A Study on Intelligent Area Selection Query

（知的領域選択問合せに関する研究）

by

ANNISA

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF ENGINEERING

In The Graduate School of Engineering

Supervisor: YASUHIKO MORIMOTO, Assoc. Prof

HIROSHIMA UNIVERSITY

September, 2017

Dissertation Summary

Problem of selecting good location is very important in the services and business field, either for customers or for land owners. User/customer uses location selection query to find good location to buy, to rent, or to visit. From user/customers perspective, a location is more valuable if it is close to desirable facilities that can bring profits and benefits for the location, and it is far from undesirable facilities that can reduce profits and bring unfavorable effects for the location. On the other hand, location selection for land owner aims to find other locations which are as good as his/her owned locations. One of the important criteria for comparing his/her location to others is the locations distance to desirable and undesirable facilities.

Skyline query is a well-known method for selecting small number of data objects, and it also has been applied in the location selection problem. In some situation, there are some candidate points, which are for example vacant rental rooms, on a map. In such cases, we can utilize skyline query to select a preferable point. However, in some real world situations, we cannot assume there are candidate points for the selection problem on a map. For example, assume a businessman wants to build a new supermarket if there is a good vacant area. The businessman may also want to take over a building that is located in a good area at any cost. In such situation, the candidate points are not given and the businessman has to find a good location in an area on the map. Two-dimensional

area is much more complicated than a point; therefore, the area selection problem for two-dimensional area is challenging and important. However, previous skyline algorithms for location selection problem only consider zero dimensional objects and are based on the assumption that there will always be some candidate points to be selected.

In this dissertation, we introduced two new variants of skyline query problem, Area Skyline Query: Skyline query for selecting spatial area objects from customers perspective and Reverse Area Skyline Query: Skyline query for selecting spatial area objects from land owners perspective. To answer area skyline queries, first we developed Unfixed-shape Area Skyline (UASky) algorithm in our feasibility study. To find area skyline in UASky, we divide the query area by overlaying all Voronoi diagrams of all facility types to generate the unfixed-shape disjoint areas, and calculate the minimum (min) and maximum (max) distance from an area to the closest facility of each type. In order to calculate these distances efficiently, we first compute distance from each of vertexes that encloses an area. Note that we can efficiently compute the closest facility from the vertexes by using Voronoi diagram. Also, note that one vertex is included by more than one area. Then, after computing min distance from vertexes, we calculate min-max distance for each area and record them in a table called Minmax table. Given two unfixed-shape areas, a and a', we say area a dominates area a' if and only if max distance of a is smaller or equal to min distance of a' for all facility types. Skyline query for areas selects all non-dominated areas from the set of the disjoint areas. Based on the extensive experiments, UASky is affected by the number of facility types, as well as the number of objects for each facility. The drawback of UASky is that it selects relatively many areas as skylines, since a large area is likely to be selected as a skyline because they have large max distance.

One countermeasure for UASkys drawback is to divide a large area into smaller areas. We used grid data structure to divide query area and proposed an efficient and practical solution to the area skyline query problem called Grid based Area Skyline (GASky) algorithm. GASky first divides query area into s number of grids. Then, it finds non-dominant grids as the result. Comprehensive experiments show that the processing time of GASky increases with the increase of the number of grids, facility types, and number of objects. The experiment also shows that we can decrease the ratio of skyline area by increasing the number of grids. Thus, higher number of grid means smaller size of each disjoint area, which in turn will decrease the number of skyline areas. By applying grid data structure, the GASky can control the number of area skyline by changing the number of grids. In actual usage scenario, if a user prefers selective areas, she/he had better increase the number of grid, which tends to reduce the ratio of skyline areas.

The author has considered another area skyline query problem called ”reverse area skyline query”, which are based on land owners' perspective. Here, we combine GASky method to compute min-max distance for each grid and a state-of-the-art reverse area skyline algorithm using global skyline concept for two-dimensional area. We extend conventional global skyline concept so it can be applied to two-dimensional area. This query is very important for location selection in business or land owners perspective. One of the important applications of reverse area skyline query is selecting promising buyers of the area, since reverse area skyline query may give clues to the owner of the area in finding who will be interested in the area. Furthermore, it also may help to predict what type of business that would be suitable for the area considering the type of business that has already exist in the reverse area skylines. To answer reverse area

skyline problem, we proposed Reverse Area Skyline (RASky) algorithm. Comprehensive experiments are conducted to show the effectiveness and efficiency of the proposed algorithm.

To summarize, this dissertation addresses two new skyline queries in the location selection problem for two dimensional areas: skyline queries for selecting spatial areas from customers perspective and owners perspective. The outline of this thesis is organized like in the following passage. Chapter 1 presents the introduction of this dissertation. Chapter 2 reviews about related works: skyline query and its variants, and also some issues in skyline for location selection. We reported our feasibility study of area skyline query problem in Chapter 3. In this chapter, we present our starter algorithm, UASky. In Chapter 4 we proposed our efficient algorithm, GASky and compare both algorithms, UASky and GASky, based on some related parameters. We find that GASky outperforms UASky according to complexity computation and experiment results. In Chapter 5, we introduced another new skyline query problem, reverse area skyline query, which is area selection problem based on owners perspective. In this chapter, we presented a new definition of dynamic area skyline and proposed RASky algorithm to answer reverse area skyline query problem. Our extensive experimental study confirms that GASky and RASky algorithms are able to find reasonable number of desirable skyline areas and can help users to find good locations. Finally in Chapter 6, we conclude our study and present some future directions

Acknowledgements

First of all, I want to express my sincere gratitude to almighty Allah, the most Beneficent, the most Gracious and the most Merciful, for giving me the opportunity to successfully complete my research work.

I would like to express my special appreciation and thanks to my advisor Assoc. Prof. Yasuhiko Morimoto. He has been a tremendous mentor for me. I would like to thank him for encouraging my research, for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. He is my inspiration to be a better scientist and researcher.

I would like to thank my committee members, Professor Tsukasa Hirashima, Professor Takio Kurita and Assoc. Prof. Sayaka Kamei, for letting my defense be an enjoyable moment, and also for their brilliant comments and suggestions that enrich my knowledge. I would like to thank to all the faculty members of the Department of Information Engineering, Hiroshima University, especially to Ms. Kazue Nakao, who helped me so much from the beginning until the end of my study year.

My gratefulness and appreciation to Indonesian Government, especially for DG-RTHE Department and its team, for providing DG-RSTHE Scholarship to pursue my study. My deep gratefulness also for Bogor Agricultural University, The Dean of MIPA

faculty, Ms, Sri Nurdiati, Phd., The Head of Computer Science Department, Mr. Agus Buono, Phd, and all lecturers, staffs, and students in Computer Science Departement, Bogor Agricultural University. Special thanks also to all my lab mates for their valuable discussion, and all my friends in Japan and Indonesia, for warm friendship, advice, co-operation, and encouragement.

I am deeply grateful to my children, my parents, and all my family members for their prayers and constant love for me. Words cannot express how grateful I am to have you all in my life. Last but not least, I would like to express my gratitude to my beloved husband, Choirul Hafidz Akhmadi for his wonderful love and immeasurable sacrifices. He always understand me and believe in me no matter how hard it is. Thank you for give me faith, I knew I will never get through this without you.

Finally, I hope this work will give significant contributions and advantages, especially for data management research in Japan and Indonesia.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Selecting locations is a very important subproblem of data mining. In all fields, like business, tourism, or even in selecting restaurant for dinner, choosing the best location will have a direct impact on cost, time, and effort. In some cases, when the location has been chosen, it is difficult to be canceled because doing so and then finding another location will require more cost, time, and effort. Therefore it is very important to decide the right location in the first place. In business field, a company should consider proximity to sites that usually attract many people, such as retail centers, stations, customers' house, office complexes, and hotel and entertainment centers as valuable criteria for selecting a location. That is because those sites can bring profit and benefit for the location. In addition, a company should also review potential competitor existence or sites that can reduce profit and bring unfavorable effect on the location, such as garbage dumps, pollution sources, high-crime areas, etc. We call sites that we prefer to be close to our selected locations as desirable facilities, and otherwise as undesirable facilites. In general, a location that is close to some desirable facilities and far from undesirable facilities is called a better location. For example:

- In the business field: a property company would like to build a new apartment in a new region. The company has some candidate locations to be selected. To attract customers, the apartment should be in an area that is close to train stations, shopping centers, and schools, and far from open landfill.

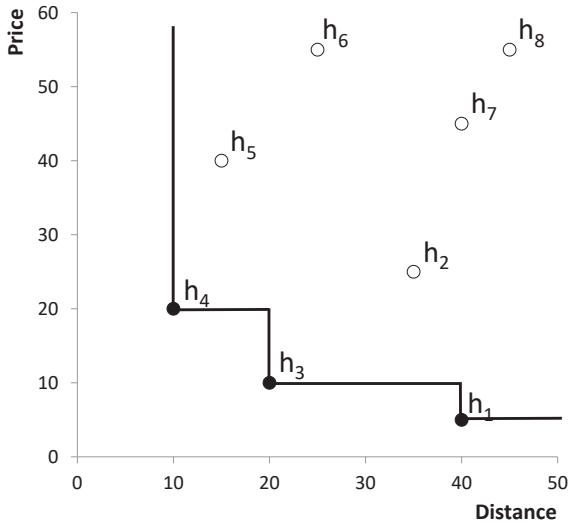
- In the travel planning: when planning trip to a new area or country, a traveler would like to stay in a hotel that will be convenient in location and cost. He/She has some candidate hotels to be selected. He/She would like to find which hotel that is close to attraction sites, train stations, and convenience stores and far from crime areas and polluted areas.

In above cases, we consider a problem for selecting spatial "points" in a map from a set of candidate "points" given by user that satisfy the user's criteria which is close to desirable facilities and far from undesirable facilities. To answer the problem for selecting spatial "points", we can apply the idea of the skyline query.

In database management system, skyline query [6] is a well known method for selecting small number of data objects. It selects objects that are not dominated by another object. Figure 1.1 shows a typical example of skyline. Consider a typical online booking system. A user can select a hotel from the list in Figure 1.1(a) based on her/his preference on the price and distance of the hotel to the beach. In this example, we assume that smaller value is better for each attribute. In this situation,$\{h_1, h_3, h_4\}$ are skyline objects because they are not dominated by another object. It means no other objects have smaller value in both two dimensions compared to $\{h_1, h_3, h_4\}$. Other objects ($\{h_7, h_8, h_2, h_6, h_5\}$) are dominated by h_4, since h_4 has smaller value in both dimensions. Figure 1.1(b) shows skyline hotels from the given hotel list. Using the skyline query, we can easily answer the problem for selecting spatial "points" in a map.

ID	Distance	Price
h1	40	5
h2	35	25
h3	20	10
h4	10	20
h5	15	40
h6	25	55
h7	40	45
h8	45	55

(a)

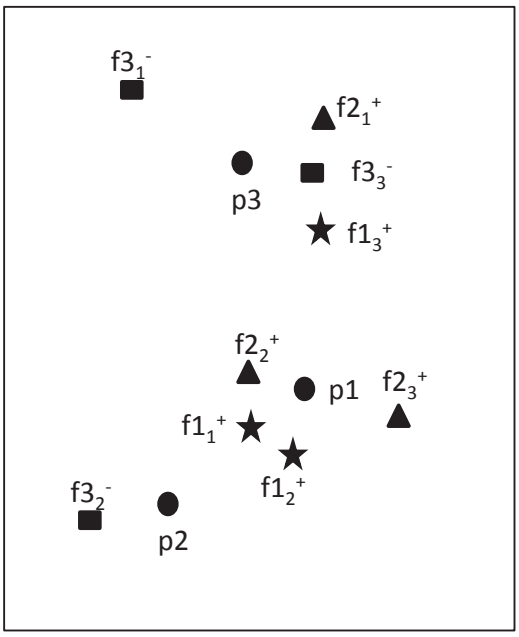


(b)

Figure 1.1: Conventional Skyline

Figure 1.2(a) illustrates the skyline query for spatial points. Consider a situation that a businessman wants to open a new supermarket. She/He would like to open the supermarket in a building for rent. She/He prefers a building that is close to desirable facilities such as a bus/train station, a university, and so forth, but it should be far enough from some undesirable facilities, such as a similar supermarket (a potential competitor).

Let P be a set of spatial points, which are buildings for rent, to be chosen. Let F be a set of facilities, which can be categorized into m types, $F1$, $F2$, ..., Fm. Each type is classified into desirable or undesirable. We annotate "+" mark on the facility symbol of desirable facilities like F^+, while we annotate "−" mark on undesirable ones like F^-. Points p_1, p_2, and p_3 ($\in P$) in this figure are buildings for rent. Points illustrated with star symbol, $F1^+ = \{f1_1^+, f1_2^+, ..., f1_{m_1}^+\} \in F$, represent locations of universities, which are desirable facilities. Another desirable facilities are stations, which are illustrated with triangle symbol, $F2^+ = \{f2_1^+, f2_2^+, ..., f2_{m_2}^+\} \in F$. Points with square symbol, $F3^- = \{f3_1^-, f3_2^-, ..., f3_{m_3}^-\} \in F$, represent competitors' supermarkets, which are undesirable



a) Map on Example 1

Point	F1$^+$	F2$^+$	F3$^-$
p1	4	3	-10
p2	5	6	-2
p3	4	4	-6

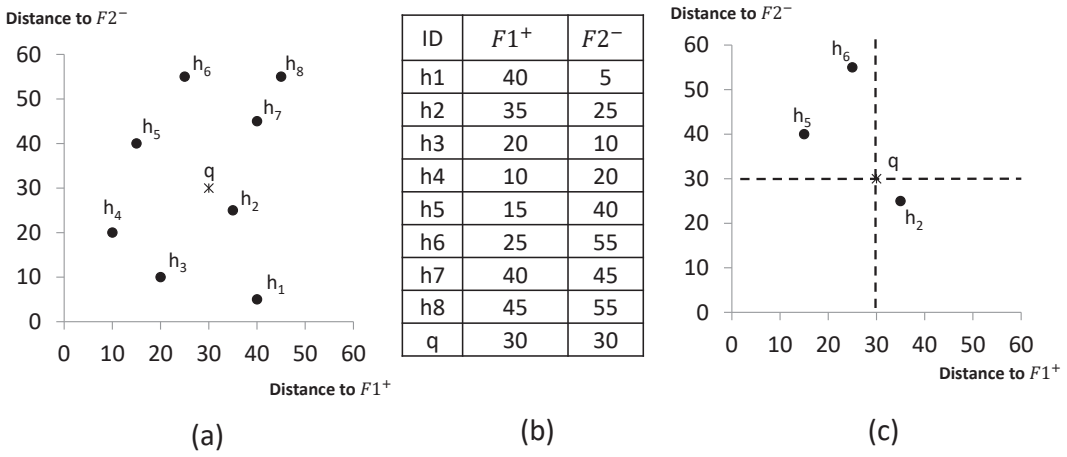
b) Data Table on Example 1

Figure 1.2: The skyline queries for spatial points' illustration

facilities.

Based on the map of Figure 1.2(a), we calculate a table as in Figure 1.2(b). In the table, we record distance from a building to the closest facility of each of $F1^+$, $F2^+$, and $F3^-$ types. For example, the closest university ($F1^+$ facility (star)) from p_1 is $f1_2^+$ and the distance is 4. Similarly, the closest station ($F2^+$ facility (triangle)) from p_1 is $f2_2^+$ and the distance is 3. The closest competitor ($F3^-$ facility (square)) from p_1 is $f3_3^-$ and the distance is 10. We multiply -1 to each distance value of undesirable facilities F^- so that we can say that smaller value is better in each of the attributes. In Figure 1.2, p_1 dominates p_2 and p_3 since p_1 is located closer to desirable facilities and farther to undesirable facilities. Therefore, skyline query for the spatial points returns p_1.

Choosing good location in a map is not only important for businessman or a tourist as consumer of location selection (a building for rent or a hotel in a map). It is also



ID	$F1^+$	$F2^-$
h1	40	5
h2	35	25
h3	20	10
h4	10	20
h5	15	40
h6	25	55
h7	40	45
h8	45	55
q	30	30

Figure 1.3: Reverse Skyline Query Example

important for the owner of a building for rent or the owner of a hotel. Let us consider a hotel dataset in Figure 1.3(a). Assume that a businessman wants to build a new hotel q, which its details are represented as its distances to desirable ($F1^+$) and undesirable facilities ($F2^-$). He/She would like to know which other hotels think that q is interesting, because customer who choose those hotels might also be interested in q. Reverse skyline query [13] (RSQ) is a variant of skyline query that answers this kind of problem. RSQ(q) find a set of points that its distance to q is not dominated by its distance to another point. Using a table like in Figure 1.3(b), we can easily find reverse skyline points of q using reverse skyline algorithm [13]. Figure 1.3(c) shows that h_2, h_5, and h_6 are RSQ of q. For h_2, its distances to q is the smallest distance in x-axis and y-axis compared to its distances to other points. While for h_5, its distance to q is the smallest distance in y-axis, and h_6's distance to q is the smallest distance in x-axis. Intuitively, users who are interested in h_2, h_5, and h_6 hotels may also be interested in q. While skyline query retrieves a set of skyline objects based on customer's perspective, reverse skyline query

retrieves a set of skyline objects based on business owner's perspective.

1.1 Motivation

Notice that to perform the point location selection query, there are some candidate points for example vacant rent rooms, on a map. In such cases, we can utilize skyline query to select a preferable point. However, in some real world situations, we cannot assume there are candidate locations for the selection problem on a map. For example, a businessman wants to build a new supermarket if there is a good vacant area. The businessman may also want to take over a building that is located in a good area at any cost. In such situation, the candidate points are not given and the businessman has to find a good location in an area on the map. In other words, she/he has to find two-dimensional area on the map. In the owner's perspective, their owned object/property in a map is not always a spatial point, it might also in the shape of area, which is two-dimensional area. Two-dimensional area is much more complicated than a point; therefore, the area selection problem for two-dimensional area is challenging and important. However, previous skyline algorithms for location selection problem only consider zero dimensional objects and based on the assumption that there are always some candidate points on a map to be selected.

1.2 Contributions

All previous skyline algorithms are only suitable for spatial point location selection, as later discussed in Chapter 2). Therefore this thesis addresses two new skyline queries in the location selection problem for two dimensional areas: skyline queries for selecting

spatial area objects from customers perspective and owners perspective.

1. **Skyline queries for selecting spatial area objects from customers perspective**

   Let A be a rectangular target area, let $F = \{F1, ...Fm\}$ be a set of facility types, which can be categorized into m types and each type is classified as desirable or undesirable facility. In this thesis we introduce a new variant of skyline query, called area skyline query, which divides A into smaller two-dimensional disjoint areas and select a set of areas which has closer distance from desirable facilities and farther distance from undesirable facilities. To answer area skyline query, first we conducted feasibility study on area selection problem. In this feasibility study, we presented an algorithm called Unfixed-shape area skyline (UASky) algorithm. We conducted extensive experiments to UASky and discovered its limitation. In order to overcome the drawback of UASky, we proposed Grid-based area skyline (GASky) algorithm. Comprehensive experiments showed that GASky has better performance than UASky and in the same time resolves the limitation of UASky.

2. **Skyline query for selecting spatial area objects from owners perspective**

   Not only from customer's perspective, in this thesis we also extend area skyline query problem from land owner's perspective, called reverse area skyline query problem. Let A be a rectangular target area in which there are spatial objects. Each spatial object can be categorized into one of m facility types. Let Fk be a set of type k ($k = 1, ...m$) objects, which are $Fk = \{fk_1, fk_2, ..., fk_{n_k}\}$ where n_k is the number of objects of the type k facility. Let g is an area in A belongs to the owner, and g has distances to each closest facility type as its attributes. Reverse area skyline query selects a set of disjoint areas in A which are as preferable as g. By using

the idea of "reverse skyline", we developed Reverse area skyline (RASky) algorithm to answer reverse area skyline query problem. Comprehensive experiments are conducted to show the effectiveness and efficiency of our proposed algorithm.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 reviews about related works which are skyline query and its variants, and also some issues in skyline for location selection. We report our feasibility study of area skyline query problem in Chapter 3. In this chapter we present our starter algorithm, UASky. In Chapter 4, we propose our efficient algorithm, GASky, and compare both UASky and GASky algorithms based on some related parameters. We found that GASky outperform UASky according to complexity computation and experiment results. In Chapter 5, we introduce a new skyline query called reverse area skyline query, which is area selection problem based on property or land owner's perspective. In this chapter, we present a new definition of dynamic area skyline and propose RASky algorithm to answer reverse area skyline query problem. Our extensive experiments confirm that GASky and RASky algorithms are able to find reasonable number of desirable skyline areas and can help users to find good locations. Finally in Chapter 6, we conclude our study and present some future directions.

Chapter 2

Related Works

In this chapter, we elaborate related works in skyline query, skyline query related to location selection problems, and reverse skyline query. In Section 2.1, first we formulate the basic concepts of the skyline query, address the importance of skyline query in information filtering, and describe examples of skyline query algorithms. In Section 2.2 we discuss important skyline query variants on selecting spatial point objects. Finally in Section 2.3, we present reverse skyline query and some important concepts such as dynamic skyline and global skyline. We also describe the state-of-the-art reverse skyline algorithms in this section.

2.1 Skyline Query

2.1.1 Problem Definition

Let DB be a n-dimensional database, and $D = \{d_1, ...d_n\}$ are n attributes of DB, $P = \{p_1, ...p_r\}$ are r tuples (objects) of DB, and $p_i.d_m$ is the value of p_i in m dimension.

Definition 2.1.1 *(Dominance). Object p_i is said to dominate another object p_j (denoted as $p_i \prec p_j$) if $\exists\ 1 \leq m \leq n$ $p_i.d_m$ is better than $p_j.d_m$, and $\forall 1 \leq m \leq n, p_i.d_m$ is not worse than $p_j.d_m$. Skyline objects are a set of objects in P, each of which is not dominated by another object.*

2.1.2 The Applications of Skyline

Skyline query is very useful for user to select a few interesting objects from huge amount of objects without concern there is a better object that is not included in the result. This result contains smaller number of non-dominated objects, so user can easily select some of the objects in the list based on his/her further preference. For example while selecting a house in a housing company, we have to face a huge number of houses instead of just three or five houses. Thus, it is difficult for us to select best houses based on the price, quality of building material, and distance to the main street, in a short time. Similar example is selecting hotels in an online booking system or selecting products like cameras, cell phones, computers, etc in e-commerce sites. There are so many number of products which makes it impossible to compare all of them manually. Using skyline query we can remove large number of products whose features are worse than any other products and generate manageable number of non-dominated products for user's further selection. Consider selecting hotel example in Figure 1.1. Skyline query reduces the hotel list from 8 hotels in Figure 1.1(a) to 3 hotels $\{h_1, h_3, h_4\}$.

2.1.3 The Skyline Algorithms

Borzsonyi et al. first introduced the skyline operator over large databases and proposed three algorithms: *Block-Nested-Loops* (BNL), *Divide-and-Conquer* (D&C), and

B-tree-based schemes [6]. *Sort-Filter-Skyline* (SFS), which improves BNL by presorting, was proposed by Chomicki et al. as a variant of BNL [10]. Two progressive methods Bitmap and Index for computing skyline have been proposed by Tan et al., which improve previous algorithm [38]. Currently, the most efficient method in computing skyline is *Branch-and-Bound Skyline* (BBS), proposed by Papadias et al., which is a progressive algorithm based on the *Best First-Nearest Neighbor (BF-NN)* algorithm [31]. All of the proposed algorithms for skyline query processing are categorized in two categories: index-based algorithms and non-index-based algorithms. BNL, D&C, and Bitmap are examples of non-index-based skyline algorithms, while B-tree based schema, BF-NN, and BBS are examples of index-based skyline algorithms. In this section, we will describe about BNL as the example of non-index-based skyline algorithms, and BBS as the example of index-based skyline algorithms.

Block Nested Loop (BNL) Algorithm

Comparing any object to other objects is a naive way to compute skyline. Although it is simple, this method is definitely not efficient, since we have to compare n^2 times, where n is the number of objects in dataset. The ideas of BNL [6] is reducing object's comparison by maintaining a window of non-dominated objects in main memory. BNL also uses a temporary file to store candidate objects when the window is full. At first iteration when window is still empty, the first object will be automatically inserted into the window. Then, as iteration continues there are three cases which can occur for the next object o:

1. if o is dominated by another object in the window, then o will be discarded.

2. if o dominates some objects in the window, then those objects will be discarded,

and o will be inserted to the window.

3. if o is not dominated by and also not dominates other object in the window, then o will be inserted to the window.

After all of the data objects are processed, the candidate objects in the window which are processed when the temporary file was empty are output as skyline. Otherwise, compare other objects in window with objects in temporary file using the same way, and output non-dominated objects as skyline. Consider a dataset in Figure 1.1(a). We compute this dataset using BNL with window size set to 2, and process it sequentially based on the subscript number of data. At first, h_1 is inserted into empty window, followed by h_2. Since h_2 is not dominated by h_1, h_2 is also inserted into window. When h_3 is processed, h_2 is dominated by h_3, so h_2 is discarded from window and h_3 is inserted. Temporary file is built when h_4 processed. Next h_5 is inserted to temporary file since it is not dominated by h_1 and h_3 in window. When h_6, h_7, and h_8 is processed, all three of them are discarded since they are dominated by h_1 and h_3. At the end of iteration, there are h_1 and h_3 in window, and h_4 and h_5 in temporary file. Since h_1 and h_3 are processed when temporary file was empty, then both of them are member of skyline. BNL then processed the temporary file using the same way, and output h_4 as member of skyline. Finally, the skyline objects are h_1, h_3, and h_4 as illustrated in Figure 1.1(b).

Branch and Bound Skyline (BBS)

BBS [31] is based on nearest neighbor search [33]. In this section, we will explain BBS using R-tree to partition the dataset. Every leaf/intermediate entry is executed based on their mindist to the query point in ascending order. Minimum distance (mindist) of object from the origin is equal to the sum of object's coordinates and the mindist of

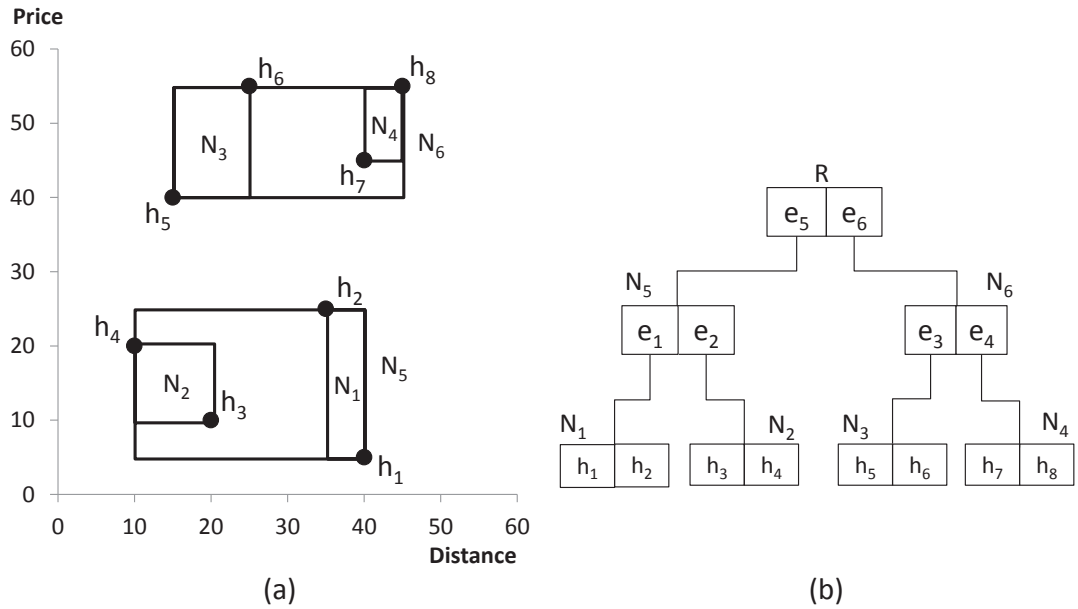


Figure 2.1: Minimum Bounding Rectangle (a) and R-tree (b)

a MBR (i.e., intermediate entry) is equal to the mindist of its lower-left corner point. BBS uses the best-first search to process the R-tree. Figure 2.1 illustrates minimum bounding rectangle (MBR) and R-tree for example dataset in Figure 1.1(a). First, BBS starts from the root node, then (e_5,e_6) are inserted to the heap H. The entry with smallest mindist to origin (e_5) is then expanded, thus removes it from the heap and enheaps its children, so the members of heap are (e_2,e_1,e_6). The next node to be visited is (e_2), so e_2 is removed from heap, and its children are inserted. Now, the heap content are (h_3,h_4,e_6,e_1). Continue to the next node to be visited, h_3. Since it is not an intermediate node, h_3 would become the first skyline object, followed by h_4 which is also a skyline object. Next step is expanding e_1, and inserting h_1 and h_2 into heap. Note that after h_1

Table 2.1: Heap Contents

Description	*HeapContents*	*Skyline*
access root	$< e_5, 15 >< e_6, 55 >$	$\emptyset$
expand e_5	$< e_2, 20 >< e_1, 40 >< e_6, 55 >$	$\emptyset$
expand e_2	$< \mathbf{h_3}, \mathbf{30} >< \mathbf{h_4}, \mathbf{30} > < e_1, 40 >< e_6, 55 >$	$\{h_3, h_4\}$
expand e_1	$< \mathbf{h_1}, \mathbf{45} > \sout{< e_6, 55 >< h_2, 60 >}$	$\{h_3, h_4, h_1\}$

is put into skyline list, we can discard e_6 and h_2, since e_6 is dominated by h_4 and h_2 is dominated by h_4 and h_3. Finally BBS output h_3, h_4 and h_1 as skyline objects. Table 2.1 illustrates heap contents during BBS computation.

