shekhar, S., Xia, T. and Zhang, D. (2007). Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors, *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pp. 806 –815.

Katayama, N. and Satoh, S. (1997). The sr-tree: An index structure for high-dimensional nearest neighbor queries, *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, ACM, New York, NY, USA, pp. 369–380.

Katoh, N. and Iwano, K. (1992). Finding k farthest pairs and k closest/farthest bichromatic pairs for points in the plane, *Proceedings of the Eighth Annual Symposium on Computational Geometry*, SCG '92, ACM, New York, NY, USA, pp. 320–329.

Kolahdouzan, M. and Shahabi, C. (2004). Voronoi-based k nearest neighbor search for spatial network databases, *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, VLDB Endowment, pp. 840–851.

Korn, F. and Muthukrishnan, S. (2000). Influence sets based on reverse nearest neighbor queries, *SIGMOD Rec.* **29**(2): 201–212.

Krussel, J. W. and Schaudt, B. F. (1994). A note on higher order voronoi diagrams, *Nordic J. of Computing* **1**(2): 268–272.

Kumar, Y., Janardan, R. and Gupta, P. (2008). Efficient algorithms for reverse proximity query problems, *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, GIS '08, ACM, New York, NY, USA, pp. 39:1–39:10.

Lee, D.-T. (1982). On k-nearest neighbor voronoi diagrams in the plane, *Computers, IEEE Transactions on* **C-31**(6): 478–487.

Lee, D. T. and R. L. Drysdale, I. (1981). Generalization of voronoi diagrams in the plane, *SIAM Journal on Computing* **10**(1): 73–87.

Lian, X. and Chen, L. (2008). Monochromatic and bichromatic reverse skyline search over uncertain databases, *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, ACM, New York, NY, USA, pp. 213–226.

Liu, J., Chen, H., Furuse, K. and Kitagawa, H. (2010). An efficient algorithm for reverse furthest neighbors query with metric index, *Database and Expert Systems Applications*, Springer, pp. 437–451.

Liu, J., Chen, H., Furuse, K. and Kitagawa, H. (2012). An efficient algorithm for arbitrary reverse furthest neighbor queries, *in* Q. Sheng, G. Wang, C. Jensen and G. Xu (eds), *Web Technologies and Applications*, Vol. 7235 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 60–72.

Liu, W. and Yuan, Y. (2013). New ideas for fn/rfn queries based nearest voronoi diagram, *in* Z. Yin, L. Pan and X. Fang (eds), *Proceedings of The Eighth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), 2013*, Vol. 212 of *Advances in Intelligent Systems and Computing*, Springer Berlin Heidelberg, pp. 917–927.

Lu, J., Lu, Y. and Cong, G. (2011). Reverse spatial and textual k nearest neighbor search, *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, ACM, New York, NY, USA, pp. 349–360.

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

Nghiem, T. P., Maulana, K., Waluyo, A. B., Green, D. and Taniar, D. (2013). Bichromatic reverse nearest neighbors in mobile peer-to-peer networks, *PerCom*, pp. 160–165.

Nghiem, T. P. and Waluyo, A. B. (2011). A pure p2p paradigm for query processing in mobile ad-hoc networks, *MoMM*, pp. 182–189.

Nghiem, T. P., Waluyo, A. B. and Taniar, D. (2013). A pure peer-to-peer approach for knn query processing in mobile ad hoc networks, *Personal and Ubiquitous Computing* **17**(5): 973–985.

Oishi, Y. and Sugihara, K. (1995). Topology-oriented divide-and-conquer algorithm for voronoi diagrams, *Graphical Models and Image Processing* **57**(4): 303 – 314.

Okabe, A., Boots, B., Sugihara, K. and Chiu, S. (2009). *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, Wiley Series in Probability and Statistics, Wiley.

Okabe, A., Satoh, T., Furuta, T., Suzuki, A. and Okano, K. (2008). Generalized network voronoi diagrams: Concepts, computational methods, and applications, *International Journal of Geographical Information Science* **22**(9): 965–994.