2.1.4 Other Related Works in Skyline Queries

Beside the conventional skyline queries, recently different aspects of skyline have been investigated. Dynamic skyline query (DSQ) [31] is one of the variation of conventional skyline query explained above. Instead of static position of query, the query location in dynamic skyline query is continuously changing thus the skyline results are also changing frequently. The objective of DSQ is to return the skyline in the new data space, which is the data space when the q becomes the origin point. Intuitively, in DSQ, we want to find a set of skyline points which are interesting according to q. The problem of DSQ has been recently investigated in several papers [34, 43, 8]. Another important variation of skyline query which is called Reverse Skyline Query, proposed in [13], offers very important concept to select skyline query based on business owner's perspective. In the worst case, skyline query returns too many answer which makes user may not be able to choose and make decision through so many objects. Hence, some papers

like in [39, 24, 28] proposed methods to return only representative objects of skylines. Representative skylines reduces skyline result's number and return most representative objects to the user. Instead of representative skyline, to help user choosing skyline objects, some papers work to rank skyline result based on quality of skyline objects using different techniques, such as $tf - idf$ weighting scheme [40] or subspace dominance relationship [41]. Another way to find more important and meaningful skyline points in high dimensional space is proposed by Chan et al. [7]. They introduced k-dominant skyline query which retrieves points that are not dominated by another point in at least one of k-dimension. K-dominant skyline relaxes the concept of dominance in skyline query, so that it can reduce the result of skyline query computation. Conventional skyline in [6] only considers categorical attribute either with total or partial order. In fact, in real life, the categorical attributes usually does not come with predefined order, but it depends on each customer's preferences. Wong et al. in [44] proposed an efficient skyline algorithm to handle different preferences on nominal attributes. Moreover, in order to handle various user preferences, study on user preferences using keywords or textual description had been done in [9]. Along with the increasing of the dataset's size that costs both IO and CPU time, some researches apply distributed/parallel settings on skyline query such as in [1] and [29].

2.2 Skyline on Selecting Spatial Objects

2.2.1 Spatial Skyline Query (SSQ)

Spatial skyline query was first introduced by Sharifzadeh et al. [35]. Given a set of data points P and a set of query points Q, SSQ retrieves those points of P which are not

dominated by any other point in P considering their derived spatial attributes, which is the points distance to query points. The difference between spatial skyline query with the regular skyline query is that the domination condition of P depends on the distance to query points Q.

Assume members of a team want to have an important meeting in a restaurant. Their offices are located at different fixed locations in a map. There are so many restaurants in a map, but to reduce the choices, they need to find restaurants that are interesting based on its distances from each of the team members' location. A restaurant p is interesting if there is no other restaurant that has smaller distance to all members' location than p.

Let the set of points as member locations P in the d-dimensional space R_d, and $D(.,.)$ be a distance metric defined in R_d. Given a set of d-dimensional query points $Q = \{q_1, \cdots, q_n\}$ as restaurant locations on a map.

Definition 2.2.1 *(Dominance for SSQ). p spatially dominates p' with respect to Q iff $D(p, q_i) \leq D(p', q_i)$ for all $q_i \in Q$ and $D(p, q_j) < D(p', q_j)$ for at least one $q_j \in Q$. Spatial skyline is a set of objects in P, each of which is not spatially dominated by another object.*

Specifically, p spatially dominates p' if every q_i is closer to p rather than to p'. To proof some query points are closer to p rather than to p', [35] draws perpendicular bisector line of the line segment pp'. With Euclidean distance metric, the point p spatially dominates p' if the query points are in the area of bisector p and not in the area of p'. Bisector concept is also used in the Voronoi diagram to divide the region and claim that all the objects in the bisector area of p will be closer to p than to any other point in area space. Spatial skyline is a set of points that are not spatially dominated by another point.

Naive way to answer spatial skyline query is to compute the distance from all p to

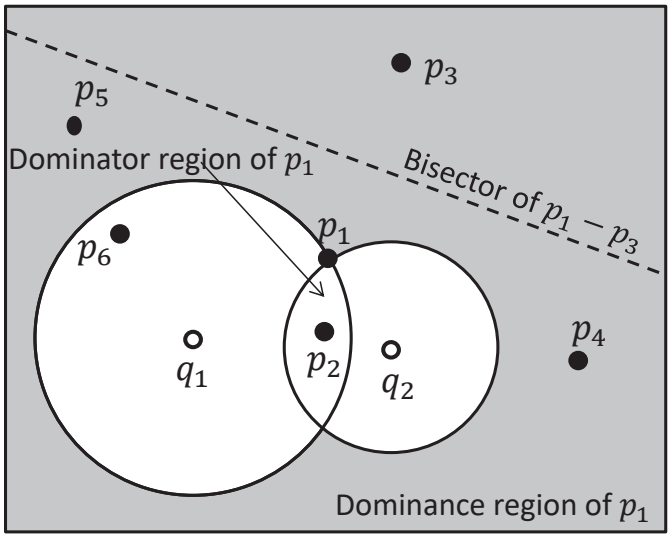


Figure 2.2: Bisector, dominator, and dominance region

all q, then compare each p using spatial dominance rule to get spatial skyline points. To reduce search space, Sharifzadeh and Shahabi [35] define concepts about dominator and dominance region and applied geometric structures like Convex Hull (CH), Delaunay graph (DG), and Voronoi Diagram (V) [11] to present some important geometric properties of spatial skyline points and proposed Branch-and-Bound Spatial Skyline (B^2S^2) algorithm and Voronoi-based Spatial Skyline (VS^2) algorithm.

Figure 2.2 shows an example of dominator and dominance region of a point p_1. Using the perpendicular bisector of $p_1 - p_3$, we know that p_1 dominates p_3 since q_1 and q_2 are in the same side with p_1. If we draw a circle centered at the query point q_1, $C(q_1, p_1)$ and q_2, $C(q_2, p_1)$ with radius $D(q_1, p_1)$ and $D(q_2, p_1)$, then every point placed outside those two circles is guaranteed to have larger distance to q_1 and q_2 than p_1, while every point placed in the intersection of two circles is guaranteed to have smaller distance to q_1 and q_2 than p_1. Region outside the two circles is defined as dominance region of p_1 and intersection region of two circles is defined as dominator region of p_1.

(B^2S^2) is an enhanced modification of BBS algorithm [31]. All the data points in P are indexed by an R-tree. Let us consider that mindist(p, Q) is the sum of distances between the data point p and the points in the set Q and mindist(e, Q) is the sum of minimum distances between the node e in R-tree and the points of Q. First, (B^2S^2) computes the convex hull of $CH(Q)$. For each node in root, (B^2S^2) computes its mindist to the $CH(Q)$ and maintains a minheap H. The node with smallest mindist to $CH(Q)$ is then expanded, until (B^2S^2) finds the first leaf node, let say p. This first leaf node then becomes the first spatial skyline point. (B^2S^2) then generates circle centered in query points on $CH(Q)$ with radius of its distance to p and determines dominator and dominance region of p. Note that (B^2S^2) does not generate this circle for query point q_{ins} that is inside $CH(Q)$. This happens because this circle is completely inside the union of the circles made of query points on $CH(Q)$, so it does not change the dominator region of p.

To simplify the calculation, (B^2S^2) builds a minimum bounding rectangle B for all union of those circles. For each spatial skyline point found, (B^2S^2) maintains the rectangle B. Each entry e is checked based on its position to B:

1. If entry e does not intersect with B, then e should be discarded.

2. If entry points p_{in} placed inside $CH(Q)$, then p_{in} is a skyline point. This is based on the fact that circles defining the dominator region of point p_{in} intersect only at p_{in} if p_{in} is inside $CH(Q)$.

3. If e does not pass above tests, then e need to be compared against the entire spatial skyline points.

(VS^2) is similar with (B^2S^2) unless instead of using R-tree, it uses Voronoi diagram

and adjacency list from Delaunay graph of P. (VS^2) uses a property that each point which its Voronoi cell intersects with the boundaries of $CH(Q)$ is a skyline point. This is obvious because when a Voronoi cell intersects with $CH(Q)$, it means at least one of q in $CH(Q)$ is closer to the Voronoi point. Similar with (B^2S^2), (VS^2) first generates $CH(Q)$, then the closest point to one of query points, let say p, is visited. Using p, (VS^2) builds corresponding circle and generates B which includes all candidate skyline points. Next, (VS^2) visited Voronoi neighbors of p. Using the similar position checking rule with (B^2S^2), (VS^2) decides which point is skyline point, which point is discarded, or compared against the entire skyline points. The experimental result showed that VS^2 outperforms B^2S^2.

2.2.2 The Farthest Spatial Skyline Query (FSSQ)

Different from spatial skyline work, the problem of the farthest spatial skyline queries is proposed in [46]. Given data points P and query points Q in two dimensional space, the farthest spatial skyline query retrieves the data points which are farther from at least one query point than from all the other data points. Let the set P contains points in the d-dimensional space R_d, and $D(.,.)$ be a distance metric defined in R_d. Given a set of d-dimensional query points $Q = \{q_1, \cdots, q_n\}$ and the two points p and p' in R_d.

Definition 2.2.2 *(Dominance for FSSQ). p spatially dominates p' with respect to Q iff $D(p, q_i) \geq D(p', q_i)$ for all $q_i \in Q$ and $D(p, q_j) > D(p', q_j)$ for some $q_j \in Q$. The farthest spatial skyline is a set of objects in P, each of which is not spatially dominated by another object.*

Opposite from the spatial skyline query, this method is helpful to identify spatial locations which are far from undesirable locations. Therefore, this problem is important

in perspective of business location. Consider a company is looking for a desirable location to build a new restaurant. The company need to select an optimal location, which is far from the undesirable locations such as like garbage dump and pollution source, among many potential locations. This problem can be solved in naive way using skyline query by computing the distance of all potential locations to garbage dump and pollution source. Using dominance for FSSQ, we can compute the farthest spatial skyline points for the new restaurant. Since this method requires a lot of computation, to reduce the computation, You et al. in [46] use some geometric properties and proposed *Branch-and-Bound Farthest Spatial Skyline* (BBFS). BBFS uses top-down branch-and-bound search on R-tree to access nodes in decreasing order of distance from query points.

Let $\ell_{\perp}(p, p')$ be a bisector line between p and p',so it separates a space into two half spaces, $h(p, p')$ and $h(p', p)$. $h(p, p')$ is the half space in which every point is closer to p than p', and $h(p', p)$ is otherwise.

There are some important properties in BBFS:

1. For two points p and p', if $CH(Q)$ is inside $h(p', p)$, then p dominates p'.

2. For two points p and p', if $\ell_{\perp}(p, p')$ intersects $CH(Q)$, then p is incomparable with p'.

3. A point p dominates every point in a nodes e, if four corner points of minimum bounding rectangle e are dominated by p.

Besides R-tree structure, P also maintains Voronoi diagram of $VD(P)$. BBFS maintains SumDist(e, Q) in a max-heap H of each entry e, while SumDist(e, Q) is the sum of maxdist(q, e) for all $q \in Q$, and maxdist(q, e) is the maximum distance of all p to q in e. To reduce dominance test, BBFS uses two different lists to maintain skyline points, FS_N

and FS_I. Based on property in BFFS, skyline points in FS_I are incomparable, so BBFS only performs dominance test in FS_N. The node with largest distance in the root then expanded. When it reaches the first leaf node, let say p, then p becomes the first farthest spatial skyline point, then it checks whether its Voronoi cell is disjoint, contained in, or overlapped with $CH(Q)$:

1. If a $V(p)$ is disjoint with $CH(Q)$, then p is inserted into FS_N.

2. If a $V(p)$ is contained inside $CH(Q)$, then p is inserted into FS_I.

3. If none of the above conditions, then p must be checked for the possible intersection and inserted into FS_I if it intersects $CH(Q)$ at least in one edge, otherwise p is inserted into FS_N.

For each iteration, every leaf node is compared to FS_N, if it is not dominated by any other point in FS_N then it is checked whether it will inserted be into FS_N or FS_I. Figure 2.3 shows an example of the farthest spatial skyline in [46].

First, max-heap H is filled with nodes of the root node, N1, N2, and N3. In this example, N3 has the maximum SumDist, so N3 is popped and expanded. The first leaf node that comes up is p_9, and since FS_N is empty and $V(p_9)$ is disjoint with $CH(Q)$, p_9 becomes the first skyline point in FS_N. The second leaf node popped is p_1. Using BBFS property, $\ell_\perp(p_1, p_9)$ is intersect with $CH(Q)$ and $V(p_1)$ is disjoint with $CH(Q)$, so p_1 becomes skyline in FS_N. Using similar way, p_2 and p_8 are also become skyline points in FS_N. On the other hand, since $V(p_{17})$ intersects with $CH(Q)$, p_{17} inserted into FS_I so it cannot dominate any other points. On the next iteration, N8 is dominated by p_8 and N6 is dominated by p_1. At final iteration, there are 14 skyline points, contained in FS_N ($\{p_9, p_1, p_2, p_8, p_{12}, p_{19}, p_{20}, p_3, p_4\}$) and in FS_I ($\{p_{17}, p_5, p_{18}, p_{13}, p_{15}\}$).

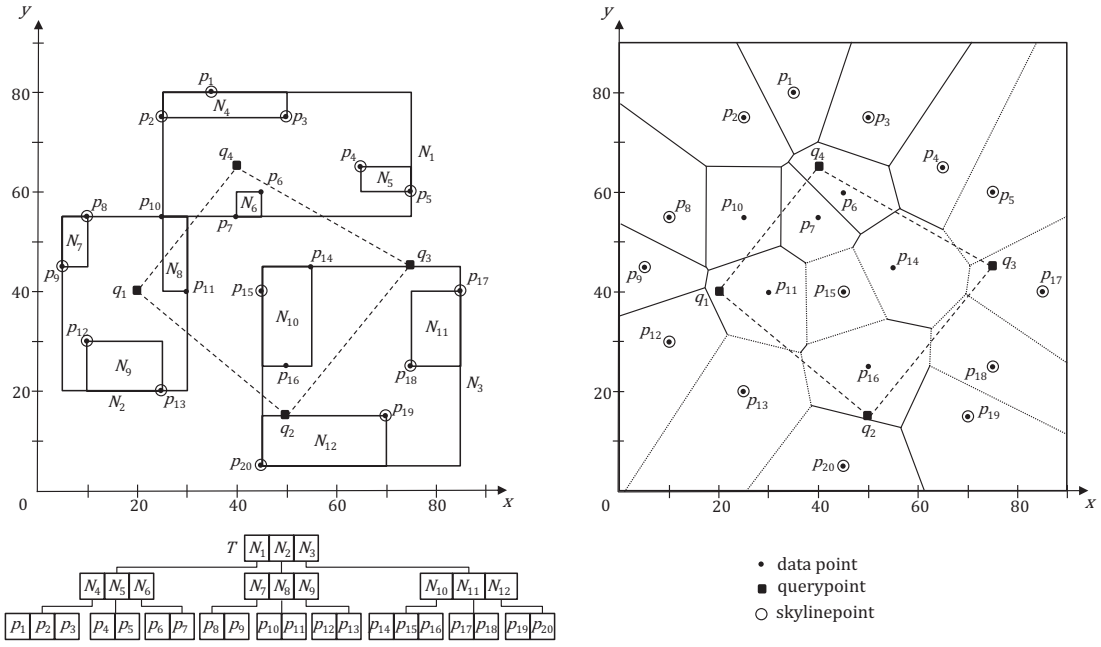


Figure 2.3: The farthest skyline query example [46]

2.2.3 General Spatial Skyline (GSSky) Query

In real life situation, there are many types of facility that have to be considered in selecting good locations. Not only considering one type of facility, in [22], Lin et al. introduced the problem of spatial skyline query with different types of facility, called General Spatial Skyline (GSSky) query. Assume there is a set of hotels, a set of train stations and a set of restaurants in the map, and a user wants to stay at hotel which is close to train station and a restaurant. GSSky query tries to find skyline objects, hotel in this case, that has smaller distance from every type of facilities (train station and restaurant). Given a set of objects $P = \{p_1, ...p_n\}$ and a set of facility types $F = \{F1, ...Fm\}$, which can be categorized into m types. Each facility type has several members, for

example, $F1$ has three objects $F1 = \{f1^1, f1^2, f1^3\}$.

Definition 2.2.3 *(Dominance for GSSkyQ). p spatially dominates p′ with respect to Q iff* $D(p, F_i) \leq D(p', F_i)$ *for all* $F_i \in F$ *and* $D(p, F_j) < D(p', F_j)$ *for some* $F_j \in F$. *The general spatial skyline is a set of objects in P, each of which is not spatially dominated by another object.*

The naive way to find GSSky is first calculates distance from each object to each facility type, gets the smallest distance from each type for each object, and then calculates the skyline. This naive method is not efficient since we have to calculate distance for each object against all facility members. Lin et al. [22] proposed progressive GSSKy algorithm which computes nearest neighbor distance values of the objects with two ways search method, object oriented search and facility oriented search. In object oriented search, for each object p, they applied nearest neighbor query where p is the query point against all facility types F. While on the contrary, in facility oriented search they applied incremental nearest neighbor(INN) algorithm against facilities simultaneously where the query point is a facility point. Objects and each facility types are organized using separate R-Tree. A local priority queue is employed for each facility member for computing incremental nearest neighbor query, to retrieve the next closest object. A global priority queue maintains the current closest object in each facility type. For each iteration, object which is hit at least by one type of facility becomes a candidate skyline, object that is fully hit first becomes skyline, and object that has not been hit after there is one object had fully hit is pruned. Figure 2.4 shows the example of progressive GSSky algorithm.

In Figure 2.4, apartments(a), train stations(b), and restaurants(s) are organized by each R-tree, let say R_a, R_b, R_s. In the first iteration of facility oriented search, a_3 is hit

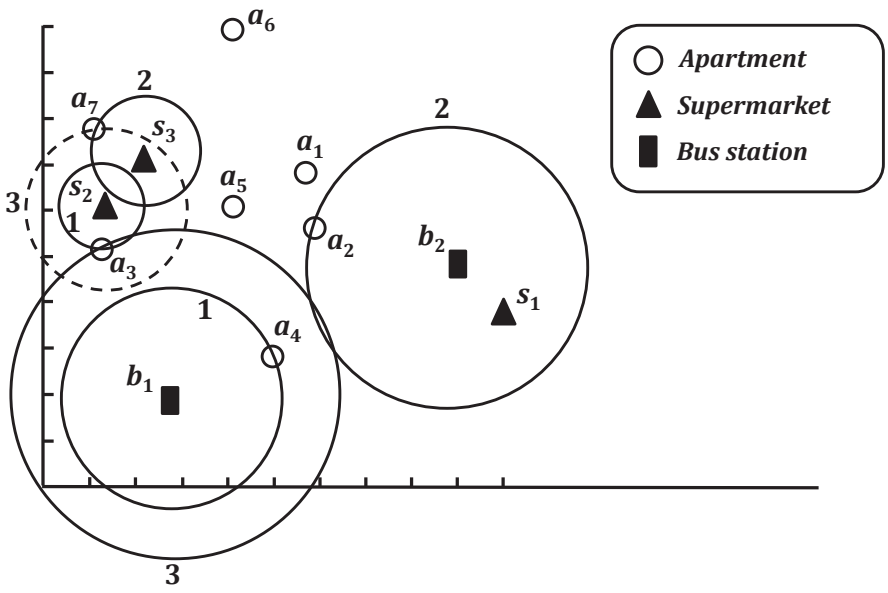


Figure 2.4: The general skyline query example [22]

by s_2 and a_4 is hit by b_1. In the second iteration, a_7 is hit by s_3 and a_2 is hit by b_2. In the third iteration, a_7 is hit by s_3 and a_3 is hit by b_1. We say that a_7 has a redundant hit, since it has already been hit by s_2, the closest restaurant from a_7. For a_3, it has been fully hit by both type, so a_3 become the first skyline and we can pruned the rest of objects that have no hit so far ($\{a_6, a_5, a_1\}$). The next step is running the object oriented search for candidate skyline objects, which are objects that have been hit by at least one type ($\{a_4, a_7, a_2\}$).

2.2.4 Skyline based on Nearest and Farthest Neighbour (SkyNFN) Query

Combining the farthest and nearest problem, [25] introduced Skyline based on Nearest and Farthest Neighbor (SkyNFN) Query. Suppose a family wants to rent a house. According to the family, some favorable facilities like restaurant, supermarket, school, and

library must be close to the house, while some unfavorable facilities like garbage dump, criminal area, and noise sources must be far. Intuitively, a house is a better choice if its farthest distance to desirable facility is smaller, and its nearest distance to undesirable facility is larger. Given a set of objects $P = \{p_1, ...p_n\}$ and a two set of facility objects $F = \{F_d, F_u\}$, while F_d represents set of desirable objects and F_u represents set of undesirable objects. pfn_{p_i} is the farthest neighbor of object p_i to desirable facility and pnn_{p_i} is the nearest neighbor of object p_i to undesirable facility.

Definition 2.2.4 *(Dominance for SkyNFN). p spatially dominates p′ with respect to F iff* $pfn_p < pfn_{p'}$ *and* $pfn_p \geq pfn_{p'}$ *or* $pnn_p \leq pnn_{p'}$ *and* $pnn_p > pnn_{p'}$*. The SkyNFN is a set of objects in P, each of which is not spatially dominated by another object.*

Finding the nearest distance from undesirable facility is to guarantee that there is no unfavorable facility within this range. On the other hand, finding the farthest distance from desirable facility is to be sure that all favorable facilities are inside this range. In [25], Lin et al. proposed EFFN algorithm to solve SkyNFN Query efficiently. EFFN used quad-tree [14] to find nearest neighbor of an object to undesirable facilities. The nearest neighbor of query object q might fall in the same block with q in quadtree, or in adjacent block. First we compute the distance between q and the undesirable facilities in the same block, let say p, then we compare its distance to the distance between q to the undesirable facilities in the adjacent leaf blocks to see if there exists any leaf blocks adjacent to query block which contains closer undesirable facility to query object. Figure 2.5 shows that the closest undesirable facility from q is d_8 since it is in the same block with q, and no other point in the adjacent blocks that has closer distance than d_8.

To find the farthest neighbor, EFFN divides the two-dimensional space into four quadrants. In each quadrant, EFFN runs SFS algorithm with different dominance rules

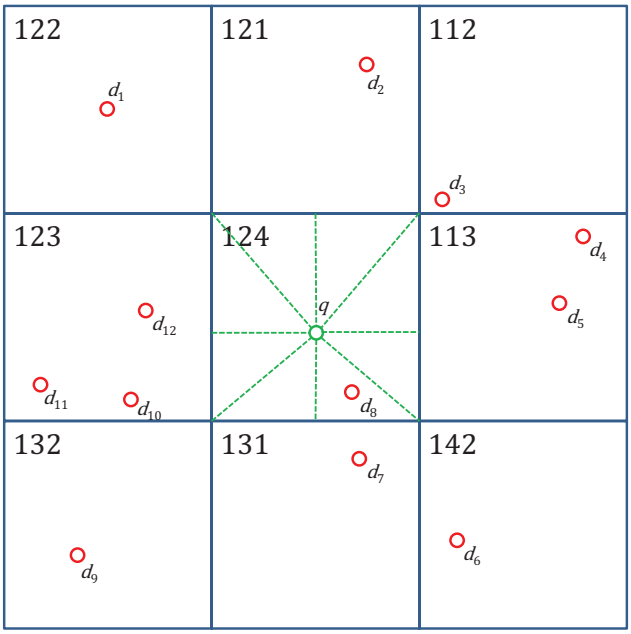


Figure 2.5: The nearest neighbor query using quadtree example [25]

to find skyline point for each quadrant. In quadrant I, the larger values are preferred in either axis X and Y. In quadrant II, the smaller values in axis X and the larger value in axis Y are preferred. Quadrant III prefers the smaller values in both axis, and quadrant IV prefers the larger values in axis X and the smaller value in axis Y. The farthest neighbor for any query point must be one of the skyline point in those four quadrants. First, EFFN checks in which quadrant q is located and computes q's distance to the skyline point in the opposite quadrant, since opposite quadrant might have higher chance to have the farthest neighbor. Next, similar to finding nearest neighbor, EFFN calculates distance from q to each of skyline quadrants. Figure 2.6 illustrates the fourth quadrant of desirable facilities' space. EFFN computes the skyline points for each quadrant. In this example, q is in quadrant four, and the opposite of quadrant four is quadrant two, then we set initial farthest distance as the distance between q and f_2. After initialization, we compare initial distance with distance between q to other skyline points in other

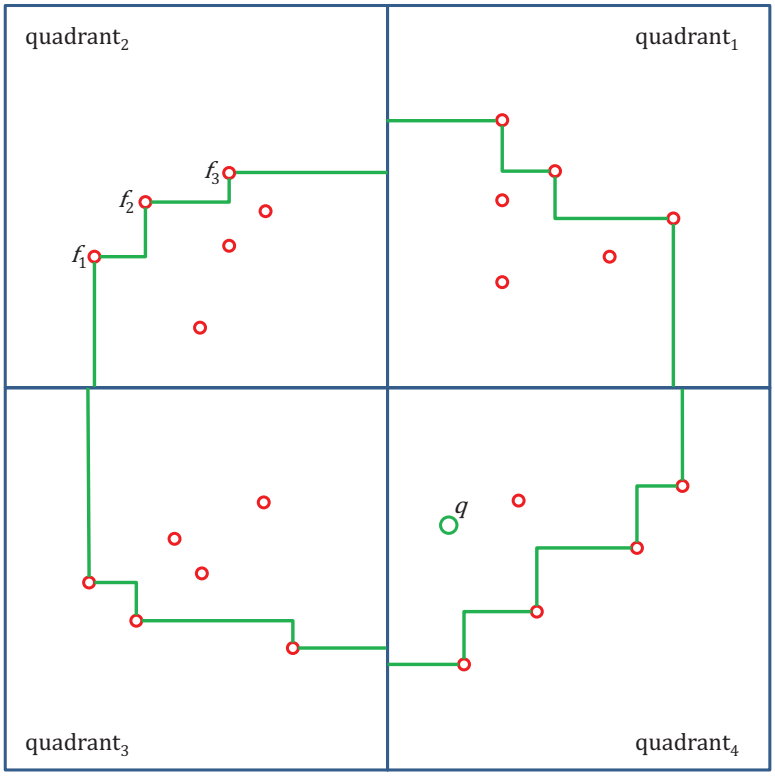


Figure 2.6: The farthest neighbor query using quadrant example [25]

quadrants to find the farthest desirable facilities.

2.2.5 Spatial Skyline with Preferences Query

In spatial skyline, distance is the first parameter that needs to be considered. In addition to that, sometimes users also consider other preferences beside distance. For example, in selecting a restaurant, a user might includes price, rating, or type of the restaurant into his/her consideration. In [19], Kodama et al. consider distance and categorical attribute of spatial point.

Assume some restaurants have locations in X and Y coordinates, a type of cuisine served, like Chinese, French, Japanese, or Italian food, and a type of price, like low, medium, or high price. Before choosing a restaurant, user describes his/her preferences

in each categorical attribute in his/her profile, for example he/she prefers higher price and prefer Italian food the most, then Japanese and French, and the last is Chinese food. Formally, we can order the categorical preference of user profile into this order: Italian $\prec$ Japanese $\simeq$ French $\prec$ Chinese and high $\prec$ medium $\prec$ low. Using this order we can say that Italian food is better than Japanese and French food, while Japanese and French food are incomparable, and three of them are better than Chinese food. Given a query point q, a set of objects $P = \{p_1, ...p_n\}$, each object p has an ID, a spatial attributes $p.l$, and a value for each attribute $a \in A$, where A is the set of non-spatial attributes. Let $p.l$ be the distance from q to object p and $p.a_i$ be the value of attribute a_i from object p. We say object p dominates p', if p is equal to or better than p' in spatial and non-spatial attributes, and if p is better than p' at least in one attribute. Figure 2.7 shows the example of spatial skyline with preferences query proposed in [19].

Using the same dominance rule as Definition 2.1.1 w.r.t user profile and BBS algorithm, we can retrieve skyline object of query q. Based on user profile above, we can discard a and f since they are far from user and user do not like them in categorical attributes. c and e are dominated by b, and i, j, h are also not good in categorical attributes. Finally we can find the skyline restaurants in Figure 2.7 are b, d, and g.

Different with Kodama et al., Arefin et al. [5] utilized surrounding facilities for calculating the importance of locations and demonstrated that surrounding environment is as important as other attributes for selecting spatial objects. Assume a user wants to rent an apartment within 1000 meters of a well known place q. The user specifies that the apartment must have nearby restaurants and supermarket within 500 meters. We called the apartment as target facility, and the restaurants and the supermarkets as surrounding facilities. The number of restaurants and supermarkets within 500 meters from the

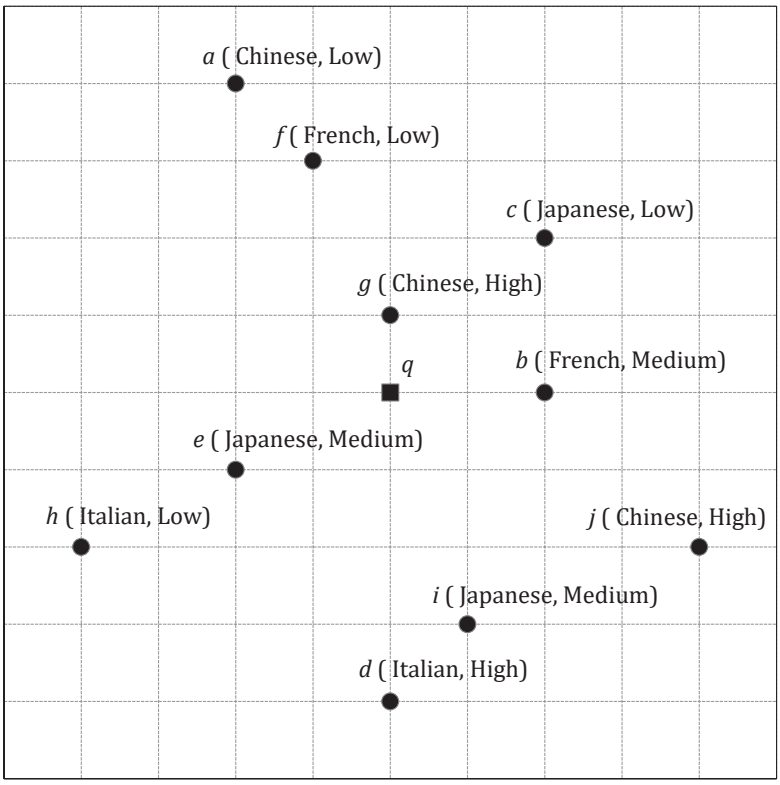


Figure 2.7: The spatial skyline with preferences query example [19]

apartment also taken into account by the user, and the larger number is better in this case. Moreover the user also considers about price and rating of the restaurants and supermarkets. The algorithm proposed in [5] has several steps. First, it indexes the target facility and each surrounding facility into different aggregation R-trees (aR-tree) [21] to maintain index of both spatial and non-spatial information of each facility. The aR-tree has similar structure with R-tree, but each node entry in aR-tree is an aggregate values. In the leaf node, aR-tree collects objects and their corresponding attributes values, while internal node collects the minimum value in each attribute of its descendant objects and total number of descendant objects. Second, user specifies a query point q and its distance D_q to select target facility, and distance to select surrounding facility from target facility D_s. According to D_q, the algorithm selects the target facilities in the range of

D_q. Third, user specifies preferable value of attributes in surrounding facilities which influences the quality of the selected objects in the range D_s from the selected target facilities. Fourth, it counts the number of such objects for each selected target facilities and combines the count of surrounding facilities and the non-spatial information into a table. Finally, it performs skyline queries to select spatial objects from the combined table.

In addition, Arefin et al. in [4] also proposed a spatial skyline query for group of users located at different positions. Their method can select a convenient place for all users of a group if the group wants to hold a meeting in a restaurant, for example. Their method not only considers the distance from each user in group and each restaurant, but also take into account the non-spatial properties of the restaurants. In order to efficiently search non-dominated objects, [4] uses VoR-tree [36], a variation of R-tree that is also using the concept of Voronoi diagram to maintain Voronoi neighbor of each object. Objects are indexed with R-tree, then each leaf node in R-tree has a pointer to disk block that stores the information of Voronoi neighbor. Figure 2.8 shows the example of Vor-tree applied in [4].

For selecting non-dominated restaurants, first, we calculate the centroid of user locations, and select the closest restaurant from the centroid, in this example r_7 is the closest restaurant from the centroid. Using the same method with spatial skyline query, we draw circles whose radius are the distance from each user's location to r_7. The union region of all the circles now becomes the search region of r_7. Next, r_7 becomes the first skyline, and for all adjacent Voronoi points with r_7 that has Voronoi cell completely inside or intersect with search region of r_7 ($\{r_5, r_2, r_9\}$) are also inserted into the heap H. Next, since r_5 has the smallest sum of distance to all q, and r_5 is not dominated by r_7, r_5

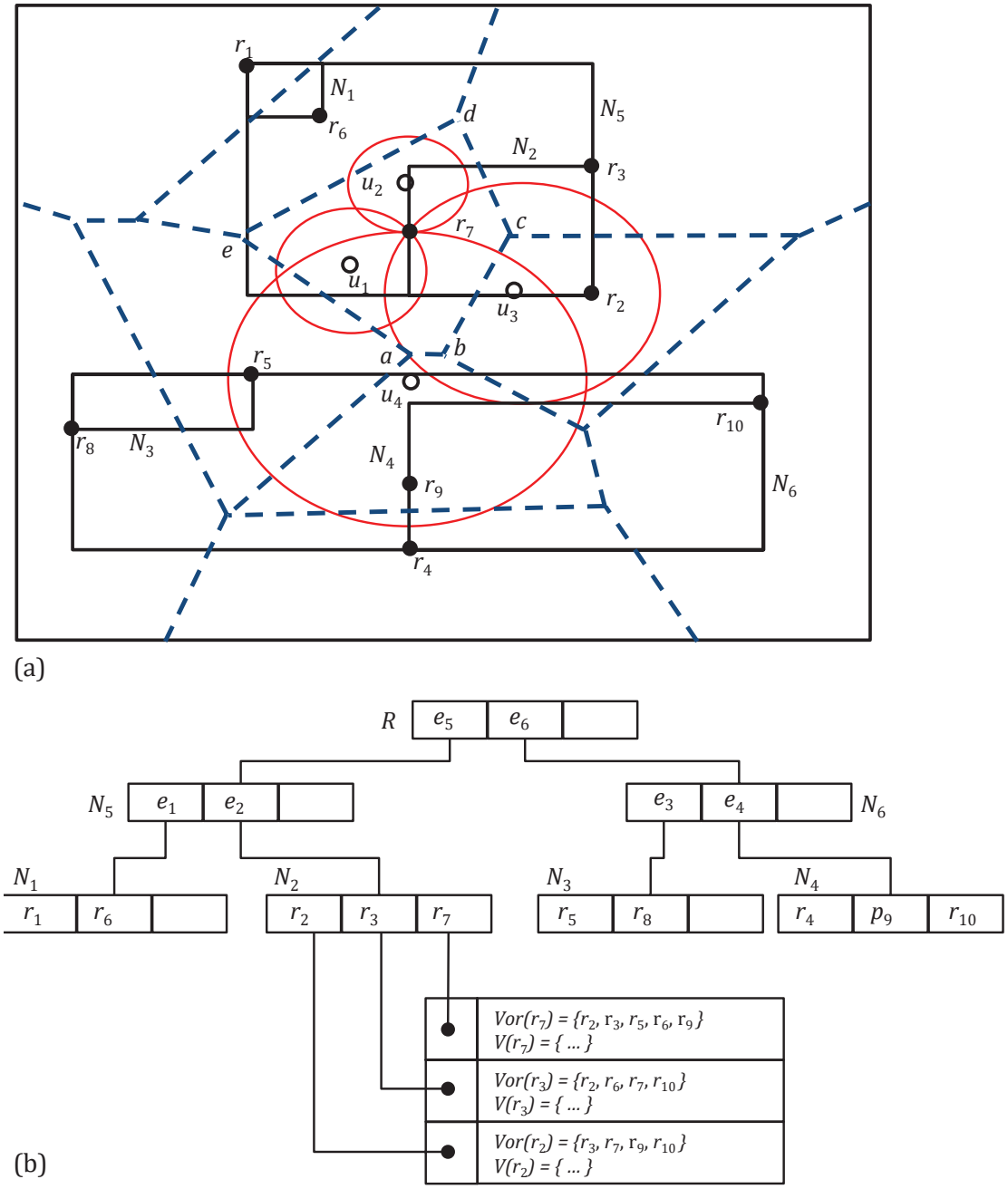


Figure 2.8: The Vor-tree example in [4]

becomes the next skyline point. Then, r_5 is popped from heap H and Voronoi neighbors of r_5 that have Voronoi cell inside of or intersecting with search region of r_7 and have not been checked, are inserted into the heap. Similarly, the process continues until the

heap is empty.

2.2.6 Other Related Works in Skyline on Selecting Spatial Objects

Besides all of the researches explained above, there are many more researches which discussed about spatial skyline in location selection. Based on the fact that spatial objects not only have spatial locations but also have quality attributes like rating and price, Lu & Yiu [27] proposed Farthest Dominated Location (FDL) which takes into account both quality attributes and spatial locations. FDL selects an object which have the farthest distance from other object which dominates it in quality attributes. Conventional spatial skyline only considers about distance but does not take into account about user direction. Guo et al. [16] stated that direction is also should be considered to select best location, especially for mobile user. They proposed Direction-based Spatial skylines (DSS), which retrieves nearest objects around the user from different directions. In mobile environment, [17] studied on the problem of continuous skyline query, where the query point is a moving object, so that the skyline points changes continuously according to the query movement. Moreover, [20] proposed an approach for computing skylines on routes (path) in a road network, which also considers various preferences, not only distance attributes. In location based application, point locations are also augmented with textual descriptions. To find places that are near to some spatial query points and textually relevant with a set of keywords given by the user, Shi et al. [37] proposed a method to answer Spatio-textual skyline (STS) query problem. Due to the limited precision of mobile device and privacy issue, the input of a user location in location based application is often in the form of spatial range. Lin et al. [23] proposed some methods to answer Range-based Skyline Query (RSQ) problem.

All of the above studies are based on the assumption that there are candidate points to choose skyline location and focused only on spatial data points. In this thesis, different from previous researches, we focused on the problem of area selection in which the location of candidate query points q are not given.

2.3 Reverse Skyline Query

To understand reverse skyline query, first we will discuss about Dynamic Skyline Query (DSQ). In the traditional skyline, the origin point is always (0,0) which we refer as static skyline, because for the given dataset the skyline points will remain the same. In some cases, when we consider different query point q as the origin, we will obtain different skyline points according given query point, which we refer as dynamic skyline points of q. The dynamic skyline query retrieves all objects that are not dynamically dominated by other objects with respect to their distances to a given query point. With respect to a given query point means that dynamic skyline query will transform every object into new n-dimensional space with the query point as the new origin point.

Definition 2.3.1 *(Dynamic Skyline Query). Given a query point q and a d-dimensional data set P, a DSQ according to q retrieves all data points in P that are not dynamically dominated. A point* $p \in P$ *dynamically dominates* $p' \in P$ *w.r.t q if* $|q_i - p_i| \leq |q_i - p'_i|$*, for all* $i \in d$*, and* $|q_j - p_j| < |q_j - p'_j|$ *for at least one* $j \in d$*.*

Let us consider a list of hotel in Figure 1.1(a). For example, we want to compute dynamic skylines of h_5, $(x = distance; y = price) = (15, 40)$. To find dynamic skyline based on h_5, we first transform objects as follows. If x value of objects is less than 15, then transform the x value into $15 + (15x)$. Similarly, if y value of objects is less than 40,

then transform y values into $40+(40y)$. These transformation results in transformed data objects as in Figure 2.9(a). In the example, $h_2 = (35, 25)$ is transformed to $h'_2 = (35, 55)$, $h_4 = (10, 20)$ is transformed to $h'_4 = (20, 60)$, and so forth. We, then, compute skyline query for the transformed data objects as dynamic skyline query for $h_5 = (15, 40)$, which retrieves h_4, h_6, q, h_7. Different from skyline query, reverse skyline query [13] answers skyline query from company's/business owner's perspective.

Definition 2.3.2 *(Reverse Skyline Query). Given a d-dimensional dataset P and a query point $q \in P$, we call an object $p \in P$ a "reverse skyline" of q if q is a dynamic skyline of p. In other words, p is in the reverse skyline of q iff $\nexists p' \in P$ such that:$|p'_i - p_i| \leq |q_i - p_i|$, for all $i \in d$, and $|p'_j - p_j| < |q_j - p_j|$ for at least one $j \in d$*

Intuitively,"reverse" skyline query of q retrieves a set of points that are as preferable as q. Consider our example dataset in Figure 1.1(a). Suppose a business owner wants to build a new hotel with price 30 and distance 30, respectively. She/he wants to know which customers from the existing hotels that will be interested with his new hotel. The naive way to find all reverse skyline objects for q is by performing dynamic skyline query for each point p in dataset P. All p in P that have q as their dynamic skyline become the reverse skyline set of q. Figure 1.1(b) shows hotels that have q as their dynamic skyline. Intuitively, a user that is interested in h_5 hotel may also be interested in q since q is a dynamic skyline of h_5. The similar intuition holds on h_2 and h_6. Therefore, we can expect users who are interested in h_2, h_5, and h_6 might also be interested in q. Calculating dynamic skyline for each p in P to find reverse skyline of q needs very large computation, since we have to consider each p in P, transform the d-dimension and calculate dynamic skyline for each p. In order to reduce the search space, in [13], Dellis and Seeger introduced Branch and Bound Reverse Skyline (BBRS) algorithm using a

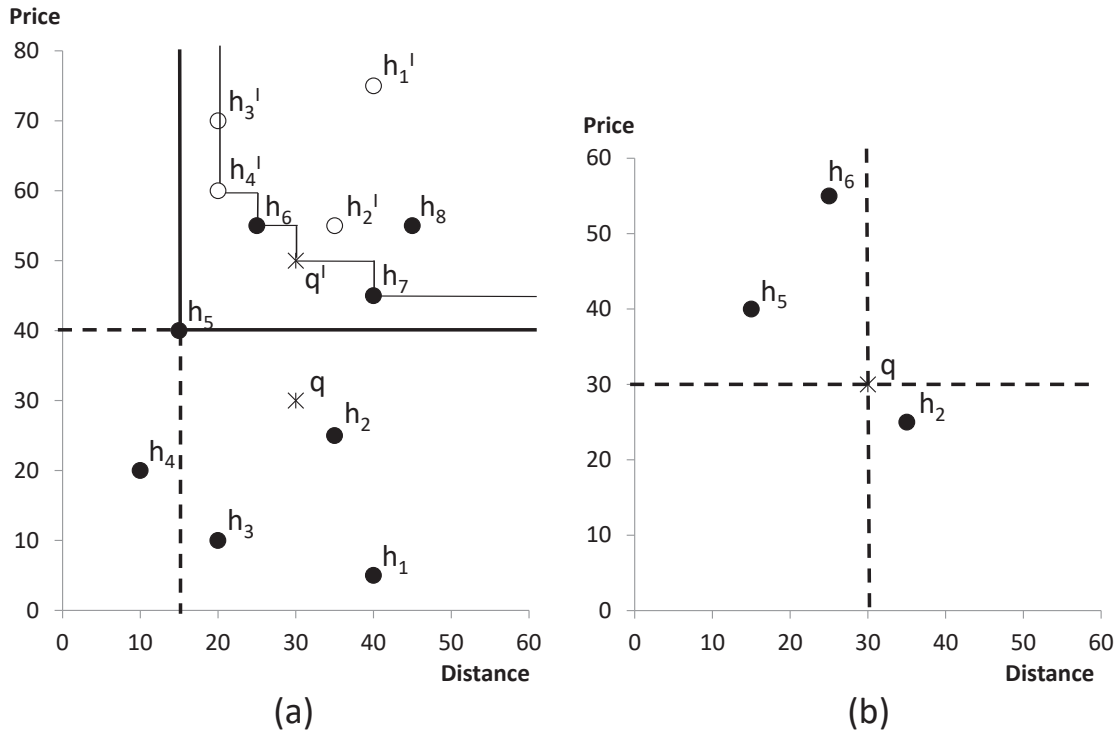


Figure 2.9: Dynamic Skyline (a) and Reverse Skyline (b)

concept called Global Skyline (GSL) and propose two efficient algorithms, Branch and Bound Reverse Skyline (BBRS) and Reverse Skyline using Skyline Approximations (RSSA). Furthermore, to overcome the limitation in RSSA, Gao et al. [15] proposed Full Reuse-based Reverse Skyline (FRRS) and global-skyline-based reverse skyline (GSRS).

Definition 2.3.3 *(Global Skyline (GSL)). Given a d-dimensional data set P and a query point q, p is said globally dominates p′ with respect to q if: (1) (p-q)(p′-q)>0 for all dimensions, and (2) distance p to q are smaller or equal then distance p′ to q for all dimensions, and smaller at least in one dimension.*

Black point ($\{h_2, h_3, h_4, h_5, h_6, h_7\}$) in Figure 2.10 shows the global skyline points of query point q. In [13], Evangelos and Seeger has proved that reverse skyline point is always a member of global skyline point.

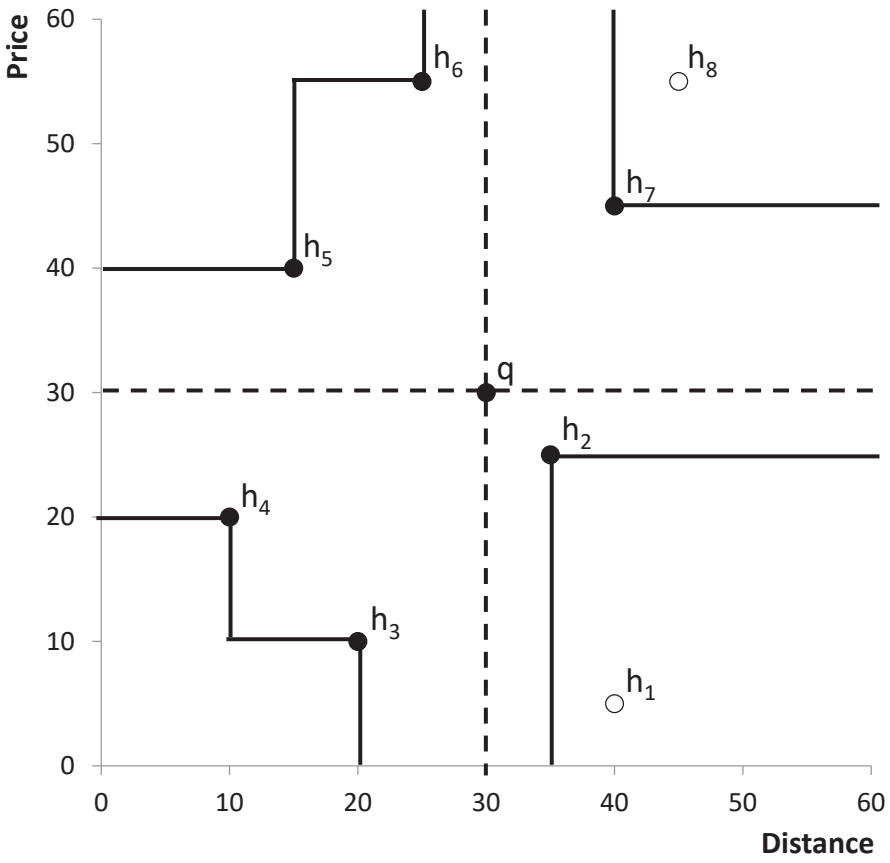
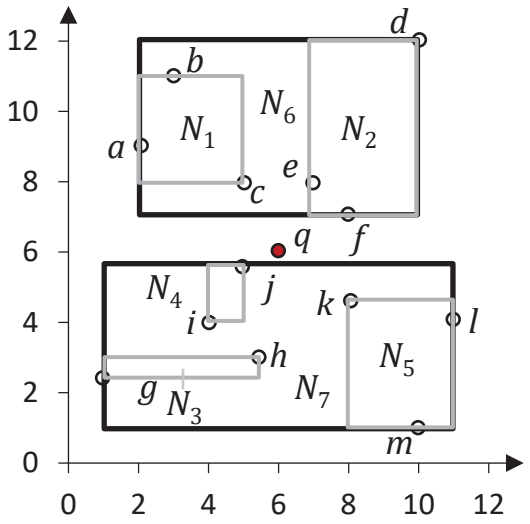


Figure 2.10: Global Skyline Points

2.3.1 BBRS algorithm

BBRS is improvement version of BBS algorithm [13, 15]. BBRS computes the reverse skyline of q by expanding minheap H of R-tree index which maintains the distance of each node entry to q. First, BBRS computes GSL(q) which contains the candidate of reverse skyline points. For each $p \in$ GSL(q), BBRS runs a window query. Window query is the rectangle area with p as its center, and distance to a given query point q and its extention as its border coordinates. If there is another point inside this rectangle, then p is not in a reverse skyline of q, otherwise p is a reverse skyline of q. Figure 2.11 shows an example of MBR of R-tree(a) and heap situation(b) in BBRS.



(a)

Action	Heap Contents	*S*
Access *Root*	<N_7, N_6 >	∅
Expand N_7	<N_6, N_4, N_5, N_3 >	∅
Expand N_6	<N_4, N_2, N_1, N_5, N_3 >	∅
Expand N_4	<**j**, N_2, N_1, N_5, i, N_3 >	{j}
Expand N_2	<~~e, f~~, N_1, N_5, i, N_3, d >	{j}
Expand N_1	<**c**, N_5, i, N_3, a, b, d >	{j, c}
Expand N_5	<**k**, i, N_3, a, l, b, m, d >	{j, c, k}
Expand N_3	<~~h~~, ~~a~~, ~~l~~, ~~b~~, ~~g~~, ~~m~~, ~~d~~ >	{j, c, k}

(b)

Figure 2.11: BBRS example in [15]

Similar with BBS, BBRS will expand node at top of the heap. In this example, j is the first global skyline, and BBRS then builds a window query for j. When performing window query, each time BBRS must traverse R-tree. Since there is no other point inside the window query of j, then j becomes reverse skyline point. Using the same calculation, after some iteration, h is another global skyline, but there is another point in its window query, so h is not a reverse skyline point. Figure 2.12 illustrates window query for j (a) and for h (b). This step continues until heap is empty, and the reverse skyline points of q are ($\{j, c, k\}$).

2.3.2 RSSA algorithm

The main idea of RSSA is to compute the dynamic skyline for each database object in pre-processing step. These pre-computed dynamic skylines are then used to discard points that are not belong to the reverse skyline. Using the dynamic skyline informa-

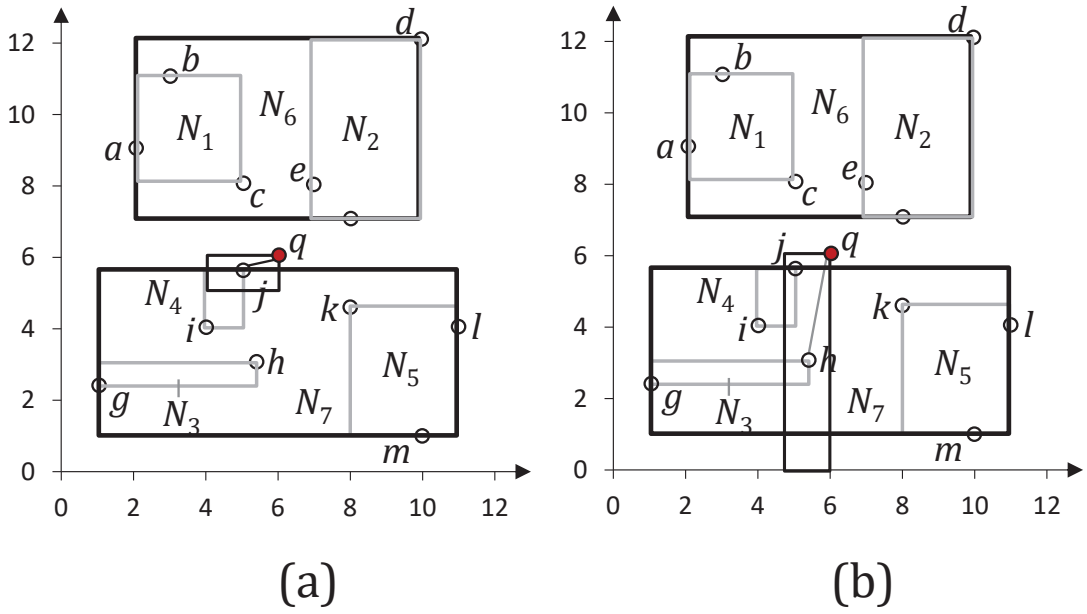


Figure 2.12: Window queries example in [15]

tion of each point p, RSSA generates two region, The Dynamic Dominance Region of p, DDR(p) and The Dynamic Anti-Dominance Region, DADR(p). DDR(p) contains dominated points while DADR(p) contains non-dominated points, so that p which has q inside DDR(p) will be discarded while p which has q inside DADR(p) will be a reverse skyline point. The points that do not fall in DADR(p) or DDR(p) are then examined in the refinement step. Similar to BBRS, in the refinement step, RSSA also uses a window query for each candidate, but since the number of candidates is smaller after the filtering step, the number of window query applied can be significantly reduced. Figure 2.13 shows the example of RSSA algortihm. In this example, since q falls in DADR of j, then j is reverse skyline of q. RSSA has better performance than BBRS. Unfortunately, every time RSSA runs the window query, it has to search R-tree from root repeatedly thus burdening I/O and CPU costs, especially if the number of global skylines that do not fall in DADR(p) and DDR(p) is very large.

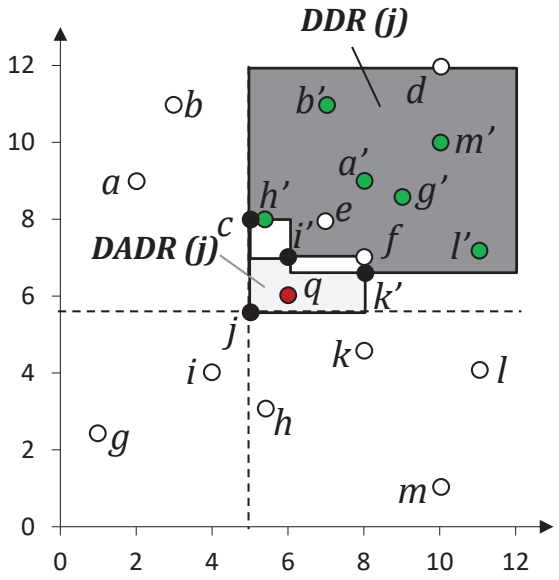


Figure 2.13: DDR and DADR example in [15]

2.3.3 FRRS algorithm

To avoid repeated traversal of the R-tree in RSSA when performing window query for each global skyline point, Gao et al. [15] proposed Full Reuse-based Reverse Skyline (FRRS) algorithm. In order to achieve better computational performance, they use single traversal of the R-tree and efficient pruning technique. They used two heaps to maintain visited nodes in R-tree, which they called as reuse technique. First heap, H_g, maintains the nodes that are not globally dominated by global skyline points; another heap, H_w, maintains the nodes that are globally dominated by global skyline points.