O'Rourke, J., Rao Kosaraju, S. and Megiddo, N. (1986). Computing circular separability, *Discrete Computational Geometry* **1**(1): 105–113.

Papadopoulos, A. and Manolopoulos, Y. (1997). Performance of nearest neighbor queries in r-trees, *in* F. Afrati and P. Kolaitis (eds), *Database Theory ICDT '97*, Vol. 1186 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 394–408.

Quine, M. and Watson, D. (1984). Radial generation of n-dimensional poisson procesess, *Journal of Applied Probability* pp. 548–557.

Rhynsburger, D. (1973). Analytic delineation of thiessen polygons*, *Geographical Analysis* **5**(2): 133–144.

Roussopoulos, N., Kelley, S. and Vincent, F. (1995). Nearest neighbor queries, *SIGMOD Rec.* **24**(2): 71–79.

Safar, M. (2005). K nearest neighbor search in navigation systems, *Mobile Information Systems* **1**(3): 207–224.

Safar, M., Ibrahimi, D. and Taniar, D. (2009). Voronoi-based reverse nearest neighbor query processing on spatial networks, *Multimedia Systems* **15**: 295–308. 10.1007/s00530-009-0167-z.

Shamos, M. I. and Hoey, D. (1975a). Closest-point problems, *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pp. 151–162.

Shamos, M. I. and Hoey, D. (1975b). Closest-point problems, *Foundations of Computer Science, 1975., 16th Annual Symposium on*, IEEE, pp. 151–162.

Sharifzadeh, M. and Shahabi, C. (2009). Approximate voronoi cell computation on spatial data streams, *The VLDB Journal* **18**(1): 57–75.

Sharifzadeh, M. and Shahabi, C. (2010). Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries, *Proc. VLDB Endow.* **3**(1-2): 1231–1242.

Sibson, R. (1978). Locally equiangular triangulations, *The Computer Journal* **21**(3): 243–245.

Singh, A., Ferhatosmanoglu, H. and Tosun, A. c. (2003). High dimensional reverse nearest neighbor queries, *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, ACM, New York, NY, USA, pp. 91–98.

Slimani, H., Najjar, F. and Slimani, Y. (2011). Voronoi-neighboring regions tree for efficient processing of location dependent queries, *International Journal of Advanced Science and Technology* **33**: 101–119.

Smid, M. (1995). *Closest point problems in computational geometry*, Citeseer.

Stanoi, I., Agrawal, D. and Abbadi, A. E. (2000). Reverse nearest neighbor queries for dynamic databases, *In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pp. 44–53.

Stanoi, I., Riedewald, M., Agrawal, D. and Abbadi, A. E. (2001). Discovery of influence sets in frequently updated databases, *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 99–108.

Sugihara, K. and Iri, M. (1989). Two design principles of geometric algorithms in finite-precision arithmetic, *Applied Mathematics Letters* **2**(2): 203 – 206.

Sugihara, K. and Iri, M. (1992). Construction of the voronoi diagram for 'one million' generators in single-precision arithmetic, *Proceedings of the IEEE* **80**(9): 1471–1484.

Supowit, K. J. (1990). New techniques for some dynamic closest-point and farthest-point problems, *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 84–90.

Tao, Y., Papadias, D. and Lian, X. (2004). Reverse knn search in arbitrary dimensionality, *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, VLDB Endowment, pp. 744–755.

Tran, Q. T., Taniar, D. and Safar, M. (2010). Bichromatic reverse nearest-neighbor search in mobile systems, *Systems Journal, IEEE* **4**(2): 230 –242.

Tran, Q., Taniar, D. and Safar, M. (2009). Reverse k nearest neighbor and reverse farthest neighbor search on spatial networks, *Transactions on Large-Scale Data- and Knowledge-Centered Systems I*, Vol. 5740 of *Lecture Notes in Computer Science*, Springer Berlin - Heidelberg, pp. 353–372.