First, FRRS computes the global skyline of query point and insert them to H_g, while the other entries that are globally dominated by retrieved global skyline points are pushed into H_w. Next, using pre-computed dynamic skyline, FRRS checks each global skyline in H_g whether it falls in DADR or DDR, and inserts the point into H_w. Finally, for other global skylines that are not placed in DADR or DDR, FRRS runs window query and checks each window query using H_g and H_w without traversing R-tree.

Although this reuse technique can reduce R-tree traverse time, maintaining H_g and H_w in memory are also not very efficient especially for large data sets.

2.3.4 GSRS algorithm

To overcome the shortcomings in FRRS, Gao et al. [15] proposed global-skyline-based reverse skyline (GSRS) algorithm and introduced global-1-skyline concept. Global 1-skyline query points are set of points that is globally dominated by exactly one point. In Figure 2.10, global 1-skyline points are h_1 and h_8. Given a D-dimensional data set P and a query point q, for a window query corresponding to any point $p \in P$, if it does not contain any global skyline point or global 1-skyline point, the window query must be empty. Gao et al. [15] proved that we do not have to check all points against all windows queries, we simply need to check whether there are any global skyline point or global 1-skyline point inside the query window.

2.3.5 Other Related Works in Reverse Skyline Queries

Because of the importance of its use in various fields of application, research in the reverse skyline has gained much attention in the database research community such as in business planning and preference based marketing [13], and also in profile-based investment [45]. Similar with traditional reverse skyline query, many reverse skyline variants had been proposed. Reverse skyline on data stream is proposed in [47]. In [42], reverse skyline query is also applied in wireless sensor networks for environmental and habitat monitoring. Furthermore in skyband, Liu et al. [26] introduced reverse k-skyband. Dealing with big data, [32] proposed reverse skyline in parallel computation using MapReduce. In [18], Islam et al. showed how to answer why-not questions in reverse skyline

queries. The paper answered why a particular point is not in a reverse skyline, and what should be done to put the point into a reverse skyline.

All of the proposed related works above only considers for zero dimensional data. Specifically, none of them considers about how to select skyline or reverse skyline from two dimensional objects. Therefore, we can not directly use these algorithms to answer skyline and reverse skyline problem in two dimensional objects, such as areas in a map.

Chapter 3

Unfixed-Shape Area Skyline

Let us consider again our example map in Figure 1.2(a). In some real world examples, we cannot assume there are candidate points like p_1,..., p_3, for the selection problem on a map. For example, the businessman wants to build a new supermarket if there is a good vacant area. The businessman may also want to take over a building that locates in a good area at any cost. In this situation, the candidate points are not given and the businessman has to find a good location in an area on the map. In other words, she/he has to find two-dimensional area on the map whose distance from desirable facilities should be closer and distance from undesirable facilities should be farther. We call such area as "Area Skyline". Since an area may be in various shapes, the closest distances and the number of areas are different for each different shape.

As discussed on the previous chapter, skyline query is a well known method for selecting small number of desirable objects. It selects objects that are not dominated by another object. In this chapter, we consider a method for selecting areas that are not dominated by another area. Different from conventional skyline query, selecting areas is more complicated because we are considering unfixed shaped areas and an area's

properties are changed if its shape is changed. To solve this problem, we introduce the concept of area skyline query and propose Unfixed-shape Area Skyline (UASky) algorithm.

3.1 Problem definition

Let A be a rectangular target area, where the businessman wants to build his supermarket, on a map. Let $F = \{F1, ...Fm\}$ be a set of facility types, which can be categorized into m types. Each type is classified into desirable (annotated by + mark) or undesirable (annotated by − mark). Each facility has several facility objects, for example, a desirable facility $F1^+$ has three objects $F1^+ = \{f1_1^+, f1_2^+, f1_3^+\}$. In Figure 1.2(a), desirable facilities are university (F1+ facility (star)) and station (F2+ facility (triangle)), while the undesirable facility is competitor supermarket (F3- facility (square)).

3.1.1 Disjoint Areas

To find area skyline, first of all, we have to divide the area A into some disjoint areas, and we calculate the distance from an area to the closest location for each facility of F^+ and F^- for each disjoint area. In our area selection problem, we have to find the closest facility from a given area frequently. Voronoi diagram [11] is a well known data structure that is useful for computing the closest facility.

Given a set of P points of F, the Voronoi diagram of P is the subdivision of the plane into P disjoint regions. Each Voronoi region contains a point of P, say p, which is called the Voronoi point of the region $V(p)$.

In the Voronoi diagram of P, a point q lies in the region $V(p_i)$ if and only if $dist(q, p_i) \leq$

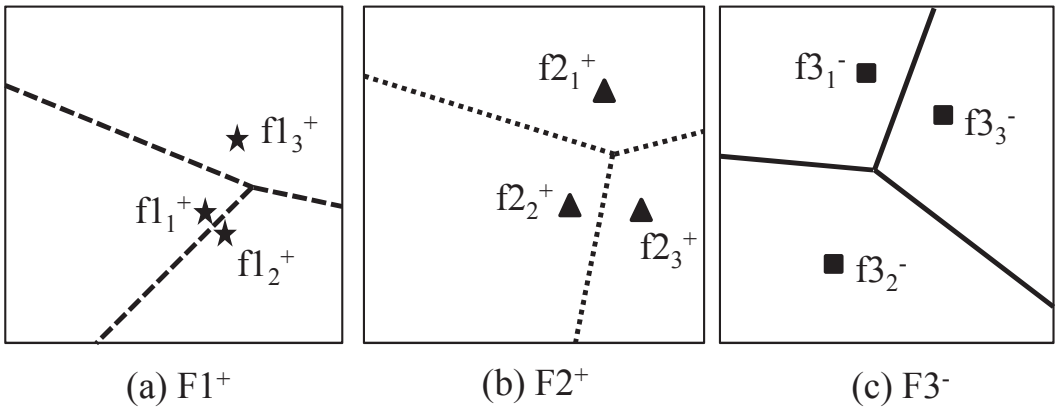


Figure 3.1: Voronoi Diagram for $F1^+$ (a), $F2^+$ (b), and $F3^-$ (c)

$dist(q, p_j)$ for each $p_j \in P$ with $j \neq i$, where $dist(q, p)$ denotes the distance between q and p. Figure 3.1 shows how we apply a Voronoi diagram in target area A based on each facility type, university (a), station(b), and competitor supermarket(c). Using Voronoi diagram, we can find the closest point in P from a given point q efficiently. For example, if a query point lies in the region that contains $f1_1^+$ in Figure 3.1 (a), the closest university (star) from the query point is $f1_1^+$.

Since we are considering distances to all of the three facilities, we divide the area by using three Voronoi diagrams. Figure 3.2 shows 13 disjoint areas $(a_1, a_2, ..., a_{13})$ that are divided by the three Voronoi diagrams of Figure 3.1.

3.1.2 Distance from / to an Area

For each disjoint area, we calculate two distances, which are minimum (min) distance and maximum (max) distance to the closest facility for each F^+ and F^-. Figure 3.3 illustrates the min-max distance to the point (circle) from the triangle area. If the closest facility (circle) is included in an area, the min distance from the area is 0.

For each disjoint areas, we calculate the min-max distance to the closest facility.

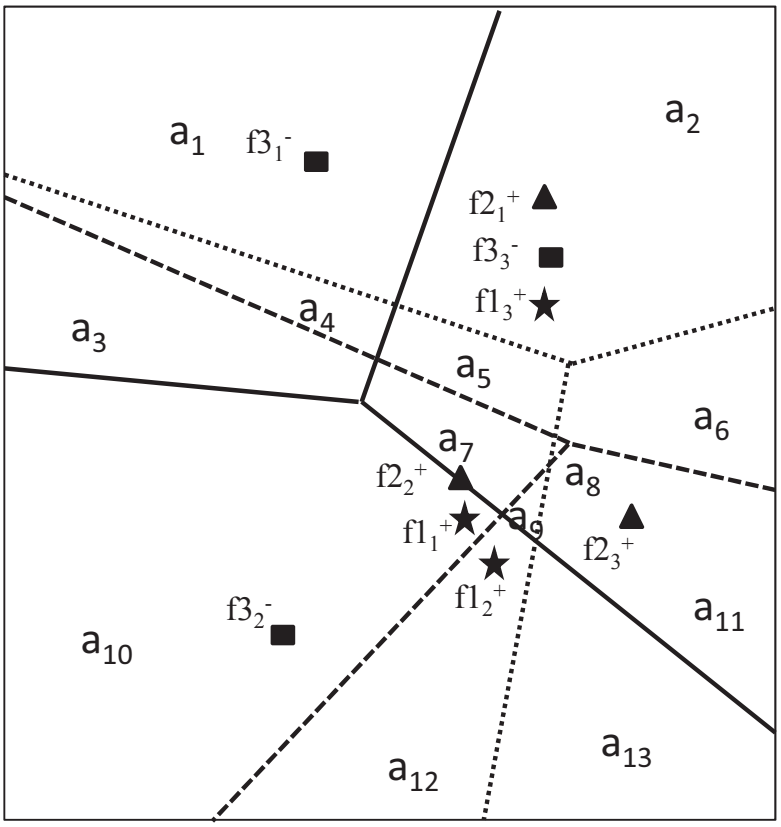
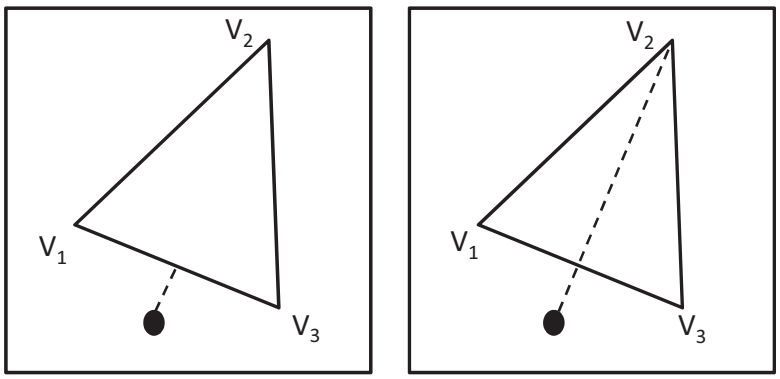


Figure 3.2: Disjoint Areas divided by 3 Voronoi Diagrams

In order to calculate these distances efficiently, we first compute distance from each of vertexes that encloses an area. In Figure 3.3, we calculate distance from v_1, v_2, and v_3, which are vertexes of the triangle area. Note that we can efficiently compute the closest facility from the vertexes by using Voronoi diagram. Also, note that one vertex is included by more than one area, for example, the vertex that lies at the intersection of the three solid segments in Figure 3.2 is included by three areas a_3, a_7, and a_{10}. Then, after computing min distance from vertexes, we calculate min-max distance for each area. Table 3.1 is the Minmax table of the restaurant example.

Figure 3.4 illustrates the corresponding facilities of a_3 that is extracted from Figure 3.2. For example, area a_3 is in the Voronoi region $f1_1^+$, $f2_2^+$, and $f3_1^-$. Thus, the



a) Closest distance b) Farthest distance

Figure 3.3: Min Distance and Max Distance

closest $F1^+$ facility from a_3 is $f1_1^+$. Similarly, $f2_2^+$ and $f3_1^-$ is the closest $F2^+$ and $F3^-$ facility from a_3, respectively.

To calculate the distance from $f3_1^-$ to a_3, we first calculate the distance from $f3_1^-$ to all vertexes that enclose a_3. In Figure 3.4, vertex i, j, k, and l are vertexes that enclose a_3. The min distance from $f3_1^-$ to a_3 is the distance to one vertex or the distance to one segment if the facility lies outside the area.

In this example, the min distance from $f3_1^-$ to a_3 is the distance to the segmentij. As we already have all the distances from $f3_1^-$ to all vertexes, the max distance is the distance from $f3_1^-$ to the farthest vertex of a_3. In this example, vertex l is the farthest vertex. So the max distance from $f3_1^-$ to a_3 is the distance from $f3_1^-$ to vertex l. Similarly, we calculate min-max distance for all areas to make the Minmax table.

3.2 Unfixed-shape Area Skyline (UASky) Algorithm

Let $a.min(Fi)$ and $a.max(Fi)$ be the min and the max distance to the closest facility of Fi from area a, respectively.

Table 3.1: Minmax Table for Areas

	$F1^+$ (star)		$F2^+$ (triangle)		$F3^-$ (square)	
Area	*min*	*max*	*min*	*max*	*min*	*max*
a_1	41.4	174.4	31.0	162.9	−95.0	0.0
a_2	0.0	103.1	0.0	82.0	−91.0	0.0
a_3	44.0	157.8	37.1	152.3	−104.3	−43.3
a_4	44.2	158.8	43.0	155.1	−87.0	−31.0
a_5	13.3	51.6	27.6	57.7	−53.4	−23.3
a_6	18.1	70.4	32.2	92.3	−78.3	−22.2
a_7	8.8	51.6	0.9	43.0	−74.0	−46.0
a_8	32.6	38.7	28.0	31.0	−57.0	−51.0
a_9	10.7	32.6	14.4	28.0	−81.0	−57.0
a_{10}	0.0	152.6	0.0	158.1	−104.3	0.0
a_{11}	13.3	93.0	0.0	70.0	−147.6	−53.5
a_{12}	0.0	106.3	14.1	114.0	−81.3	−24.7
a_{13}	11.7	106.3	21.5	90.4	−153.4	−71.6

Definition 3.2.1 *(Unfixed-shape Area Skyline Query). Given two unfixed-shape areas, a and a', we say area a dominates area a' iff $a.max(Fi) \leq a'.min(Fi)$ in $\forall Fi \in F$. Skyline query for areas selects all non-dominated areas from the set of the disjoint areas.*

For example, area a_8 (see Figure 3.2 and Table 3.1) is dominated by the area a_9, because area a_9 has smaller max distance than that of min distance in $F1^+$, $F2^+$, and $F3^-$. Max distance of F is distance from the closest F facility to the farthest point in the area. If $a.max(F)$ is equal or smaller than $a'.min(F)$, it means any sub-areas of a have equal or smaller distance than area of a'. In this condition a' is dominated by area a,

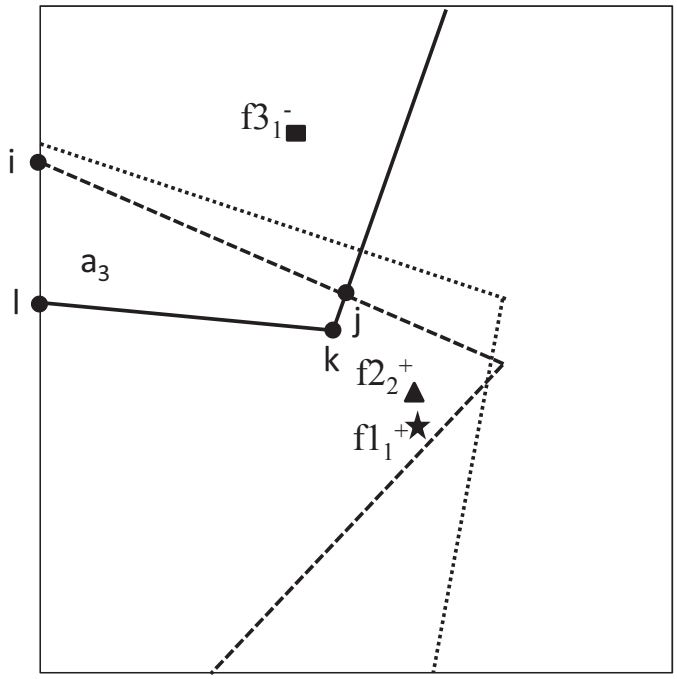


Figure 3.4: Corresponding facilities for area a_3

therefore, a' is not an area skyline.

On the other hand, for example, a_6 is not dominated by another area in our dominance condition. Though min distance (13.3) to $F1^+$ of a_5 is smaller than that (18.1) of a_6, max distance (51.6) of a_5 is not smaller than min distance (18.1) of a_6. It means that some sub-areas in a_6 may have smaller $F1^+$ distance value than those of sub-areas of a_5 and we should not eliminate a_6 because of a_5's $F1^+$ distance. In this case, some area in a_5 might be closer to $F1^+$ than area a_6, for example, area a_5 with distance 13.3 (min distance of a_5) to 18.1 (min distance of a_6), are closer than a_6.

After preparing the Minmax table, we can compute the skyline area using any conventional skyline algorithms by using the dominance condition of areas. Using any skyline algorithm, we eliminate a_8, and output the remaining areas, i.e., $a_1, ..., a_7, a_9, ..., a_{13}$ as the skyline area.

Input : $F_{1...k}$
Output : Sky
1. for each $F_k \in F$ do
2. build_voronoi $V(F_k)$
3. divide area by all $V(F_k)$ into m disjoint area $a_{1...m}$
4. assign ID for each vertex of divided areas $v_{1...n}$
5. for each $F_k \in F$ do
6. for each vertex $v_{i\ ,i=1..n}$
7. find the closest F_k in $V(F_k)$
8. for each $F_k \in F$ do
9. for each $a_{i\ ,i=1..m}$
10. calculate $dist_{min}(a_i, F_k)$ and record it into T
11. calculate $dist_{max}(a_i, F_k)$ and record it into T

// procedure to remove dominated areas

12. for each record in $T, t_{1...p}$ do
13. if $t_i.F_{k.max} \leq t_j.F_{k.min}$ for all $F_k \in F$ do
14. remove dominated area t_j from T
15. return Sky

Figure 3.5: Unfixed-shape Area Skylines Algorithm

Each user has different preference about the facilities, for example, one may prefer bus/train station compared to the university. The result of skyline area query contains preferable area of all such users. Figure 3.5 shows Unfixed-shape Area Skyline ($UASky$) Algorithm.

3.3 Experiment

There are two steps to find area skyline. Step one is to generate the Minmax table as in Table 3.1, and step two is to find skyline area using any skyline algorithm. Since the performance of the step two is not different from other conventional algorithm, in this section, we only considered the step one calculation.

Table 3.2: Parameters and Values

Parameters	*Values*
Number of all objects	9, 15, 30, 45
Number of disjoint areas	12, 26, 73, 113
Number of different facility types	3, 4, 5, 6

We have conducted a set of experiments to test our proposed algorithm. We build the system in a PC with Intel core i5 processor, 3.2GHz CPU, and 4GB main memory. We evaluated our algorithm on synthetic datasets. Each experiment is repeated five times and the average result is considered for performance evaluation. The parameters and values that have been used in our experiments are given in Table 3.2. The result are shown in Figure 3.6, 3.7, and 3.8.

3.3.1 Varying object number

In the first experiment, we evaluate the effect of number of objects on our proposed algorithm. We set the number of different facility types to 3, which consist of two types of desirable facilities and one type of undesirable facility, and vary the number of objects for each facility from 3 to 15 objects, so the total objects become 9, 15, 30, and 45. Recall that as the number of objects increase, the number of disjoint areas also increases accordingly. Figure 3.6 shows the correlation between quantity of objects and the processing time. From the result we observe that the processing time increases with the increase of object number.

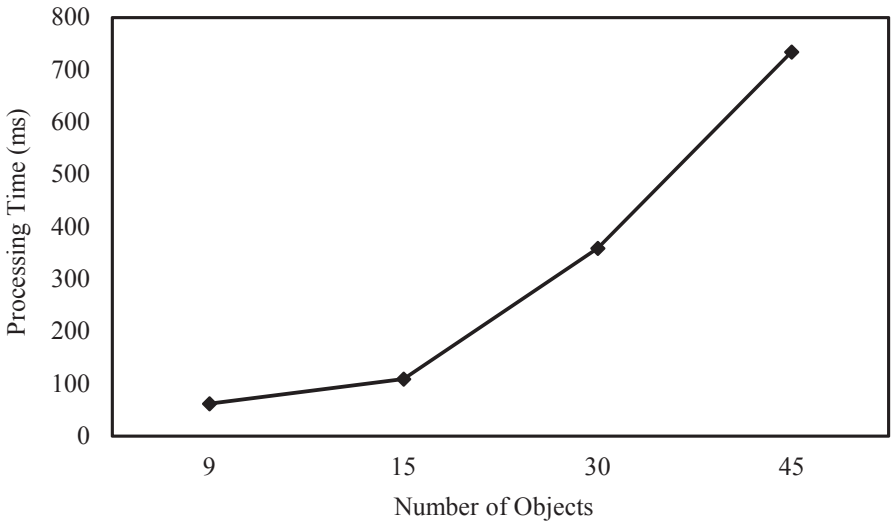


Figure 3.6: Performance for different number of objects

3.3.2 Varying the number of disjoint areas

The second experiment is a side effect of first experiment. For this experiment, again we set the number of different facility types to 3. In previous paragraph we have mentioned that the number of disjoint areas becomes large with the increase of object number. For the total objects 9, 15, 30, and 45, the number of disjoint areas become 12, 26, 73, and 113. As a result, it gradually increases the processing time. Figure 3.7 reports the result.

3.3.3 Varying the number of facility types (facility dimensions)

For the third experiment, we study the effect of number of facility types on our approach. For this experiment, we set the number of objects to 3 for each facility type and vary the number of facility types from 3 to 6. We set one type as undesirable facility for every set. Figure 3.8 shows the result of this experiment. The result indicates that as the number of facility types increases, the performance of the proposed method becomes slower.

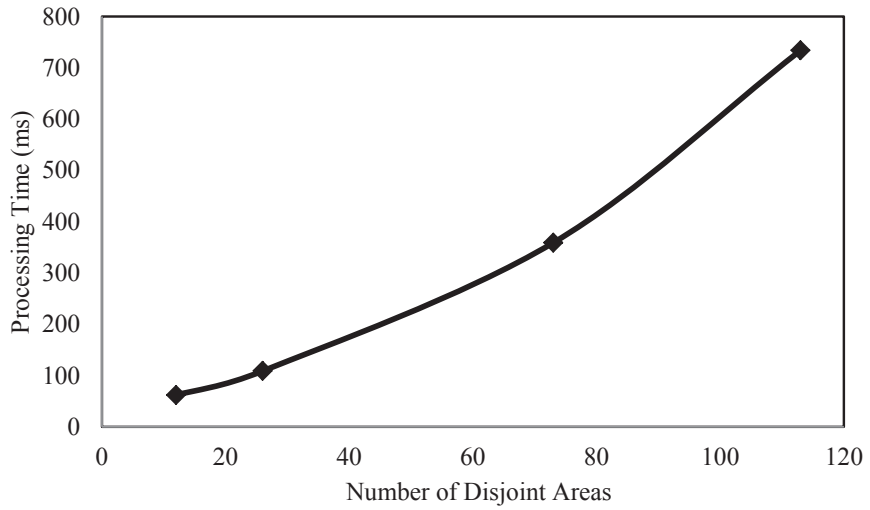


Figure 3.7: Performance for different number of disjoint areas

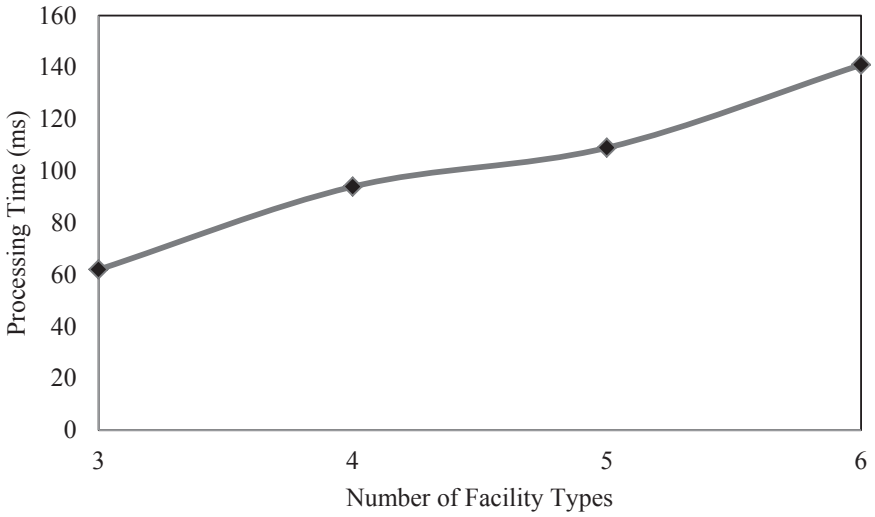


Figure 3.8: Performance for different facility dimensions

In summary all of the above experiments give the indication that the processing time depends on the number of objects and the number of facility types. If the number of facility types or the number of objects for each facilty increases then the processing time also increases. This happens because the large number of facilities as well as objects are responsible in increasing the number of Voronoi regions. As a result, the number of disjoint areas becomes larger, which increases computational overhead and requires more processing time to compute the area skyline.

In addition, we have examined the effect of the size of region that contains all query areas. We found that the size of the region does not affect the performance.

3.4 Concluding Remarks

Selecting desirable areas from an integrated geographical information system is important for various business applications. This chapter addresses a method to compute area skyline queries. Area skyline queries find some areas that are not dominated by another area according to its distance from desirable and/or undesirable facilities. Based on the extensive experiments, the proposed algorithm is affected by the number of facility types as well as the number of objects for each facility.

As we see in the motivated example, this method selects relatively many areas as skyline. A large area is likely to be selected as a skyline because a large area likely to have large max distance. One countermeasure for this property is to use a weight that has an effect to dominate larger area. Another countermeasure is to divide a large area into smaller areas. Currently, our area skyline algorithm is not so efficient to find reasonable number of area skylines. We are going to investigate this issue in our next work.

Chapter 4

Grid Based Area Skyline

Area skyline query is a method for selecting good areas, which are near to desirable facilities and far from undesirable facilities. It is an important method to select non-dominated area from the users perspective, who need some good locations based on his/her preference. In previous chapter, we introduced UASky algorithm to answer area skyline query. The drawback of UASky algorithm is that it selects relatively many areas as skyline. One of the main reason is because a large area likely to have large max distance. One countermeasure for this is to divide a large area into smaller areas. In this chapter, we have solved the drawback of UASky and improve its efficiency. We proposed an efficient and practical solution to the area skyline query using grid data structure, called Grid based Area Skyline (GASky) algorithm. We have conducted intensive experiments to prove the efficiency of our algorithm.

4.1 Problem definition

Let A be a rectangular target area, where the businessman wants to build his supermarket, on a map. Let $F = \{F1, ...Fm\}$ be a set of facility types, which can be categorized into m types. Each type is classified into desirable (annotated by + mark) or undesirable (annotated by − mark). Each facility has several facility objects, for example, a desirable facility $F1^+$ has three objects $F1^+ = \{f1_1^+, f1_2^+, f1_3^+\}$.

4.1.1 Grids and Vertexes

For simplicity, we assume the rectangular target area A is a square region. We, first, divide A into $s \times s$ grids. Figure 4.1 is an example of such grids. To identify a grid, we assign unique ID number to each of the grids from top-left $g_{(0,0)}$ to bottom-right $g_{(s-1,s-1)}$.

Each grid is surrounded by four vertexes, which we denoted as $v_{(i,j)}$, $v_{(i,j+1)}$, $v_{(i+1,j)}$, and $v_{(i+1,j+1)}$, for the top-left (tL), top-right (tR), bottom-left (bL), and bottom-right (bR) of $g_{(i,j)}$ ($0 \leq i \leq s$, $0 \leq j \leq s$), respectively.

4.1.2 Distance between Grid to Point

Given two points p and q in A, let $dist(p, q)$ be the Euclidean distance between p and q. We have to calculate the distance from a given query point q and a grid g in A. Since g is a two-dimensional region, we define two distance functions between g and q, $dist_{min}(g, q)$ and $dist_{max}(g, q)$, which we call "minimum (min) distance" and "maximum (max) distance", respectively. The definition of $dist_{min}(g, q)$ and $dist_{max}(g, q)$ are as

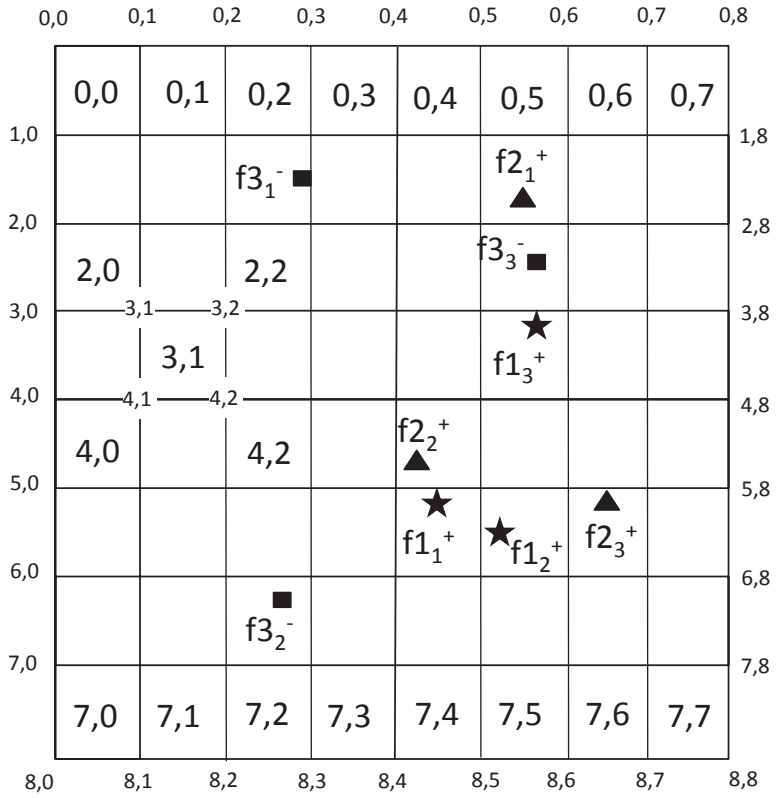


Figure 4.1: Targeted area divided into square grids

follows:

$$dist_{min}(g,q) = \begin{cases} 0 & \text{if } q \text{ lies inside } g. \\ min\{dist(p,q) \mid \forall p \text{ inside } g\} & \text{if } q \text{ lies outside } g. \end{cases}$$

$$dist_{max}(g,q) = max\{dist(p,q) \mid \forall\ p\ inside\ g\}$$

Figure 4.2 shows examples of min and max distance. Figure 4.2 (a) shows an example if the query point (black dot) is inside the grid. The min distance of the grid from the query point is 0, and the max distance is the distance from the query point to the farthest vertex, which is (*tR*), of the grid. In Figure 4.2 (b), the min distance from the query point (black dot) to the grid is the distance between the point to p^*, which is the closest point from the query point inside the grid. The max distance is the distance between the

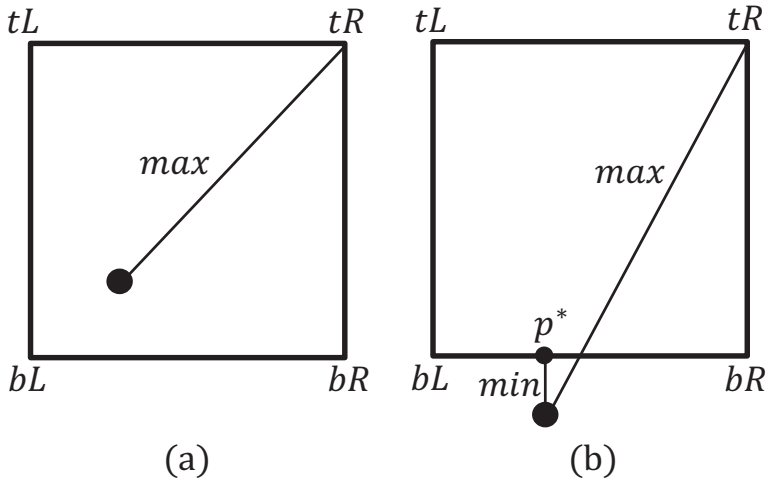


Figure 4.2: Min. and Max. distance between Grid and Point

query point to *tR*.

4.1.3 Voronoi Diagram

Given a query point q in A. To find the closest object of a facility type F from the query point, we used Voronoi Diagram as we used in UASky (see Figure 3.1).

4.1.4 Grid Dominance and Area Skyline

Let fk^{min*} be the object whose $dist_{min}(g, fk^{min*})$ is smaller than or equal to those of any other object in Fk. Similarly, let fk^{max*} be the object whose $dist_{max}(g, fk^{max*})$ is larger than or equal to those of any other object in Fk. Let $dist_{min}(g, Fk)$ and $dist_{max}(g, Fk)$ be the minimum distance and the maximum distance, respectively, from grid g to fk^{min*} and fk^{max*}.

Definition 4.1.1 *(Grid based Area Skyline Query). For two grids, g_i and g_j, we say g_i dominates g_j iff $dist_{max}(g_i, Fk) \leq dist_{min}(g_j, Fk)$ for all k ($1 \leq k \leq m$). Area skyline of A is the set of all non-dominated grids in A.*

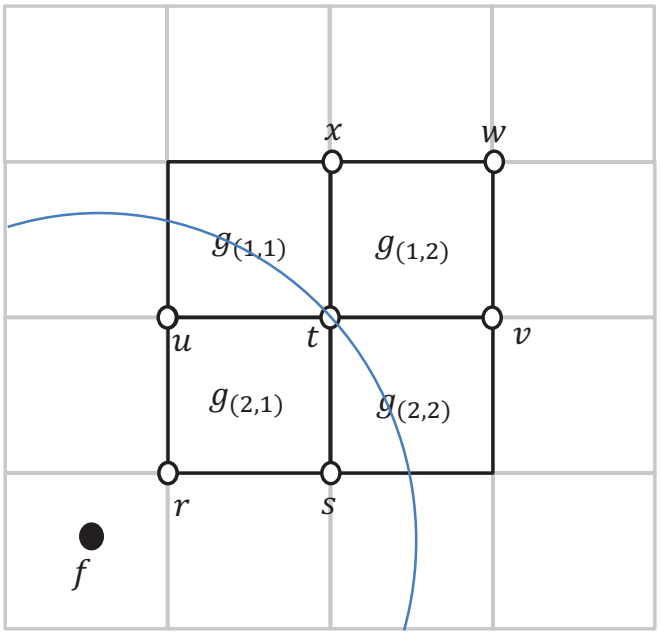


Figure 4.3: Grid dominance situation

Figure 4.3 illustrates the grid dominance. Let f, the black dot, be the nearest Fk facility for $g(1,1)$, $g(1,2)$, $g(2,1)$ and $g(2,2)$. For grid $g(1,1)$, $dist(f,u)$ is the minimum distance of Fk. Similarly, $dist(f,t)$, $dist(f,r)$, and $dist(f,s)$ are the minimum distance for $g(1,2)$, $g(2,1)$, and $g(2,2)$, respectively. For grid $g(1,1)$, $dist(f,x)$ is the maximum distance of Fk. Similarly, $dist(f,w)$, $dist(f,t)$, and $dist(f,v)$ are the maximum distance to $g(1,2)$, $g(2,1)$, and $g(2,2)$, respectively. The (blue) circle is the circle whose radius is the maximum distance from f to $g(2,1)$. As we can see in the figure, $g(1,1)$ and $g(2,2)$ intersect with this circle, which means that those two grids have smaller minimum distance from f. In this situation, we say that $g(2,1)$ and $g(1,1)$ are uncomparable with respect to f. Similarly, $g(2,1)$ and $g(2,2)$ are uncomparable. On the other hand, $g(1,2)$ does not intersect with the circle. In this situation, we say that $g(2,1)$ dominates $g(1,2)$ with respect to f.

4.2 Grid based Area Skylines (GASky) Algorithm

To handle the area skyline query, we first divide the target area into square grids. Next, for each grid, we calculate the minimum and maximum distances of each type of facilities. Using the min and max distances, we retrieve non-dominated grids, which are skyline areas.

4.2.1 Generate square-grid sub areas

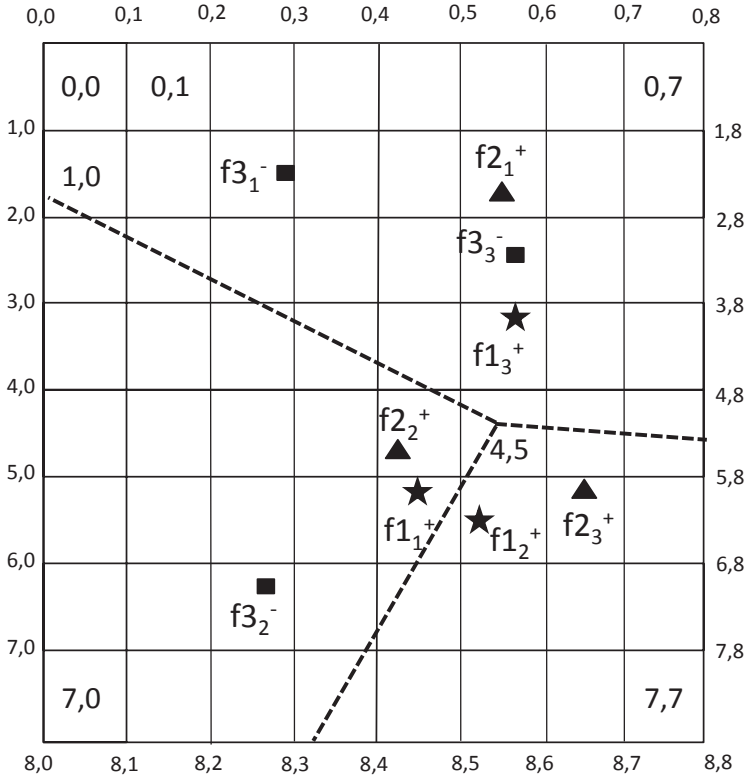
Suppose an area A in Figure 1.2 (a) has been divided into 8*8 grids ($s = 8$). Figure 4.1 shows the grids. Each of the grids and its surrounding vertexes have identification number. For example, grid $g(3, 1)$ is surrounded by vertexes $v(3, 1)$, $v(3, 2)$, $v(4, 1)$, and $v(4, 2)$.

4.2.2 Min-Max distance calculation

In this step, we build Voronoi diagram for each type of facilities. Consider Figure 4.1 again. In this figure, there are three types of facilities. Therefore, we need to build three Voronoi diagrams. Then, we find the closest facility for each type from each grid as follows.

At first, we find the closest facility from each of the surrounding vertexes. Figure 4.4 (a) shows an example of Voronoi diagram for facilities of type $F1^+$ (star symbol). Figure 4.4 (b) shows the closest $F1^+$ from each of the surrounding vertexes, $v(0, 0)$,..., $v(8, 8)$.

We calculate the minimum distance for Fk type for each grid as follows. First of all, for each Fk object, find the grid that contains the Fk object and set the minimum



(a)

Vertex ID	F1	Dist
0,0	$f1_3^+$	140
0,1	$f1_3^+$	115
0,2	$f1_3^+$	93
...	...	...
4,4	$f1_1^+$	...
4,5	$f1_3^+$	...
...	...	...
6,1	$f1_1^+$	...
7,4	$f1_2^+$	...
...	...	...
8,7	$f1_2^+$	...
8,8	$f1_2^+$	...

(b)

Figure 4.4: (a) Voronoi Diagram for facility $F1^+$ (*star*) (b) Closest $F1^+$ for each vertex

distance for Fk of the grid to zero. Next, for each non-zero grid g, we calculate the minimum distance by using the distances of the vertexes to their closest facility object. These distances are recorded in a table like Figure 4.4 (b), that shows the distance from vertexes to their closest facility type $F1^+$. Since a grid has four vertexes and each vertex can be located in different Voronoi cell, the closest object of each facility type for each vertex on one grid may be different. For example, Grid $(4,5)$ in Figure 4.4 (a), the closest $F1^+$ facility of surrounding vertex $v(4,5)$ and $v(4,6)$ is $f1_3^+$. The closest $F1^+$ facility of $v(5,5)$ is $f1_1^+$. The closest $F1^+$ facility of $v(5,6)$ is $f1_2^+$.

Figure 4.5 shows how to calculate minimum and maximum distance for non-zero grid. Let (x,y), (x',y), (x,y'), and (x',y') be the coordinate of four vertexes of grid g.

Since we have calculated minimum distance of all vertexes like in Figure 4.4 (b), let's assume that (x, y), depicted by the white dot, has the smallest distance value among the 4 surrounding vertexes of g. Let (a, b), depicted by the black dot, be the coordinate values of the closest Fk object from (x, y). The distance between (a, b) and (x, y) is not always be the minimum distance of (a, b) to g.

There are three cases to calculate minimum distance of a grid, which will be explained as follows. Let (x', y) and (x, y') be coordinate values of the two adjacent vertexes of (x, y). First case, if a is between x and x' and b is not between y and y', then the minimum distance value is the difference between y and b. Second case, if a is not between x and x' and b is between y and y', then the minimum distance value is the difference between x and a. Otherwise, for the third case, the minimum distance value of g to Fk is the Euclidean distance between (x, y) and (a, b). In Figure 4.5, minimum distance of g for Fk (the closest Fk is (a, b)) is $|b - y|$, shown as straight line. Figure 4.6 (a), (b), (c) shows the three cases to calculate minimum distance of a grid. The maximum distance of g for Fk (the closest Fk is (a, b)) is the distance from (a, b) to the farthest surrounding vertexes of g, which is (x', y'). The dashed line in Figure 4.5 shows the maximum distance.

4.3 Computational Cost Analysis

Note that step to remove dominated areas in both algorithm is using the same conventional skyline algorithm, therefore the computational cost for this step is the same for UASky and GASky and is not included in this calculation. Both UASky and GASky have same procedures which are generating Voronoi diagram for each type, finding the closest facility for each type to each vertex, and calculating minimum and maximum

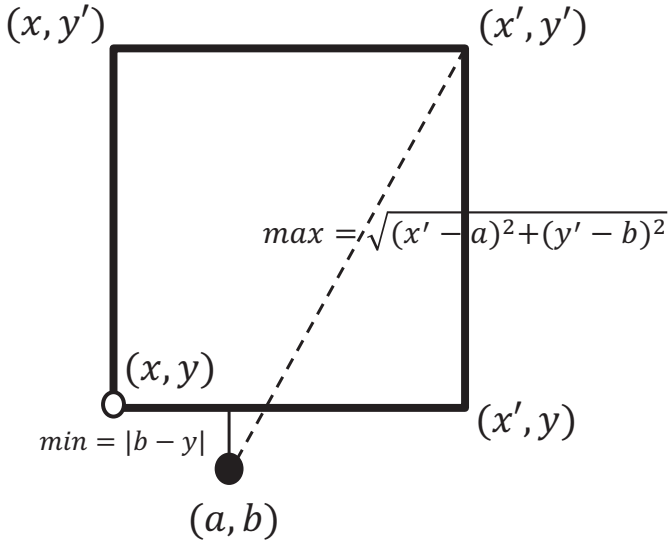


Figure 4.5: Min-max calculation for non-zero grid

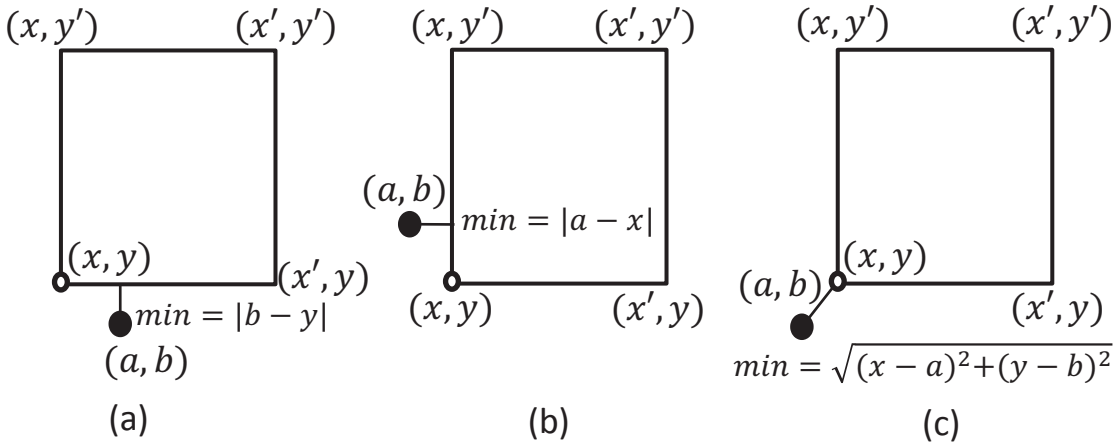


Figure 4.6: Minimum distance calculation

distance for each area. The difference between UASky and GASky is that in GASky we need to generate $s \times s$ grids, while in UASky we need to divide the whole area by all Voronoi diagrams to get disjoint areas (step 3 in UASky).

In the computational geometry literature, following Voronoi diagram's properties have been studied. The worst time complexity to build a Voronoi diagram of n points is $O(n \log n)$ and the expected time complexity to find the nearest Voronoi point is

$O(\log n)$ [12]. If we utilize a quaternary tree together with a Voronoi diagram, the expected time complexity to construct a Voronoi diagram can be reduced to $O(n)$ and the time complexity to find the nearest Voronoi point can be reduced to $O(1)$ [30].

Let m be the total number of facility types, n be the number of objects in each type. The expected time complexity to construct m Voronoi diagrams for GASky and UASky is $O(mn)$. The time to find the nearest facility for each point is $O(m)$.

Since there are $s \times s$ grids in GASky, there are $O(s^2)$ surrounding points. Therefore, GASky takes $O(s^2m)$ in addition to the Voronoi diagrams' construction time. As for UASky, there are $O(mn)$ Voronoi edges in total. The number of intersections of $O(mn)$ edges is $O((mn)^2)$, which are the number of surrounding vertexes. Therefore, UASky takes $O((mn)^2)$ in addition to the Voronoi diagrams' construction time. After calculating minimum and maximum distance for each facility type for each grid, we record them into Minmax table T.

4.3.1 Calculate Non-dominated Grid

To simplify the skyline query calculation, we multiply the min and max distance values for undesirable facilities by -1. After this simplification, smaller value is better in each of the distance values. We record $dist_{min}$ and $dist_{max}$ for each grid in the Minmax table T, then calculate area skyline from T using grid dominance condition in Section 4.1.4.

Figure 4.7 shows Grid-based Area Skyline ($GASky$) Algorithm. After calculating min and max distance for each grid, we can retrieve the skyline records from the Minmax table T as area skylines.

```
Input    : F_{1...k}, s
Output   : Sky
1.  for each F_k ∈ F do
2.      build_voronoi V(F_k)
3.      for each vertex v(i,j) from v(0,0) to v(s,s)
4.          find the closest F_k in V(F_k) from v(i,j)
5.      for each grid g(i,j) from g(0,0) to g(s−1,s−1)
6.          calculate dist_min(g(i,j), F_k) and record it into T
7.          calculate dist_max(g(i,j), F_k) and record it into T

    // procedure to remove dominated grids
8.  for each record in T, t_{1...p} do
9.      if t_i.F_{k.max} ≤ t_j.F_{k.min} for all F_k ∈ F do
10.         remove dominated grid t_j from T

11. return Sky
```

Figure 4.7: Grid-based Area Skylines Algorithm

4.3.2 UASky

Through intensive experiments in unfixed-shape area method in our previous chapter, we found that a large area is likely to be selected as skyline. This is because a large area have larger max distance which makes it difficult to be dominated. Moreover, in UASky we cannot change the shape, size and number of disjoint areas because they are produced by intersection of all Voronoi diagrams. Thus, user can not control the number of skyline areas in UASky. In contrast, in GASky user can control the number of grid, which can prevent the problem in UASky. We will discuss this issue later in Section 4.4.

4.4 Experimental Evaluation

In this section, we conduct four experiments to examine selectivity and performance. We performed our experiments in a PC with Intel Core i5 3.2GHz processor with 4GB of

RAM. We evaluated our algorithm using synthetic datasets. Each experiment is repeated ten times and we reported the average.

Since the step to remove dominated areas is the same for both UASky (Figure 3.5 step 8-10) and GASky (Figure 4.7 step 12-14), and the performance of this step is not different from other conventional skyline algorithm, we exclude it from the processing time calculation.

4.4.1 Comparison between GASky and UASky

In these experiments, we compared the performance of the proposed algorithm (GASky) and our previous algorithm (UASky) in the feasibility study [2].

First, we compared processing time of GASky with that of UASky, and second, we compared ratio of skyline areas of both algorithm. We also examined the effect of number of facility type and number of objects in both algorithm.

We used two different synthetic data, say $DB1a$ and $DB1b$ for these experiments.

For $DB1a$, the default number of objects is 128. We varied the number of types to 2, 4, 8, 16, and 32 respectively. In these experiments, the number of desirable types is set to be the same as the number of undesirable types. For $DB1b$, we fixed the number of facility type is 2. Then, we varied the number of objects to 8, 12, 24, 48, 96, and 128.

In $DB1a$, the number of disjoint areas resulted by UASky was around 1800 and in $DB1b$ it was around 260 disjoint areas. In the first experiment, we set the number of grids in GASky so that the number of grids in GASky becomes almost same to the number of disjoint areas produced by UASky.

The results of first experiments are shown in Figure 4.8 and 4.9.

From these figures, we can see that GASky is faster than UASky especially when

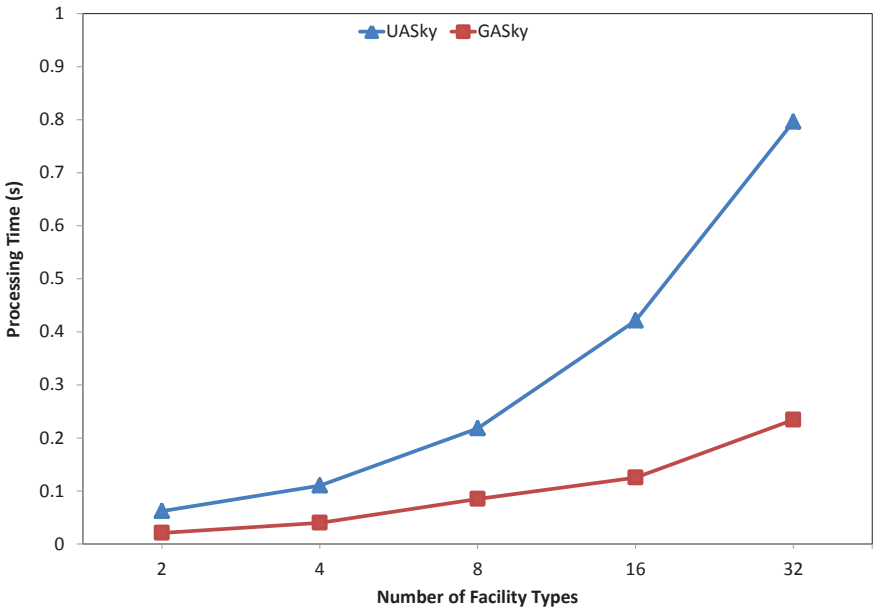


Figure 4.8: Processing time of *DB*1*a*

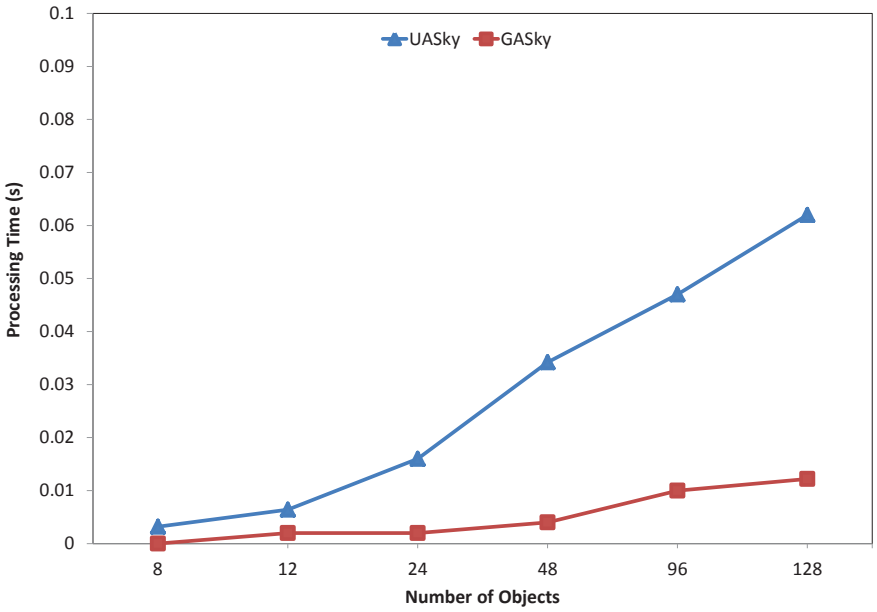


Figure 4.9: Processing time of *DB*1*b*

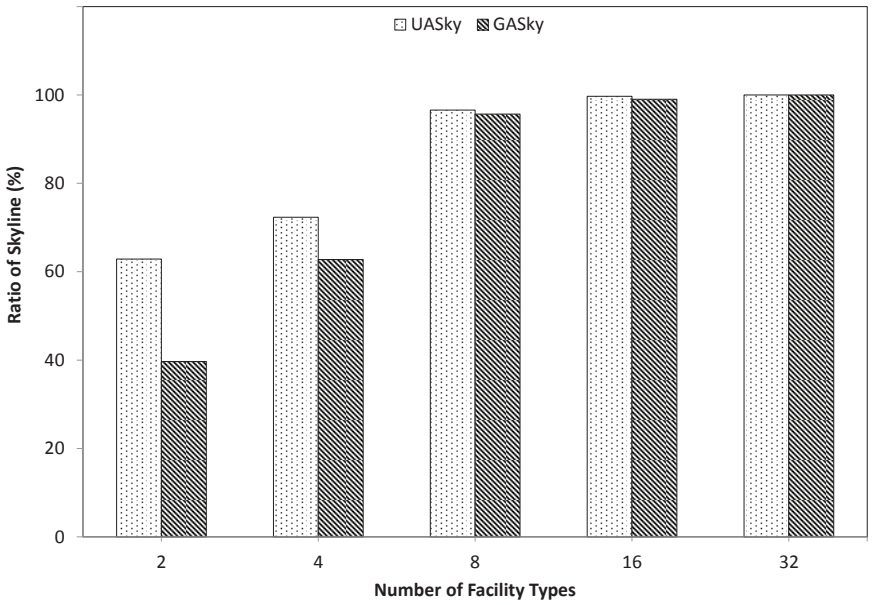


Figure 4.10: Skyline Ratio of $DB1a$

the number of types become large. We can also observe that the number of facility type affects the processing time more than the number of object. One of the main reason is that the number of facility type affects the time to build Voronoi diagrams. The ratios of skyline areas of this experiment are reported in Figure 4.10 and 4.11.

From these figures, we can see that using the similar number of grids, GASky has better skyline ratio than UASky. In these figures, skyline ratio decreases up to 40%.

In order to have better skyline ratio, in the second experiments, using $DB1a$ and $DB1b$, we set the number of grids in GASky to be 10000 and 2500. The ratios of skyline areas are reported in Figure 4.12 and 4.13. Figure 4.12 and 4.13 shows that increasing the number of grids can reduce the skyline ratio until 5%.