Wang, W., Yang, J. and Muntz, R. (2000). Pk-tree: A spatial index structure for high dimensional point data, *in* K. Tanaka, S. Ghandeharizadeh and Y. Kambayashi (eds), *Information Organization and Databases*, Vol. 579 of *The Springer International Series in Engineering and Computer Science*, Springer US, pp. 281–293.

Wang, X., Luo, Y., Yu, L. and Xu, Z. (2005). Pk+ tree: An improved spatial index structure of pk tree, *in* V. Sunderam, G. van Albada, P. Sloot and J. Dongarra (eds), *Computational Science ICCS 2005*, Vol. 3516 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 511–514.

Wang, Y., Lee, K. and Lee, I. (2011). Representing ordered order-k voronoi diagrams, *Advanced Information Management and Service (ICIPM), 2011 7th International Conference on*, pp. 71–75.

Wong, R. C.-W., Özsu, M. T., Yu, P. S., Fu, A. W.-C. and Liu, L. (2009). Efficient method for maximizing bichromatic reverse nearest neighbor, *Proc. VLDB Endow.* **2**(1): 1126–1137.

Wu, W., Yang, F., Chan, C. Y. and Tan, K.-L. (2008a). Continuous reverse k-nearest-neighbor monitoring, *Mobile Data Management, 2008. MDM '08. 9th International Conference on*, pp. 132 –139.

Wu, W., Yang, F., Chan, C.-Y. and Tan, K.-L. (2008b). Finch: evaluating reverse k-nearest-neighbor queries on location data, *Proc. VLDB Endow.* **1**(1): 1056–1067.

Xia, T. and Zhang, D. (2006). Continuous reverse nearest neighbor monitoring, *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, p. 77.

Xuan, K., Zhao, G., Taniar, D., Rahayu, W., Safar, M. and Srinivasan, B. (2011). Voronoi-based range and continuous range query processing in mobile databases, *Journal of Computer and System Sciences* **77**(4): 637 – 651. JCSS IEEE AINA 2009.

Xuan, K., Zhao, G., Taniar, D., Safar, M. and Srinivasan, B. (2011). Voronoi-based multi-level range search in mobile navigation, *Multimedia Tools Appl.* **53**(2): 459–479.

Xuan, K., Zhao, G., Taniar, D., Srinivasan, B., Safar, M. and Gavrilova, M. L. (2009). Continuous range search based on network voronoi diagram, *IJGUC* **1**(4): 328–335.

Yan, D., Zhao, Z. and Ng, W. (2012). Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces, *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, ACM, New York, NY, USA, pp. 942–951.

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

Yao, B., Li, F. and Kumar, P. (2009). Reverse furthest neighbors in spatial databases, *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on*, pp. 664–675.

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

Zhao, G., Xuan, K., Rahayu, W., Taniar, D., Safar, M., Gavrilova, M. L. and Srinivasan, B. (2011). Voronoi-based continuous k nearest neighbor search in mobile navigation, *IEEE Transactions on Industrial Electronics* **58**(6): 2247–2257.

Zhao, G., Xuan, K. and Taniar, D. (2013). Path knn query processing in mobile systems, *IEEE Transactions on Industrial Electronics* **60**(3): 1099–1107.

Zhao, G., Xuan, K., Taniar, D., Safar, M., Gavrilova, M. and Srinivasan, B. (2009). Multiple object types knn search using network voronoi diagram, *Computational Science and Its Applications–ICCSA 2009*, Springer, pp. 819–834.

Zhao, G., Xuan, K., Taniar, D. and Srinivasan, B. (2010). Lookahead continuous knn mobile query processing, *Comput. Syst. Sci. Eng.* **25**(3).

Zhu, J., Kan, B., Liu, Y., Wang, T., Pan, L. and Liu, D. (2014). A probabilistic group reverse k-nearest-neighbor query in sensor networks, *in* X. Wang, L. Cui and Z. Guo (eds), *Advanced Technologies in Ad Hoc and Sensor Networks*, Vol. 295 of *Lecture Notes in Electrical Engineering*, Springer Berlin Heidelberg, pp. 121–130.