Figure 4.12 and 4.13 also shows that GASky is sensitive to the increase of facility types rather than the increase of objects. One of the main reasons is as follows. The increase of the facility types with fixed number of objects causes decrease of density of

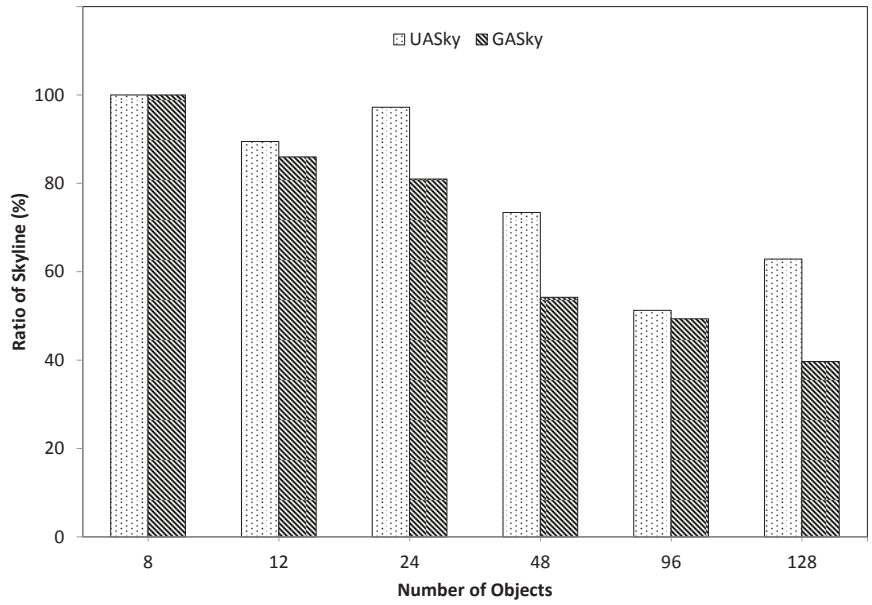


Figure 4.11: Skyline Ratio of *DB*1*b*

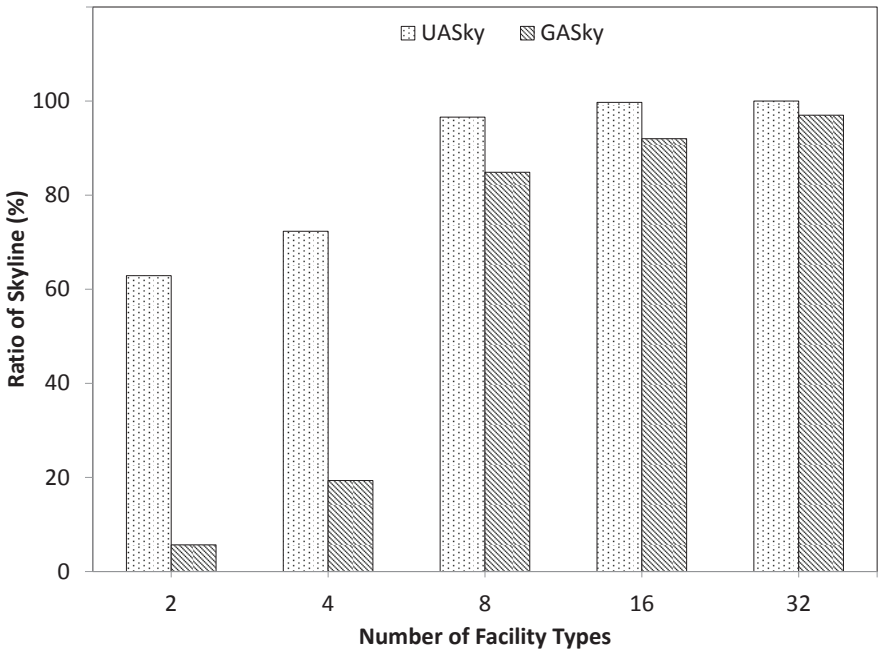


Figure 4.12: Skyline Ratio of *DB*1*a* with 10.000 grids for GASky

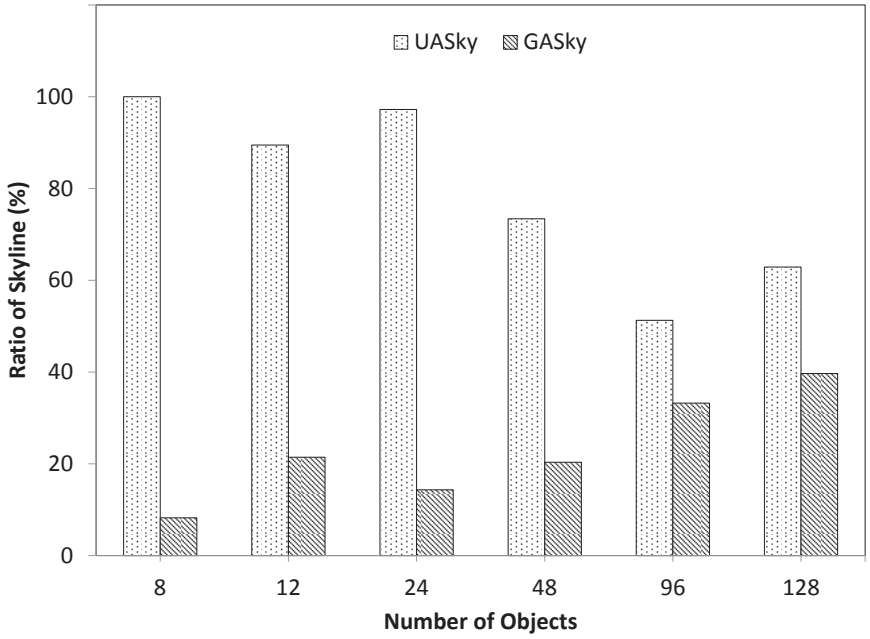


Figure 4.13: Skyline Ratio of *DB*1*b* with 2500 grids for GASky

each facility. It is equivalent to enlarging each grid, which tends to increase the ratio of skyline areas.

4.4.2 Effect on Grid number

In these experiments, we used four different synthetic data, say *DB*2*a*, *DB*2*b*, *DB*2*c*, *DB*2*d*. *DB*2*a* is random data consists of 128 objects with two types of facilities, one is desirable and the other is undesirable facility. *DB*2*b* has the same types of facilities but consists of 256 objects. *DB*2*c* and *DB*2*d* consist of 128 and 256 objects respectively, with four types of facilities, two types are desirable and the others are undesirable facilities.

For each data, we varied the number of grids to 144, 256, 400, 625, and 900. Figure 4.14 and 4.15 shows the results.

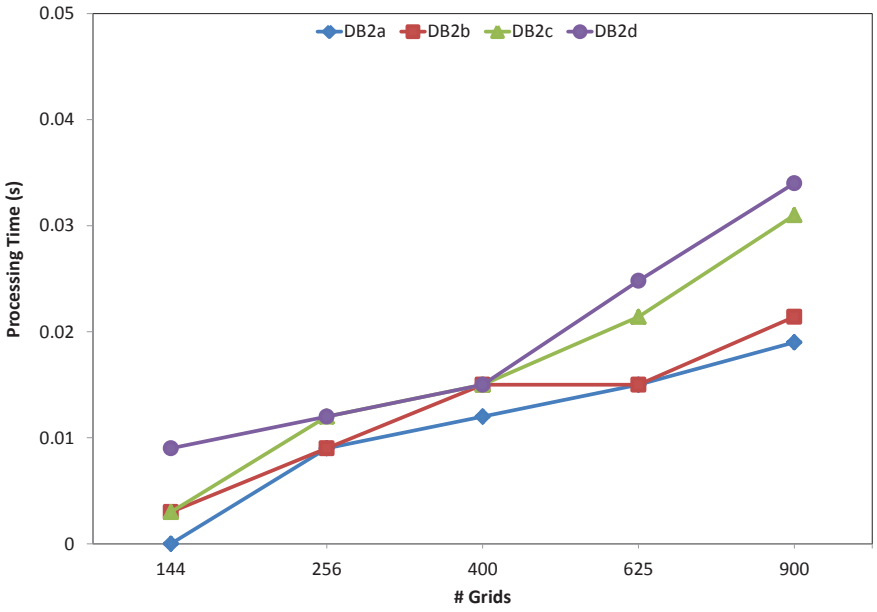


Figure 4.14: Processing time varied with number of grids

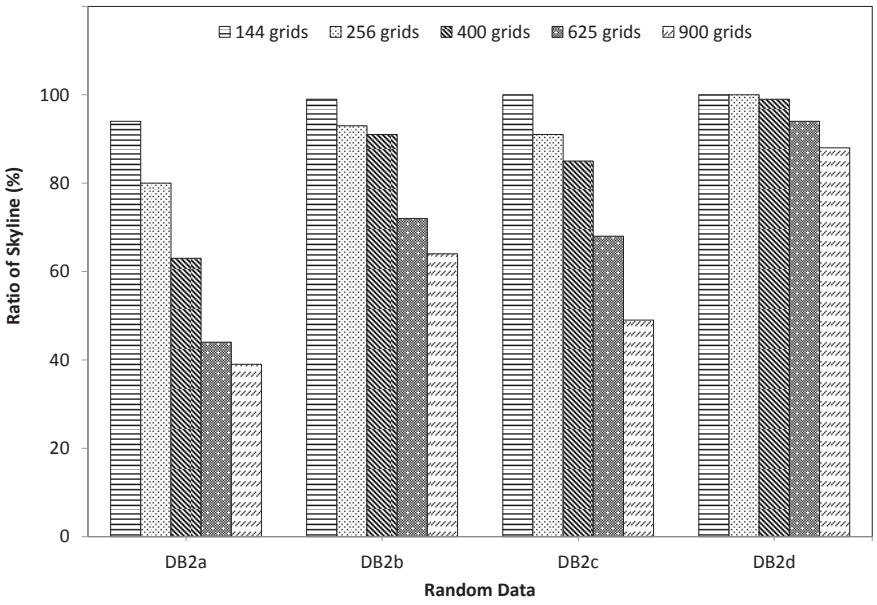


Figure 4.15: Ratio of Skyline varied with number of grids

From the results in Figure 4.14, we can observe that the processing time increases with the increase of the number of grid, and data that has more facility types and more number of objects also has longer processing time. The results in Figure4.15 illustrate that the ratio of skyline decreases with the increase of the number of grids, and data that has smaller number of facility types and number of objects decreases the ratio of skylines. In other words, we can decrease the ratio of skyline area by increasing the number of grids. Thus, higher number of grid means smaller size of each disjoint area, which in turn will decrease the ratio of skyline. By applying grid data structure, the GASky can control the number of area skyline by changing the number of grids.

In actual usage scenario, if a user prefers selective areas, she/he had better increase the s, which tends to reduce the ratio of skyline areas. Since GASky is sensitive to the increase of facility types, user should use larger s to reduce skyline ratio if she/he increases the number of facility type.

In our motivating example, UASky generated 13 disjoint areas with the ratio of skyline was 100%. Using the same method above, we applied GASky with 225 grids and the skyline ratio was decreased to 30%. Figure 4.16 and 4.17 illustrates skyline area after applying UASky and GASky.

4.4.3 Effect of Ratio of Desirable and Undesirable Types

In this experiment, we investigated the effect of ratio of desirable (or undesirable) facility among all facilities in UASky and GASky. In this experiment, we set the total objects to 100, set the number of facility type to 10, and varied the number of desirable and undesirable facility type to $(10+, 0-)$, $(8+, 2-)$, $(6+, 4-)$, $(4+, 6-)$, $(2+, 8-)$, and $(0+, 10-)$. Figure 4.18 shows the results. We can see that the difference of the ratio has

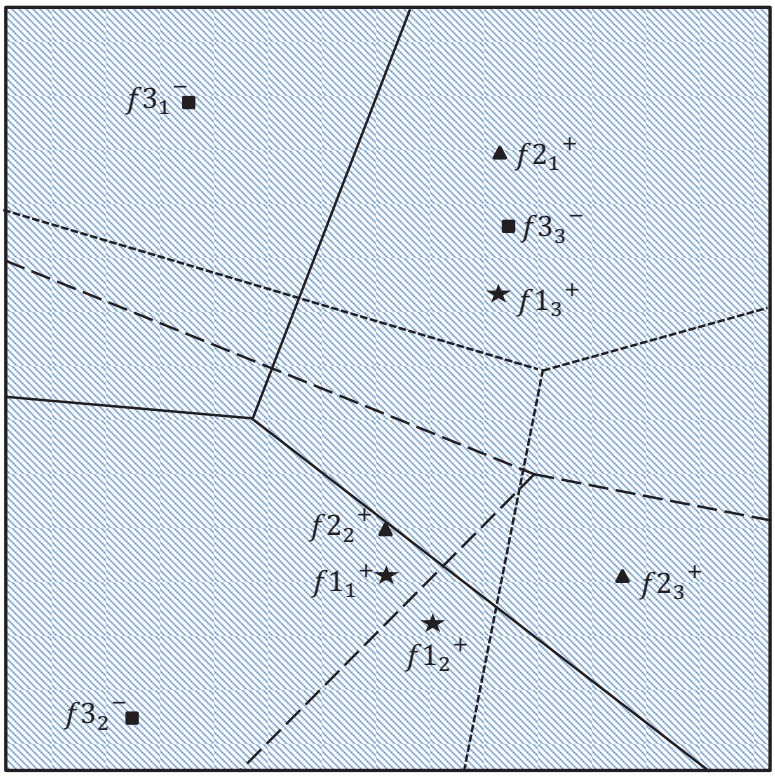


Figure 4.16: All targeted areas retrieved as skyline area using UASky

no significant effect to the processing time.

Moreover, this result also shows that GASky has better performance than UASky for different ratio of desirable and undesirable facilities.

4.4.4 Scalability

In this experiment, we examined the scalability of the proposed algorithm. For this purpose, we set the default number of facility's types to 4, among which two types are desirable facilities and the other two types are undesirable facilities. We set the number of grids to 100. We varied the number of total objects to 100K, 200K, 300K, 400K, and 500K, respectively. Figure 4.19 shows the results. In summary, all of the above experiments give the indication that the processing time depends on the number

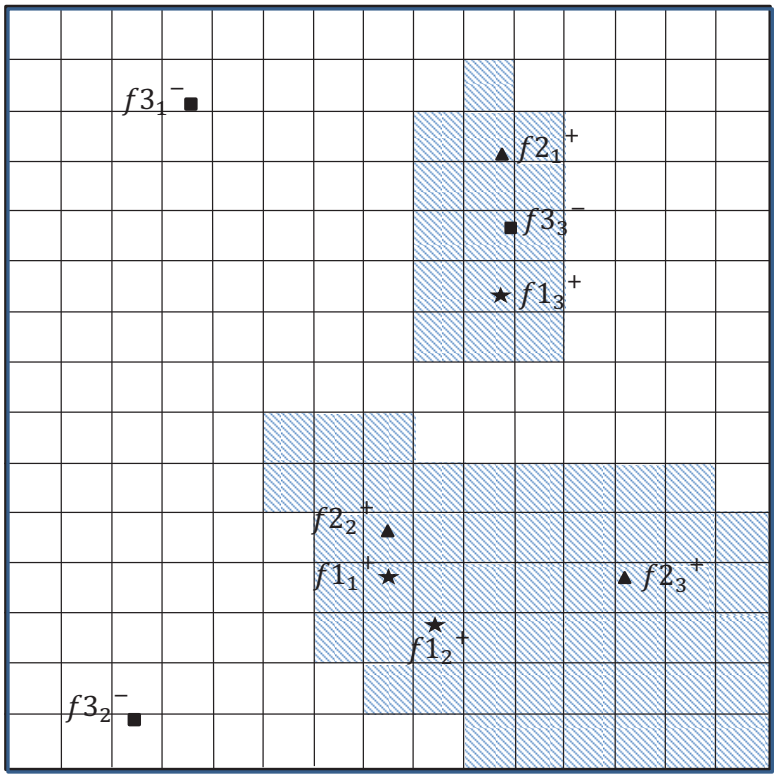


Figure 4.17: Decreased skyline area by using GASky

of objects, the number of facility types, and the number of grids. The processing time increases with the increase of the number of facility types, the number of objects, and the number of grids.

4.5 Concluding Remarks

Areas which are close to desirable facilities and far from undesirable facilities are important for various applications. The proposed area skyline queries help to find such areas, which are not dominated by another area.

This chapter addresses a method to compute area skyline query using grid data structure. Comprehensive experiments are conducted to demonstrate the effectiveness and

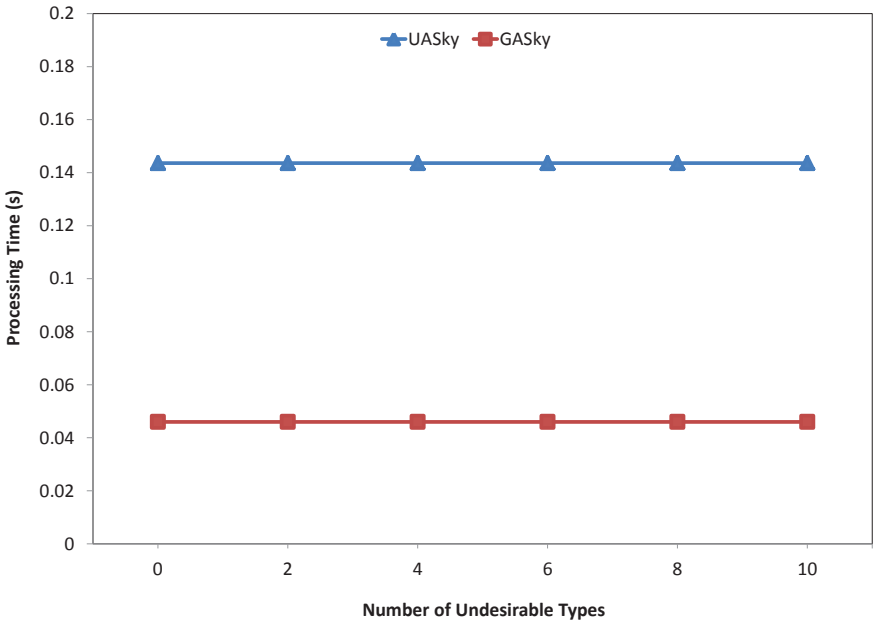


Figure 4.18: Effect of Desirable and Undesirable Types' Ratio

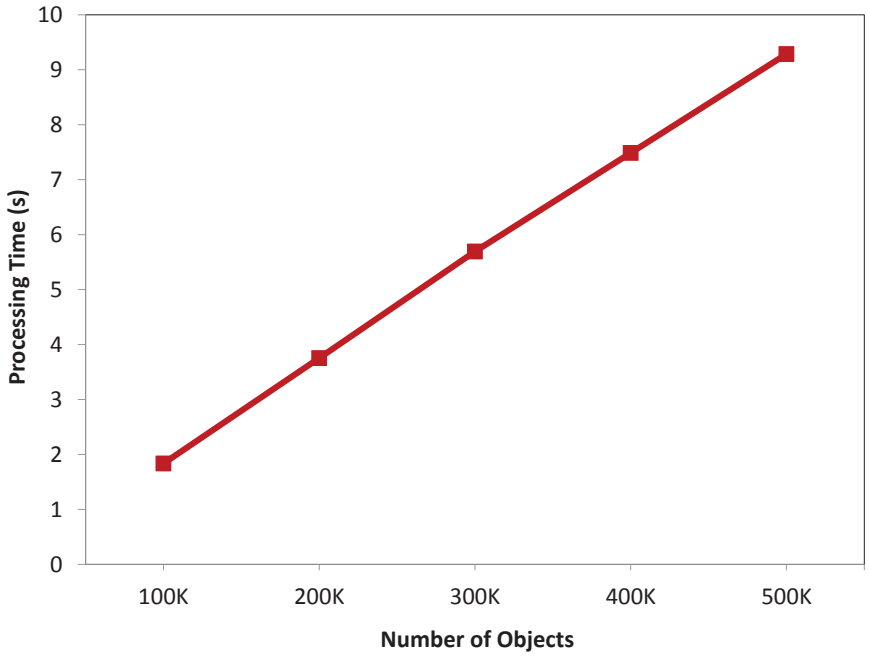


Figure 4.19: Scalability of GASky Algorithm

efficiency of the algorithm.

In future, we will extend our new area skyline query to answer area selection based on owner's perspective. We will also consider the challenging related open problems such as considering more than one object for each facility type, selecting k-dominant areas and how to utilize non-spatial property such as population density, price, etc., in the selection of areas.

Chapter 5

Reverse Area Skyline

Area skyline query is an important method to select non-dominated area from the users' perspective, who need some good locations based on his/her preference. Assume a real estate company has an area g (grid (2,18) in Figure 5.1) to develop apartment, office, or market complex. The company needs to know who will be interested in the area. By using the idea of "reverse skyline", reverse skyline areas of g can be identified. Grey grids in Figure 5.2 are dynamic area skyline of grid $(1, 14)$, while grey grids of Figure 5.3 are reverse area skyline of g. Notice that g is a dynamic area skyline of grid $(1, 14)$, so that grid $(1, 14)$ is a reverse area skyline of g. On the analogy of the utilization of "reverse skyline", the "reverse" skyline areas has invaluable information. Let us consider a real estate company that have an area g (grid (2,18) in Figure 5.3). Information about reverse area skyline, shaded area in Figure 5.3, must be useful for such company to consider effective real estate developments so that the area attracts many buyers. Reverse area skyline query can also be used for selecting promising buyers of the area, since it may give the company clues to find who will be interested in the area. In addition, it also may help to predict what type of business that would be suitable

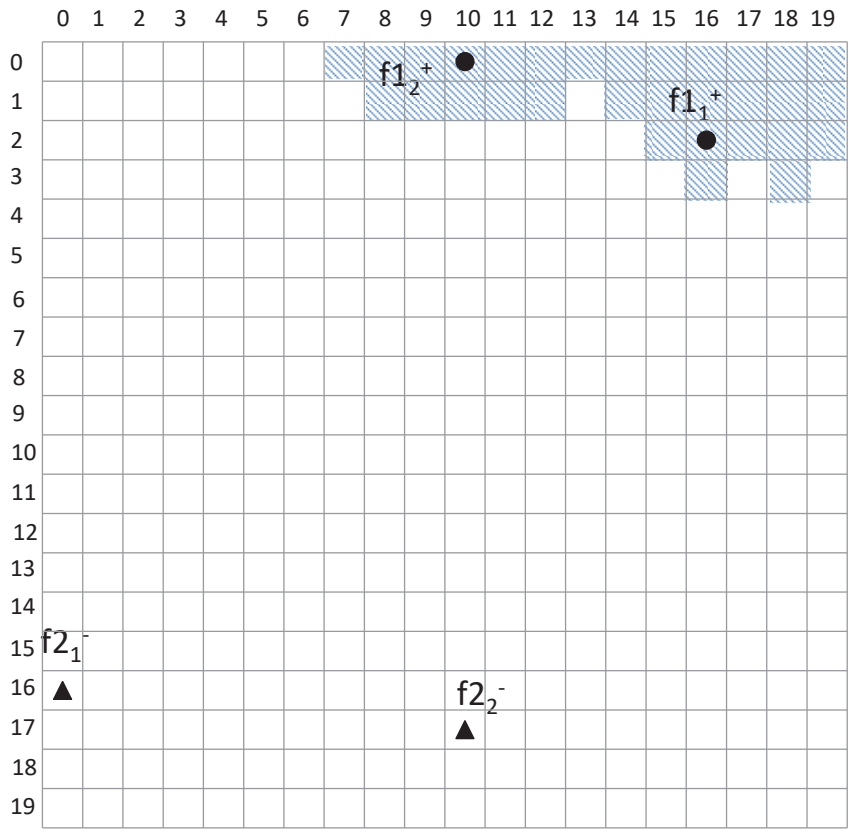


Figure 5.1: Area Skyline Queries

for the area considering the type of business that had already exist in the reverse area skylines.

In this chapter, we present the reverse area skyline query and propose an effective and efficient method to answer reverse area skyline problem.

5.1 Problem Definition

Let A be a rectangular target area in which there are spatial objects. Each spatial object can be categorized into one of m facility types. Let Fk be a set of type k ($k = 1, ...m$) objects, which are $Fk = \{fk_1, fk_2, ..., fk_{n_k}\}$ where n_k is the number of objects of the type

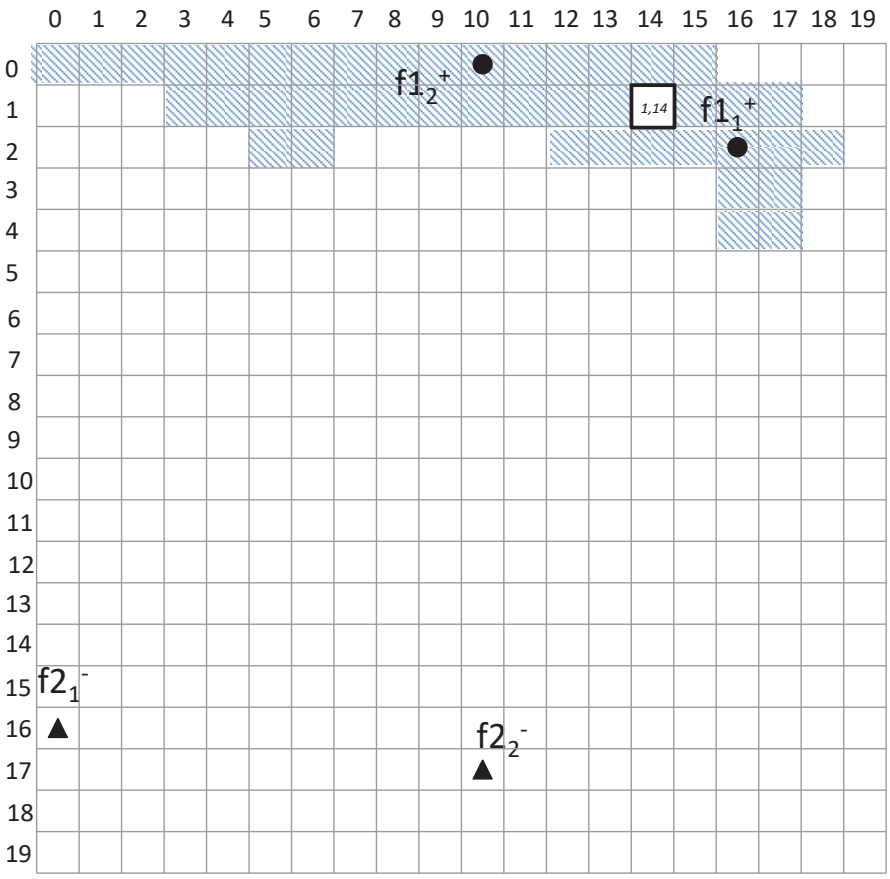


Figure 5.2: Dynamic Area Skyline of grid (1,14)

k facility.

5.1.1 Grids and Vertexes

We divide A into $s \times t$ square grids where s is the number of rows, and t is the number of column. We can identify each grid using row number and column number. For example, $g_{i,j}$ is a grid that lies in the i-th row and the j-th column. Each square grid is surrounded by four vertexes, each of which can also be identified by row number and column number. For example, top-left, top-right, bottom-left, and bottom-right vertex of $g_{i,j}$ can be identified as $v_{i,j}$, $v_{i,j+1}$, $v_{i+1,j}$, and $v_{i+1,j+1}$, respectively. In Figure 5.4, we

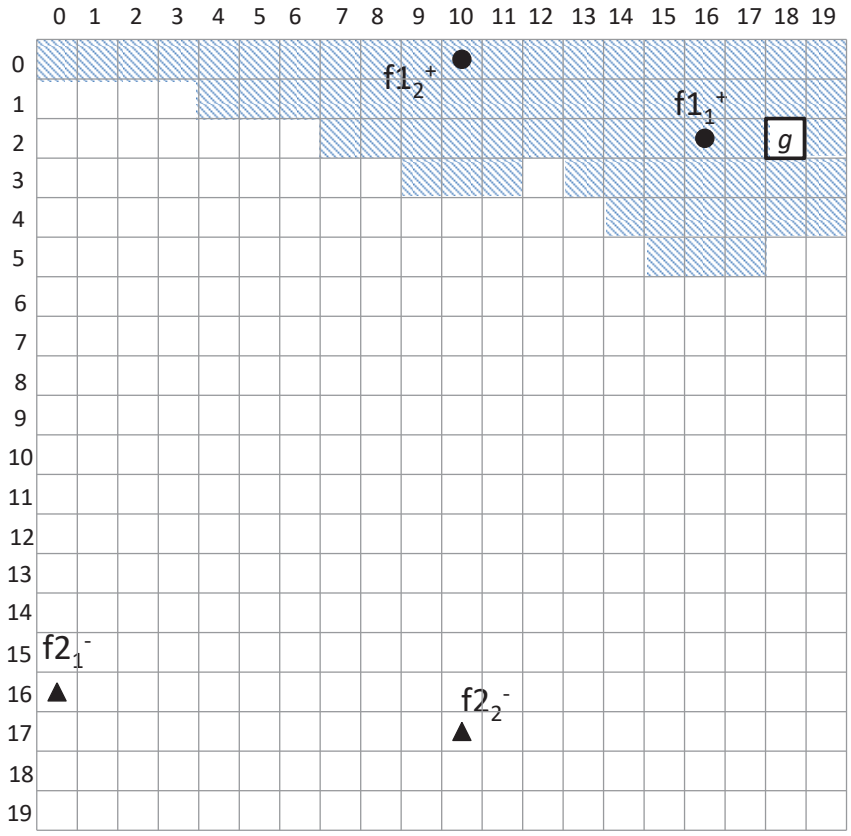


Figure 5.3: Reverse Area Skyline Result

defined the query grid g, and divided an area into 12x12 grids. $g_{4,6}$ is surrounded by four vertexes $v_{4,6}$, $v_{4,7}$, $v_{5,6}$ and $v_{5,7}$, respectively. For each vertexes, we find the nearest object of each facility type using Voronoi diagram. After that, we calculate min and max distance from each grid using the same calculation in GASky step 1 as discussed in Section 4.2.2, and record the distances in the Minmax table. From now, we call one record in Minmax table as one object.

5.1.2 Dynamic Area Skyline

Let $min(d_k(g))$ and $max(d_k(g))$ be the the min and max distance to facility f_k of grid g, and $min(d_k(q))$ and $max(d_k(q))$ be the the min and max distance to facility f_k of query

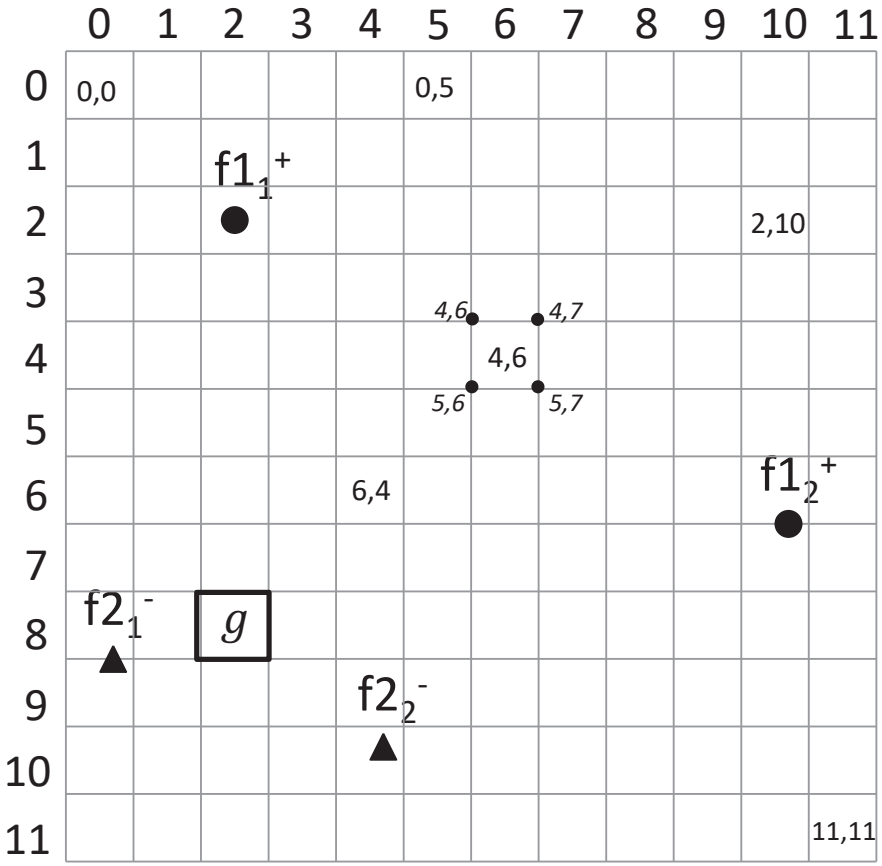


Figure 5.4: Target area divided into 12 x 12 grids

grid q. In order to calculate dynamic area skyline, we need to transform the distances similar to conventional dynamic skyline. Let $min(d_k(g))^T$ and $max(d_k(g))^T$ are the transformed min and max distance, respectively. There are six cases to transform $min(d_k(g))$ and $max(d_k(g))$ w.r.t query grid q into $min(d_k(g))^T$ and $max(d_k(g))^T$. Figure 5.5 illustrates the transformation of six cases in dynamic area skyline.

In all cases, we assume $min(d_k(q))$ and $max(d_k(q))$ are 5 and 8, respectively. In case 1, assume $min(d_k(g))$ and $max(d_k(g))$ are 9 and 11. Since 9 and 11 are larger than 8, then $min(d_k(q))^T$ and $max(d_k(q))^T$ become 1 (9-8) and 3 (11-8). In case 2, assume $min(d_k(g))$ and $max(d_k(g))$ are 6 and 11. Since 6 is between 5 and 8, and 11 is larger than 8, then $min(d_k(q))^T$ and $max(d_k(q))^T$ become 0 and 3 (11-8). In case 3, assume $min(d_k(g))$

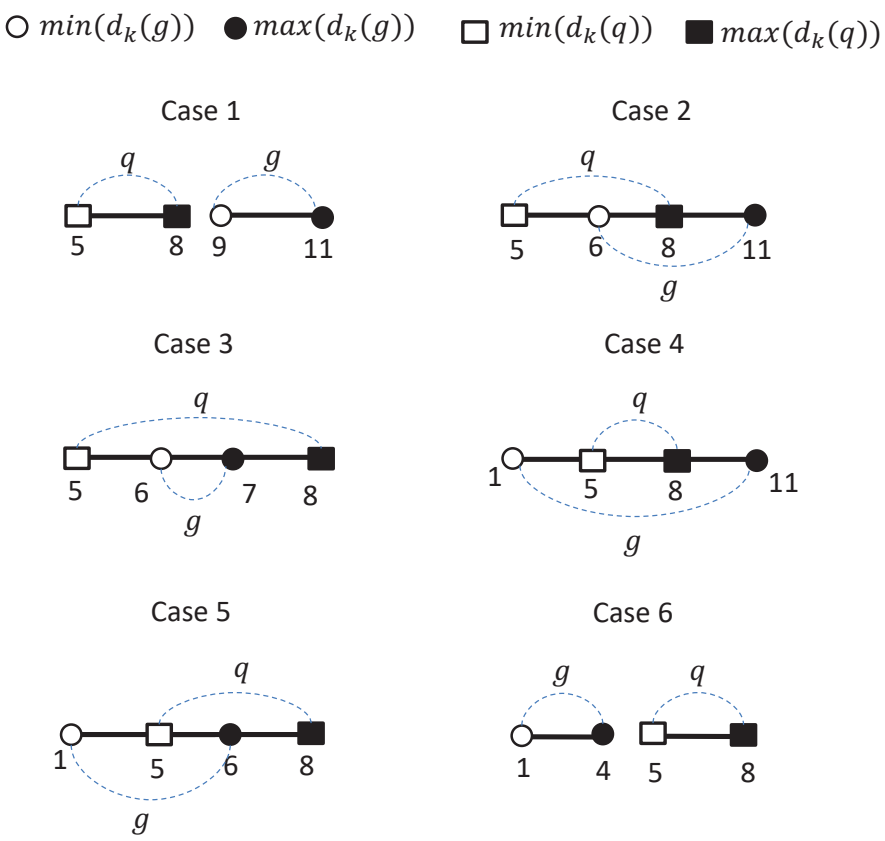


Figure 5.5: Transformation cases

and $max(d_k(g))$ are 6 and 7. Since 6 and 7 are between 5 and 8, then $min(d_k(q))^T$ and $max(d_k(q))^T$ become 0. In case 4, assume $min(d_k(g))$ and $max(d_k(g))$ are 1 and 11. Since 1 is smaller than 5 and 11 is larger and 8, then $min(d_k(q))^T$ and $max(d_k(q))^T$ become 0 and 3, just like in case 2. In case 5, assume $min(d_k(g))$ and $max(d_k(g))$ are 1 and 6. Since 1 is smaller than 5 and 6 is between 5 and 8, then $min(d_k(q))^T$ and $max(d_k(q))^T$ become 0 and 4 (5-1). In case 6, assume $min(d_k(g))$ and $max(d_k(g))$ are 1 and 4. Since 1 and 4 are smaller than 5, then $min(d_k(q))^T$ and $max(d_k(q))^T$ become 1 (5-4) and 4 (5-1).

We then formally defined Case 1 to 6 as:

if $min(d_k(g)) \geq max(d_k(q))$, then

$$\begin{aligned} min(d_k(g))^T &= min(d_k(g)) - max(d_k(q)) \\ max(d_k(g))^T &= max(d_k(g)) - max(d_k(q)) \end{aligned} \tag{5.1}$$

if $max(d_k(g)) > max(d_k(q))$ and $max(d_k(q)) > min(d_k(g)) \geq min(d_k(q))$, then

$$\begin{aligned} min(d_k(g))^T &= 0 \\ max(d_k(g))^T &= max(d_k(g)) - max(d_k(q)) \end{aligned} \tag{5.2}$$

if $min(d_k(g)) \geq min(d_k(q))$ and $max(d_k(g)) \leq max(d_k(q))$, then

$$\begin{aligned} min(d_k(g))^T &= 0 \\ max(d_k(g))^T &= 0 \end{aligned} \tag{5.3}$$

if $min(d_k(g)) < min(d_k(q))$ and $max(d_k(g)) > max(d_k(q))$, then

$$\begin{aligned} min(d_k(g))^T &= 0 \\ max(d_k(g))^T &= max(d_k(g)) - max(d_k(q)) \end{aligned} \tag{5.4}$$

if $min(d_k(g)) < min(d_k(q))$ and $min(d_k(q)) < max(d_k(g)) \leq max(d_k(q))$, then

$$\begin{aligned} min(d_k(g))^T &= 0 \\ max(d_k(g))^T &= min(d_k(q)) - min(d_k(g)) \end{aligned} \tag{5.5}$$

if $max(d_k(g)) \leq min(d_k(q))$, then

$$\begin{aligned} min(d_k(g))^T &= min(d_k(q)) - max(d_k(g)) \\ max(d_k(g))^T &= min(d_k(q)) - min(d_k(g)) \end{aligned} \tag{5.6}$$

Definition 5.1.1 *(Dynamic Area Skyline Query). For two objects, g and g′, we say g dynamically dominates g′ w.r.t q, if and only if* $max(d_k(g))^T \leq min(d_k(g'))^T$ *for all k* $(1 \leq k \leq m)$*. Dynamic area skyline query of q retrieves the set of all area objects that are not dynamically dominated by any other objects w.r.t q.*

Based on dynamic area skyline definition, we can formally define the reverse area skyline of query area q.

Definition 5.1.2 *(Reverse Area Skyline Query). Let G be a set of d-dimensional objects. Reverse area skyline query w.r.t query area q retrieves all area objects $g \in G$ where q is in the dynamic area skyline of g. In other words, we say g is reverse area skyline of q if $\nexists g' \in G$ such that $max(d_k(g'))^T \leq min(d_k(q))^T$ for all k ($1 \leq k \leq m$) w.r.t g.*

Using reverse area skyline query definition, we can compute reverse area skyline query by performing dynamic area skyline query for each grid object, and retrieving set of grid objects which have query area q in their dynamic area skyline result. But as discussed in Section 2.3, to compute reverse skyline by computing dynamic skyline for each object is time-consuming. In this chapter, we define global area skyline concept to compute reverse area skyline. We extend global skyline concept in [13] so that it can be applied in area skyline.

5.1.3 Disjoint, Overlap, Within/Contain

Using information of min and max distances, one object's min and max distance might disjoint, overlap, within/contain with another object's min and max distance. Let us consider the example of Figure 5.5 again. We define disjoint objects using case 1 and 6, overlap objects using case 2 and 5, and case 3 and 4 for within/contain objects.

Definition 5.1.3 *(Disjoint Objects). Object g disjoints with g′, if $max(d_k(g)) \leq min(d_k(g'))$ or $min(d_k(g)) \geq max(d_k(g'))$, for all $k \in m$.*

Definition 5.1.4 *(Overlap Objects). Object g overlaps with g′, if $max(d_k(g)) > max(d_k(g'))$ and $max(d_k(g')) > min(d_k(g)) \geq min(d_k(g'))$, or if $min(d_k(g)) < min(d_k(g'))$ and $min(d_k(g')) <$*

$max(d_k(g)) \leq max(d_k(g'))$, for at least one $k \in m$.

Definition 5.1.5 *(Within/Contain Objects). Object g is within g′, if $min(d_k(g)) \geq min(d_k(g'))$ and $max(d_k(g)) \leq max(d_k(g'))$, for all $k \in m$. Object g contains g′ if $min(d_k(g)) < min(d_k(g'))$ and $max(d_k(g)) > max(d_k(g'))$, for all $k \in m$.*

These disjoint, overlap, and within/contain conditions are two-dimensional objects' characteristics that are not exist in zero dimensional objects. Based on these characteristics, we define Lemma 1 and 2 which are very important to efficiently compute reverse area skyline using global area skyline concept.

Lemma 1. *Let q be the query area. If g overlaps with or within/contain q, then g must be a reverse area skyline of q.*

Proof. Assume g is not a reverse area skyline of q. Then, there should be at least one object that dynamically dominates q w.r.t g. If we apply dynamic area skyline of g, since g overlaps or within/contains q, based on case 2, 3, 4, 5 in Section 5.1.2, $min(d_k(g))^T$ is always be 0, which makes it not possible to be dominated by other objects. It means that q is a dynamic area skyline of g, and consequently, g is reverse area skyline of q. So the assumption is not true and the proof is complete.$\diamond$

Figure 5.6 shows an illustration of Lemma 1. Figure 5.6 (a) shows original min-max distance of q and g, while Figure 5.6 (b) shows min-max distance of q^T after we apply dynamic area skyline of g.

Lemma 1 provides an easy selection method for RASky algorithm to directly put overlap/within/contain objects into reverse area skyline result.

Before defining global area skyline, let us consider the definition of global skyline for zero dimensional data in [13]. Given a d-dimensional data set P and a query point q, p_1 is said globally dominates p_2 with respect to q if: (1) $(p_1\text{-}q)(p_2\text{-}q)>0$ for all di-

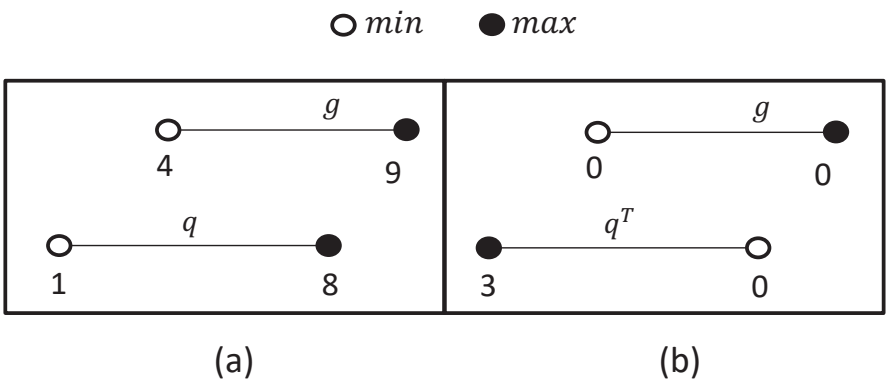


Figure 5.6: Lemma 1 situation

mensions, and (2) distance p_1 to q are smaller or equal then distance p_2 to q for all dimensions, and smaller at least in one dimension. Let us consider point h_2 and h_1 in Figure 2.10. We said h_2 is globally dominates h_1, because $(h_1\text{-}q)(h_2\text{-}q)>0$ and distance h_2 to q is smaller than distance h_1 to q for all dimensions. In two-dimensional case, the above situations become more complicated for overlap/within/contain conditions. Lemma 2 shows that overlap/within/contain objects can not globally dominate any other objects.

Lemma 2. *Overlap/within/contain objects can not globally dominate any other objects*

Proof. Let assume that g and g' is in the same quadrant w.r.t q. g is an overlap/within/contain object w.r.t q, and g' has $min(d_k(g'))^T \geq max(d_k(g))^T$, for all $k \in m$, so that g globally dominated g', and g' is not a reverse area skyline of q. On the contrary, if we apply dynamic area skyline of g', we can see that g can not dominate q, since g and q are overlap/within/contain objects. This mean that g' is reverse skyline of q and should not be discarded by g.$\diamond$

Figure 5.7 shows an illustration of Lemma 2. Figure 5.7 (a) shows the original min-max distance of q, g, and g'. Figure 5.7 (b) shows if g (overlap/within/contain object)

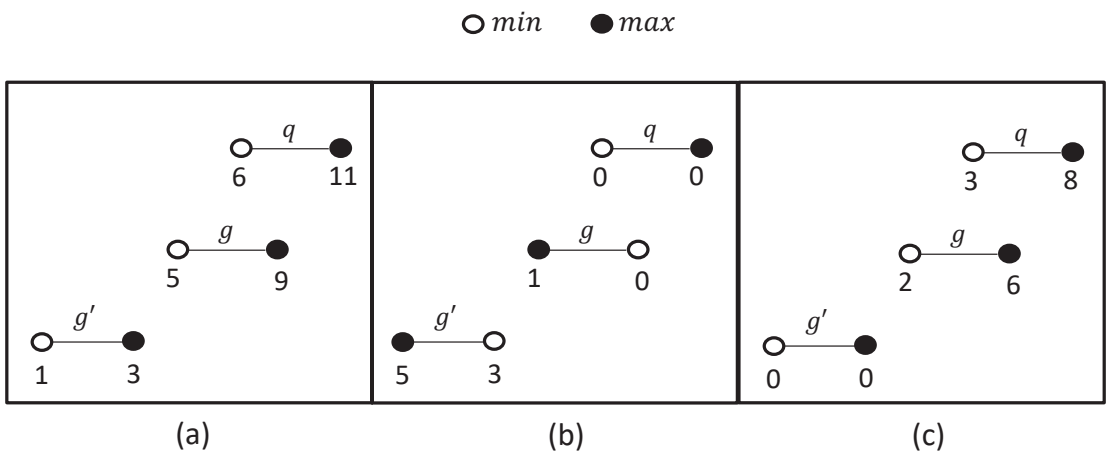


Figure 5.7: Lemma 2 situation

globally dominate g' w.r.t q, since $max(d_k(g))^T$ is smaller than $min(d_k(g'))^T$, then g' is not a reverse skyline of q. But this will lead to wrong result. If we apply dynamic area skyline on g' as in Figure 5.7 (c), q is in dynamic area skyline of g', which consequently makes g' as reverse area skyline of q. In this situation g should not be allowed to globally dominate g' in the first place, since g' is reverse area skyline of q. Using Lemma 1 and 2, we can reduce the comparison step in calculating global area skyline because all the overlap/within/contain objects do not participate in the comparison process.

5.1.4 Global and Global-1 Area Skyline

Based on Lemma 1 and 2, only disjoint objects will participate in global area skyline computation. Let us consider disjoint situations in Figure 5.5 case 1 and 6. In Figure 5.5 case 1, $min(d_k(g))$ and $max(d_k(g))$ are larger than $max(d_k(q))$, while in Figure 5.5 case 6 $min(d_k(g))$ and $max(d_k(g))$ are smaller than $min(d_k(q))$. To differentiate between these two disjoint conditions, we define $diff(g_k)$ as:

$$diff(g_k) = \begin{cases} min(d_k(g)) - max(d_k(q)) \\ \qquad\qquad \text{if } max(d_k(q)) \leq min(d_k(g)). \\ max(d_k(g)) - min(d_k(q)) \\ \qquad\qquad \text{if } max(d_k(g)) \leq min(d_k(q)). \end{cases}$$

Notice that the value of $diff(g_k)$ could be "positive" (case 1) or "negative" (case 6). Two objects g and g' are in the same quadrant w.r.t q if $(diff(g_k))(diff(g'_k))$ >0 for all $k \in m$. In Figure 5.8, since $max(d_k(g)) \leq min(d_k(q))$,(15 ≤ 20), then $diff(g_k) < 0$ while $diff(g'_k) > 0$, since $min(d_k(g')) \geq max(d_k(q))$, (30 ≥ 25). Using Lemma 1, Lemma 2, and *diff* definition, we define global and global-1 area skyline.

Definition 5.1.6 *(Global and Global-1 Area Skyline). For two objects, g and g′, we say g globally dominates g′ w.r.t q, if and only if: (1)g and g′ are disjoint objects w.r.t q, (2)*$(diff(g_k))(diff(g'_k)) > 0$ *and (3)* $max(d_k(g))^T \leq min(d_k(g'))^T$*, for all* $k \in m$*. Any objects g″ becomes global-1 area skyline if there is only one other object that globally dominates it.*

5.1.5 Window Query

Window Query of grid $w(g)$ w.r.t q, has minimum and maximum value for each k dimension, $min(w_k(g))$ and $max(w_k(g))$ where $k \in m$. It is defined as follows:

$$min(w_k(g)) = \begin{cases} max(d_k(q)) \qquad \text{if } min(d_k(g)) \geq max(d_k(q)). \\ min(d_k(g)) + diff(g_k) \\ \qquad\qquad \text{if } max(d_k(g)) \leq min(d_k(q)). \end{cases}$$

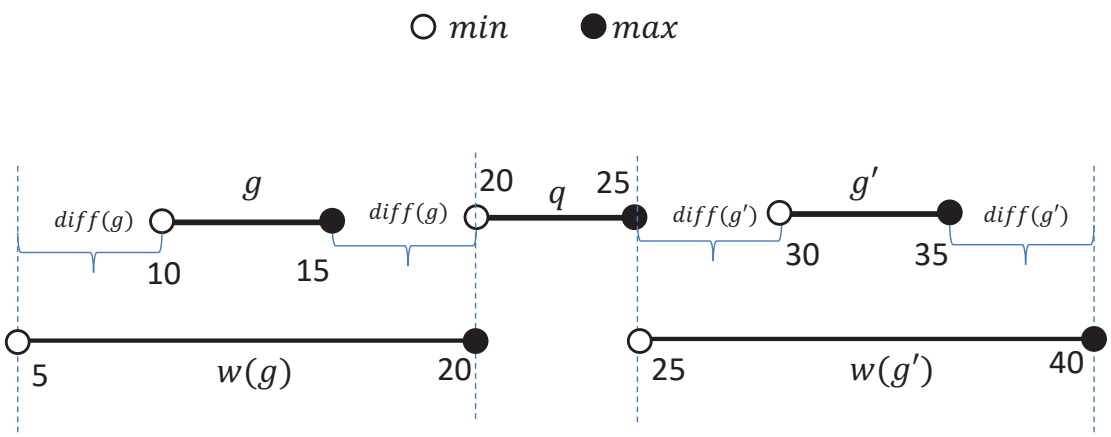


Figure 5.8: Diff and Window query

$$max(w_k(g)) = \begin{cases} max(d_k(g)) + diff(g_k) \\ \qquad\qquad \text{if } min(d_k(g)) \geq max(d_k(q)). \\ min(d_k(q)) \qquad \text{if } max(d_k(g)) \leq min(d_k(q)). \end{cases}$$

Figure 5.8 shows an illustration of window query's minimum and maximum value in one dimension. Assume min and max distance for g, q, and g' are (10,15), (20,25), and (30,35). For $w(g)$, since $max(d_k(g)) \leq min(d_k(q))$, then $diff(g_k)$ is -5 (15-20), so that $min(w_k(g))$ and $max(w_k(g))$ are 5 (10 + (-5)) and 20 (same value as $min(d_k(q))$). For $w(g')$, since $min(d_k(g)) \geq max(d_k(q))$, then $diff(g'_k)$ is 5 (30-25), so that $min(w_k(g'))$ and $max(w_k(g'))$ are 25 (same value as $max(d_k(q))$) and 40 (35+5).

Lemma 3. *Let g be a global area skyline of q, and g′ be a global or global 1-area skyline of q with the same quadrant with g. If the window query of g contains g′ w.r.t q, then g is not a reverse area skyline of q.*

Proof. If the window query of g contains g', then if we apply dynamic area skyline

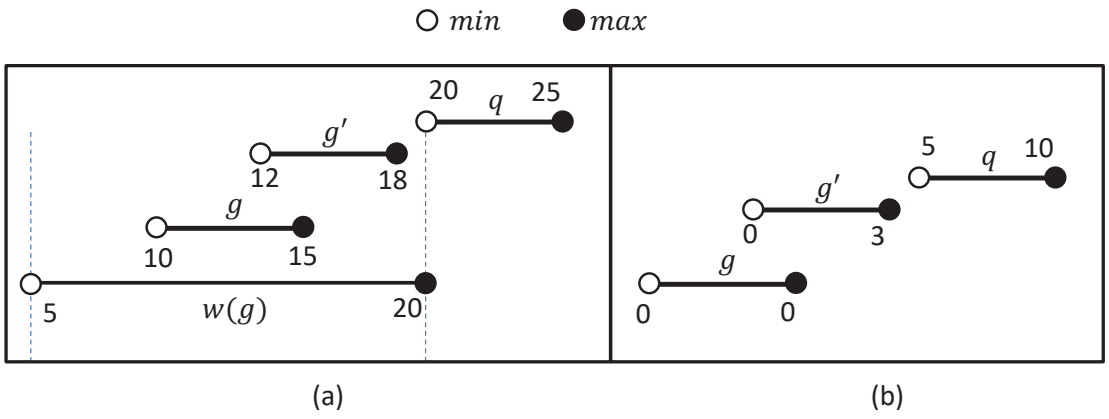


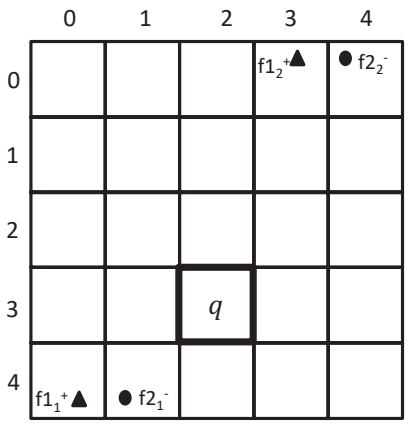
Figure 5.9: Lemma 3 situation

of g using formula in Section 5.1.2, we know that $max(d_k(g'))^T$ is always smaller than $min(d_k(q))^T$. It means that g' will dynamically dominate q w.r.t g, therefore g can not be a reverse area skyline of q.$\diamond$

Figure 5.9 illustrates Lemma 3 situation. Figure 5.9 (a) shows that $w(g)$ is contain g', while Figure 5.9 (b) shows that g' will dynamically dominate q w.r.t g, so that g is not a reverse area skyline of q. Using Lemma 3, for each global area skyline we simply just check whether at least one of other global or global 1-area skyline is within its window query or not.

5.2 Reverse Area Skyline (RASky) Algorithm

Reverse area skyline algorithm (RASky) consist of two steps. At step 1, we divide A into grids. For each grid, we find the nearest facility type, calculate its min and max distance, and complete the distance information in Minmax table using the same method in GASky step 1 [3] as explained in Section 4.2.2. In step 2, using information in Minmax table from the first step, we calculate reverse area skyline using global area



(a)

Grid ID	F1⁺ min	F1⁺ max	F2⁻ min	F2⁻ max
0,0	27	38	33	43
0,1	17	28	23	34
0,2	8	18	8	19
0,3	0	10	3	15
0,4	2	14	0	10
1,0	29	39	28	40
1,1	19	32	24	37
1,2	10	25	15	28
1,3	7	19	8	22
1,4	8	21	8	19
2,0	18	29	18	31
2,1	19	30	18	29
2,2	19	32	19	32
2,3	17	28	18	30
2,4	18	30	18	28
3,0	8	19	8	22
3,1	9	23	8	19
3,2	**15**	**29**	**10**	**25**
3,3	25	37	19	32
3,4	27	39	28	38
4,0	0	10	3	15
4,1	2	14	0	10
4,2	13	24	7	18
4,3	23	34	17	28
4,4	33	44	28	38

(b)

Figure 5.10: Sample map (a) and Minmax Table (b)

skyline. In this section, we will focus on the reverse area skyline step 2.

In this section we use sample map in Figure 5.10 (a) and set $g_{3,2}$ as query area q, then divide sample map into 5x5 grids. In this map we have two types of facilities, $F1^+$ and $F2^-$, each of them has two objects $F1^+ = (f1_1^+, f1_2^+)$, $F2^- = (f2_1^-, f2_2^-)$. After completing RASky step 1 in the sample map, we obtain Minmax table like in Figure 5.10 (b).

We index the grid by their min and max distance in Minmax table using R-tree structure. Each leaf in the R-tree is in the format $(id, qd, RECT)$, where id is the number of grid in Minmax table, qd is quadrant, and $RECT$ is a bundle of all min and max distances in a grid for all dimensions. For example, for 2 facility type, or 2-dimensional, $RECT$ has $(f1min, f2min)$ as bottom-left coordinate and $(f1max, f2max)$ as top-right coordinate. Our query object $RECT_{3,2}$ has bottom-left coordinate (15,10) and top-right coordinate (29,25). Using $RECT$ object, we build R-tree of Minmax table.

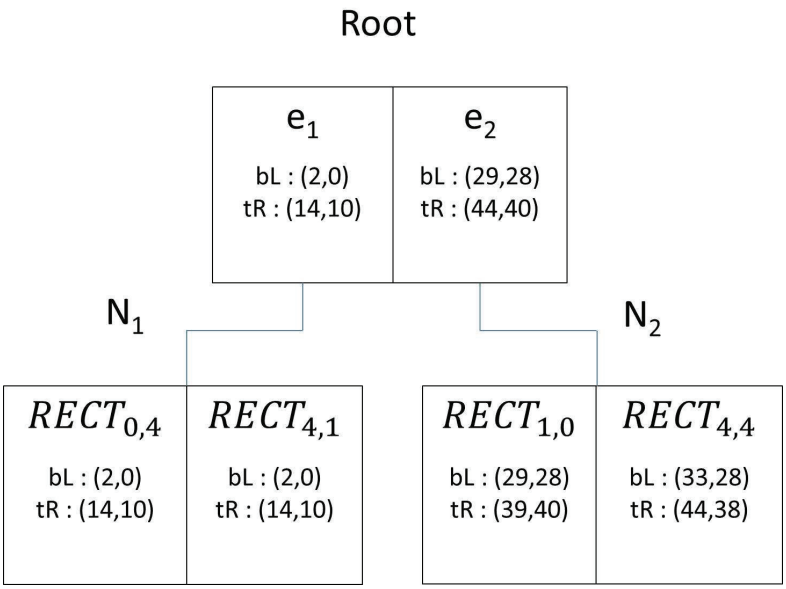


Figure 5.11: R-tree of disjoint objects

5.2.1 Building R-tree

RASky reads each object in Minmax table. Using Lemma 1 and 2, if the object is an overlap/within/contain object, then it will automatically be a reverse area skyline object, and will be excluded from R-tree and further computation. Only disjoint objects will be inserted into R-tree. Let us consider Minmax table in Figure 5.10 (b). Since $g_{0,4}$, $g_{1,0}$, $g_{4,1}$, and $g_{4,4}$ are disjoint objects (rows with bold border in Figure 5.10 (b)), they are inserted into R-tree, while others directly become reverse area skyline of q. Figure 5.11 shows R-tree after inserting disjoint objects.

5.2.2 Finding Global and Global-1 area skyline

RASky inserts all root entries into heap H and sort them by their distance from q. Besides H, we also use two additional heaps, H_g and H_{g1}, to maintain global area skyline and global-1 area skyline. Since $N1$ is closest to q, its entry is expanded, and $N1$ is

Window Query	$diff(g_1)$	$diff(g_2)$	$\min(w_1(g)), max(w_1(g))$	$\min(w_2(g)), max(w_2(g))$	bL	tR
0,4	-1	0	(1,15)	(0,10)	(1,0)	(15,10)
4,1	-1	0	(1,15)	(0,10)	(1,0)	(15,10)
1,0	0	3	(29,39)	(25,43)	(29,25)	(39,43)
4,4	4	3	(29,48)	(25,41)	(29,25)	(48,41)

Figure 5.12: Window query in sample dataset

removed from H. Now H contents become $RECT_{0,4}$, $RECT_{4,1}$, and $N2$. As top of H, $RECT_{0,4}$ then becomes the first global area skyline and is inserted into H_g. Notice that $RECT_{4,1}$ is in the same quadrant with $RECT_{0,4}$ and it is not globally dominated by $RECT_{0,4}$, so it is also inserted into H_g. Next $N2$ is expanded and $RECT_{1,0}$ and $RECT_{4,4}$ are inserted into H. Since $RECT_{1,0}$ is in different quadrant with $RECT_{0,4}$ and $RECT_{4,1}$, $RECT_{1,0}$ also becomes global area skyline and is inserted into H_g. $RECT_{4,4}$ is in the same quadrant with $RECT_{1,0}$, but since it is not globally dominated by $RECT_{1,0}$, then it is also inserted into H_g. Since there is no global-1 area skyline in this sample dataset, then H_{g1} is remain empty.

5.2.3 Applying Window Query

After getting all global area skylines, we build window query for each entry in H_g. Using window query formula in Section 5.1.5, Figure 5.12 shows bottom-left and top-right coordinate of each window query for query area $RECT_{3,2}$ whose bottom-left and top-right is (15,10) and (29,25), respectively.

Let us consider $w(g_{0,4})$, $diff(g_{0,4_1})$ is -1 (14-15) and $diff(g_{0,4_2})$ is 0 (10-10). Using

these values, we can compute min and max of $w(g_{0,4})$ in dimension 1 as (2+(-1),15) and in dimension 2 as (0+0,10), so that bL and tR coordinates are (1,0) and (15,10). Since $RECT_{4,1}$ has the same bL and tR coordinate with $RECT_{0,4}$, then min and max of $w(g_{4,1})$ are the same with $w(g_{0,4})$. This mean that $RECT_{4,1}$ always contains $w(g_{0,4})$ and vice versa, so based on Lemma 3, both of them are not reverse area skyline of $g_{3,2}$. Now for $w(g_{1,0})$, $diff(g_{1,0_1})$ is 0 (29-29) and $diff(g_{1,0_2})$ is 3 (28-25). Min and max of $w(g_{1,0})$ in dimension 1 as (29,39+0) and in dimension 2 as (25,40+3), so that bL and tR coordinates are (29,25) and (39,43). Finally, for $w(g_{4,4})$, $diff(g_{4,4_1})$ is 4 (33-29) and $diff(g_{4,4_2})$ is 3 (28-25). Min and max of $w(g_{4,4})$ in dimension 1 as (29,44+4) and in dimension 2 as (25,38+3), so that bottom-left and top-right coordinates are (29,25) and (48,41). $RECT_{4,4}$ overlaps with $w(g_{1,0})$, while window query of $w(g_{4,4})$ contains $RECT_{1,0}$. Based on Lemma 3, $RECT_{1,0}$ is reverse area skyline of q while $RECT_{4,4}$ is not. From the above computation, we can find that $g_{0,4}$, $g_{4,1}$, and $g_{4,4}$ are not reverse area skyline of $g_{3,2}$ while the others are.

Shaded area in Figure 5.13 shows reverse area skyline of $g_{3,2}$ in sample map Figure 5.10 (a), which is 88% of all grids. Next in the experiment section, we discover that smaller size of query area q will reduce reverse area skyline result.

5.3 Experimental Evaluation

We experimentally evaluated RASky algorithm in a PC with Intel Core i5 3.2GHz processor and 4GB of RAM. We conducted three experiments using three synthetic datasets. In each experiment, we repeated five times and reported the average. We examined the effect of parameters such as number of objects, number of types, and number of grids, to the step 1 and step 2 of RASky algorithm. We recorded the processing time

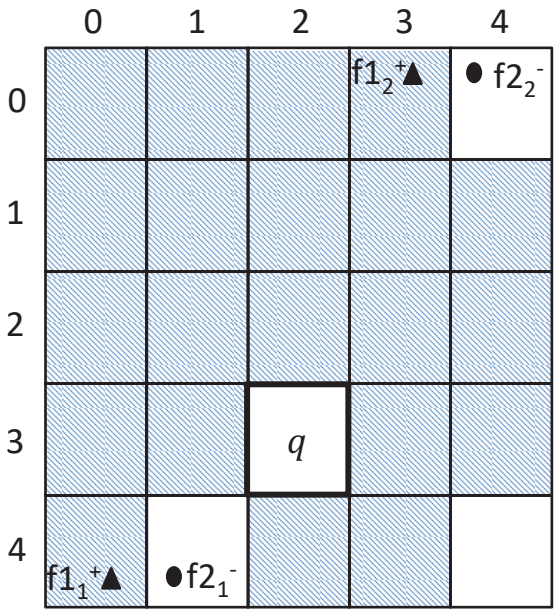


Figure 5.13: Reverse area skyline for sample map

for step 1 and step 2 of RASky and the ratio of reverse area skyline resulted from each experiment. Ratio of skyline is the number of reverse area skyline compared with the number of grids in the experiment, Table 5.1 lists the synthetic datasets and parameters in these experiments.

Table 5.1: Experimental Dataset

Dataset	**Objects**	**Types**	**Grids**
DB1	1k,2k,4k,8k,16k	2	160k
DB2	1k	2,4,8,16	40k
DB3	1k	2	10k,40k,160k,640k

5.3.1 Effect of Number of Objects

In these experiments, we examined the performance of RASky on the different number of objects, when the number of facility types and the number of grids are fix, using DB1. Figure 5.14 shows the processing time of this experiment. We can see that the increase of the number of objects will increase the total processing time of RASky. In step 1, increasing number of objects will increase the processing time to build Voronoi diagram. However, since the number of Voronoi diagrams is fix according to the number of types, increasing number of objects does not have effect in the size of Minmax table. Hence in RASky step 2, increasing the number of objects has less effect, and the processing time tends to decrease when the number of objects increases. The reason is because increasing number of objects, while the number of grids is fix, increases the number of non-disjoint objects. It means less objects will participate in global area skyline computation, since only disjoint objects participates in the computation. Therefore the processing time will be decreased. The ratio of reverse area skyline is increasing as reported in Figure 5.15. Increasing the number of objects will cause smaller value on min and max distances, but since the number of grids is fix when the number of objects increase, the ratio of reverse area skyline still will increase.

5.3.2 Effect of Number of Types

In these experiments, we used a synthetic data DB2 that have fix number of objects and number of grids. From the results in Figure 5.16, we can observe that the processing time increases with the increase of the number of types. The increasing number of types will require more Voronoi diagrams, which in turn increase the processing time. The result illustrates that increasing the number of types significantly increase the process-

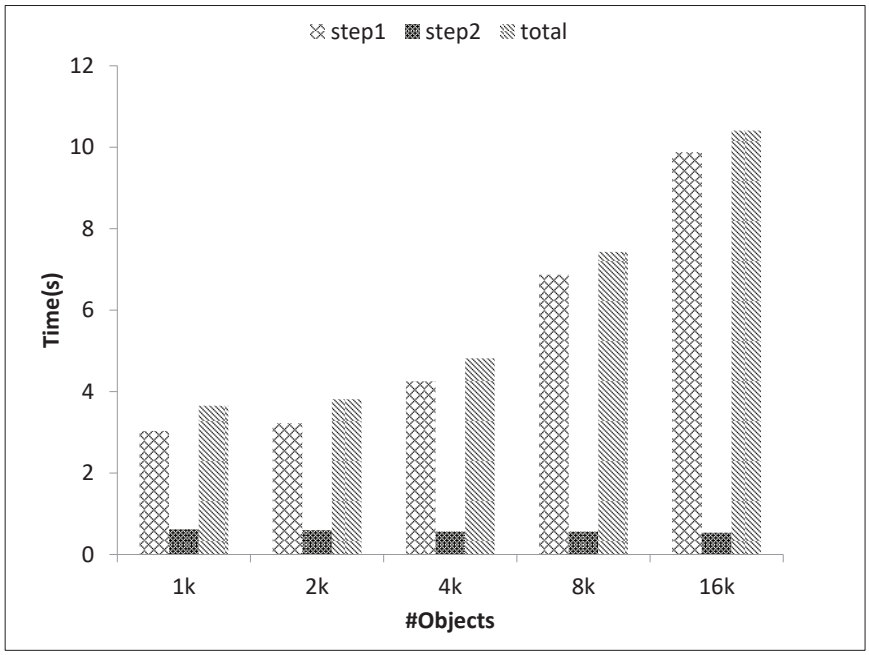


Figure 5.14: Processing time of *DB*1

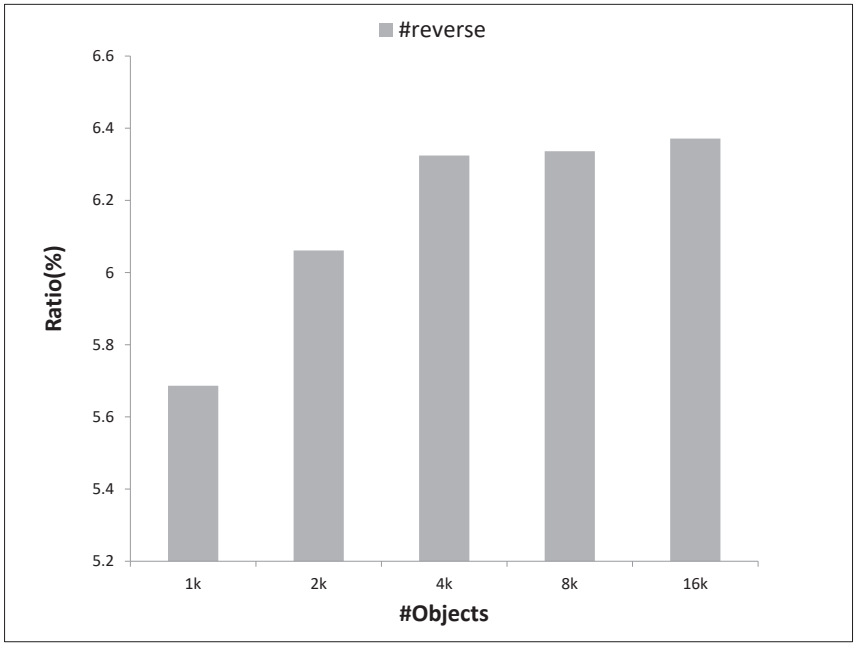


Figure 5.15: Reverse area skyline's ratio of *DB*1

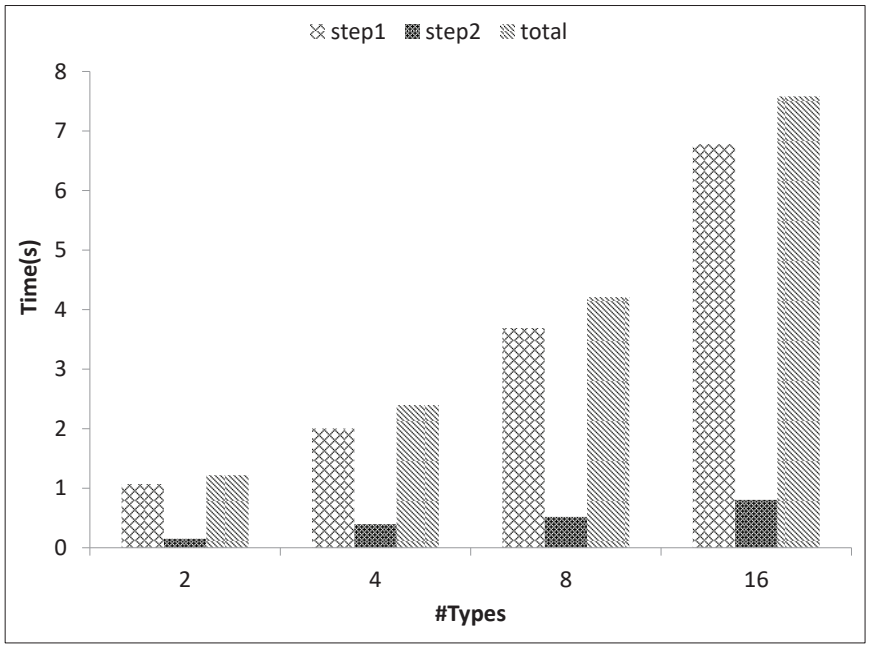


Figure 5.16: Processing time of *DB*2

ing time of step 1. Similar with increasing number of objects, increasing number of types with fixed number of grids will decrease the number of disjoint objects. Although the number of disjoint objects decreases, the processing time still increases because increasing facility types also means larger size of Minmax tables. Since the dimension is increasing as the number of facility types increase, the ratio of skyline is also increasing as shown in Figure 5.17.

5.3.3 Effect of Number of Grids

In these experiments, we evaluated the effect of number of grids while the number of objects and number of types are fix, using DB3. Figure 5.18 shows that the number of grids affects the processing time of step 1 and step 2. In step 1, increasing the number of grids means more comparison on Voronoi diagrams and more calculation time

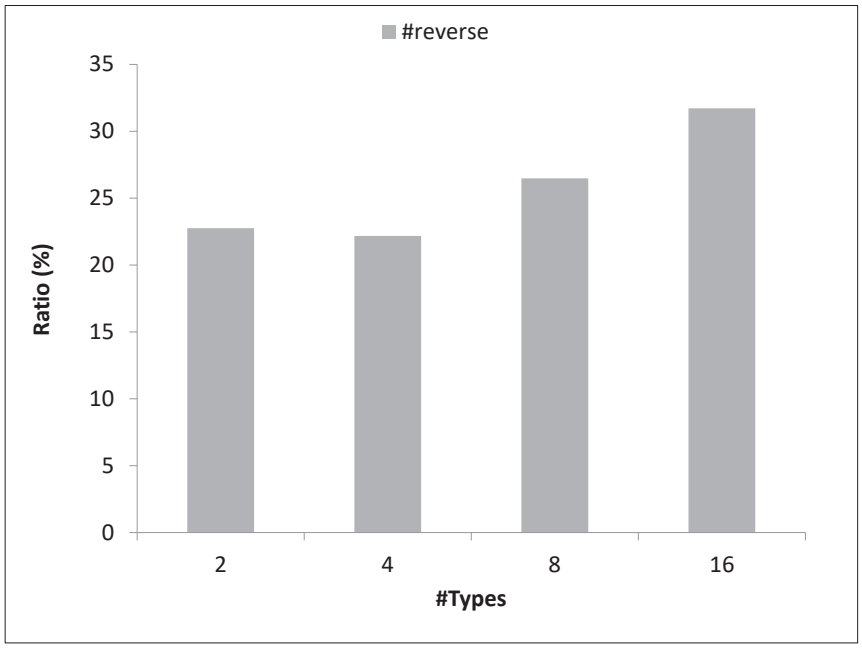


Figure 5.17: Reverse area skyline's ratio of *DB*2

to obtain min and max distance, and in the same time enlarges the number of record in Minmax table which also cause increasing the time needed for step 2 computation. Increasing number of grids while number of objects and number of types are fix also causing the number of disjoint objects to increase. In step 2, increasing number of disjoint objects will increase processing time since more objects will participate in global area skyline computation. In Figure 5.19, the effect of number of grids affects ratio of skyline differently compared to the effect of the number of objects and types. Increasing the number of grids will decrease the ratio of skyline. The important reason of that is because more grids has the same meaning of having smaller size of each grid, which significantly decrease the ratio of skyline, since smaller area is likely to be dominated by another area.

From all of experimental results, we can indicate that the total processing time of RASky increases when the number of objects, number of facility types, and the number

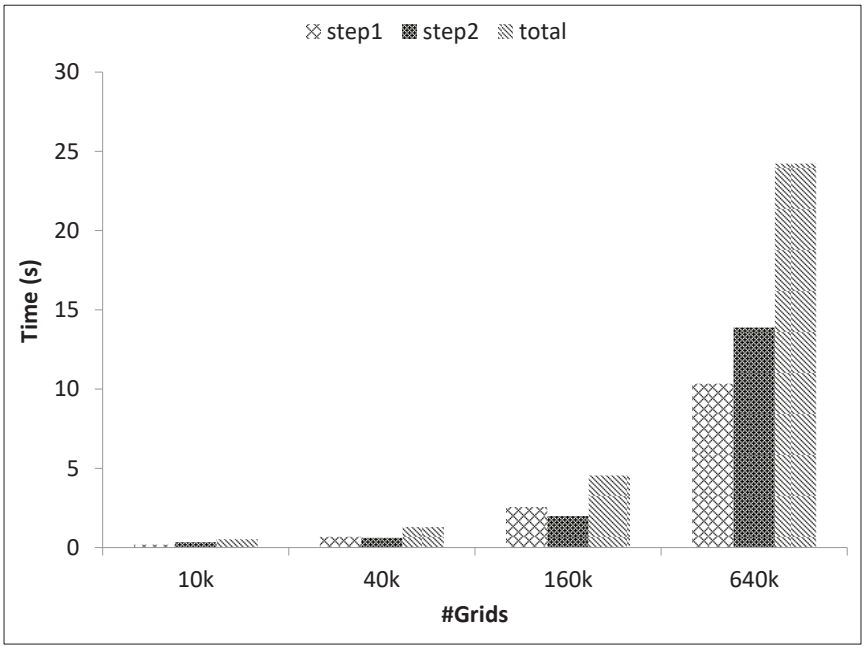


Figure 5.18: Processing time of *DB*3

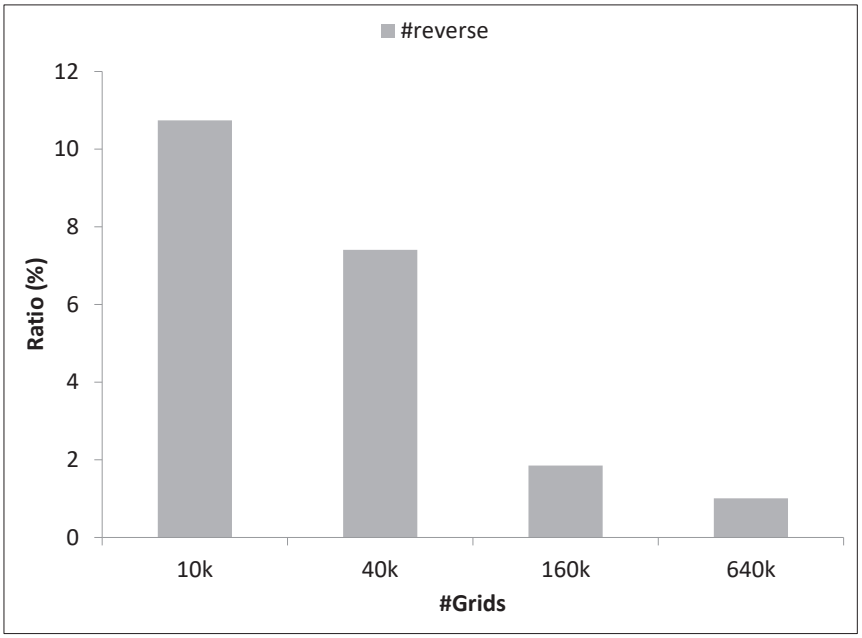


Figure 5.19: Reverse area skyline's ratio of *DB*3

of grids increases. In addition, the ratio of skyline increases when the number of objects and types increases, and decreases when the number of grids increases.

5.4 Concluding Remarks

In this chapter, we defined dynamic area skyline, global area skyline, and proposed reverse area skyline algorithm (RASky) to solve the reverse area skyline query. This query is very important for location selection from business' or landowners' perspective. RASky has two steps, step 1 to compute Minmax table and step 2 to calculate reverse area skyline. Smaller query area will obtain smaller number of reverse area skylines and vice versa. Reverse area skyline gives invaluable information for landowner to pursue targeted customer or to decide what type of business that would attract more customers. Comprehensive experiments are conducted to show the effectiveness and efficiency of the proposed algorithm. In the future, we will consider another skyline problem in two-dimensional objects, such as selecting k-dominant areas. We are also interested in the application of this method to road network, which also taken into account non-spatial properties such as population density, price, traffic condition, and so on.

Chapter 6

Conclusion

This thesis aims to answer location selection problem for two-dimensional area objects using skyline query. Specifically, this thesis covered two different location selection problem for area, one is based on user/customer's perspective and another is based on owner's perspective. We proposed novel algorithms and showed the efficiency of our approaches through extensive experiments.

The remaining of this chapter is organized as follows. Section 6.1 explores the application of our developed algorithms. In Section 6.2, we continue with the contributions of this dissertation. Finally, in Section 6.3 we present some future works which correspond to our works.

6.1 The Application of Location Selection Problem in Areas

Location selection is very important in the services and business field, either for user/customer or for location's owner. User/customer of location utilizes location selection query to

find good location to buy, to rent, or to visit. From user/customer's perspective, the location is more valuable if it is close to some facilities that can bring profit and benefits for the location, and it is far from some facilities that can reduce profit and bring unfavorable effect on the location. On the other hand, location selection for location's owner aims to find other locations which are as good as his/her owned locations. One of the important criteria for comparing his/her location to others is the location's distance to desirable and undesirable facilities. Skyline query is a well known method for selecting small number of data objects, and it also has been applied in location selection problem. However, previous skyline algorithm for location selection problem only consider zero dimensional objects and based on the assumption that there are always some candidate points to be selected. In this thesis, first we introduced area skyline queries to answer user/customer's need of location selection on two-dimensional objects. For example, in the business field: suppose a property company would like to build a new housing complex in a new region. To attract customers, the housing complex should be in an area that is close to train stations, shopping centers, and schools, and far from open landfill. Our area skyline algorithms can help the company finding potential area for its new project and gain knowledge about the region and its facilities, thus reducing cost of surveying the whole region. Moreover, consider a tourist start planning trip to a new area or country. Sometimes a traveler would like to stay in an area that will be convenient in location and cost. The proposed area skyline methods can help a tourist to know which area is close to attraction sites, train stations, and convenience stores and is far from crime areas and polluted areas. After that the tourist can search for place of stay in the skyline areas. Next part of this thesis considers location selection problem in two-dimensional area based on owner's perspective which we called as reverse area

skyline query. For example a real estate company has an area to develop apartment, office, or market complex. The company needs to know, which other areas are as good as his/her area. Our proposed algorithm help owner to obtain reverse skyline area from the query area. Using reverse skyline area, owner can make further analysis about who will be interested in the query area, since customer who are attracted with reverse area skyline might also be attracted with the query area. Moreover, reverse area skyline also helps business/land owner to learn more about the development on those area, so he/she can decide effective real estate developments that can attract many buyers on query area.

6.2 Contributions

6.2.1 Area Skyline Query: Skyline query for selecting spatial area objects from customers perspectives

In Chapter 3 and 4, we introduced area skyline queries and developed two algorithms, UASky and GASky to answer this problem. UASky is not so efficient since it selects many area as area skyline. We then presented GASky, to solve the limitation of UASky and improve its efficiency. Using grid data structure, GASky is efficient and practical solution of the area skyline query. By applying grid data structure, GASky can control the number of area skyline so it retrieves reasonable number of area skylines. We have conducted intensive experiments to prove the efficiency of our algorithm.

6.2.2 Reverse Area Skyline Query: Skyline query for selecting spatial area objects from owners perspectives

In Chapter 5, we defined dynamic area skyline, global area skyline, and proposed reverse area skyline algorithm (RASky) to solve the reverse area skyline query. RASky utilizes the step 1 of GASky algorithm to compute Minmax table. Reverse area skyline gives invaluable information for area's owner to pursue targeted customer or to decide what type of business that would attract more customers. Comprehensive experiments are conducted to show the effectiveness and efficiency of the proposed algorithms.

6.3 Future Works

Our current work uses Euclidean distance between area and facilities. In real life, areas on a map also correspond to road network which is more complicated than Euclidean distance. Moreover, road network is also taking into account many aspects such as the direction of a road. In the future we will consider area skyline and reverse area skyline using the real distance on the road network. Moreover, area in a map also has non-spatial attributes such as population density, price, traffic condition, and so on. Exploring area skyline and reverse area skyline which take into account non-spatial properties of the area is also part of our future work. The challenging related open problems such as considering more than one object for each facility type and selecting k-dominant areas would be very important to increase the selectivity of the area skyline query and reverse area skyline query.

References

[1] F.N. Afrati, P. Koutris, D. Suciu, and J.D. Ullman. ”Parallel skyline queries”. In *Theory of Computing Systems*, volume 57(4), pages 1008–1037, 2015.

[2] Annisa, Md. A. Siddique, A. Zaman, and Y. Morimoto. ”A method for selecting desirable unfixed shape areas from integrated geographic information system”. In *Proceedings of IIAI*, pages 195–200, 2015.

[3] Annisa, A. Zaman, and Y. Morimoto. ”Area skyline query for selecting good locations in a map”. In *Information Processing Society of Japan : Database Transaction*, volume 9(3), pages 946–955, 2016.

[4] M. Arefin, G. Ma, and Y. Morimoto. ”A spatial skyline query for a group of users”. In *Journal of Software*, volume 9(11), pages 2938–2947, 2014.

[5] M. Arefin, J. Xu, Z. Chen, and Y. Morimoto. ”Skyline query for selecting spatial objects by utilizing surrounding objects”. In *Journal of Software*, volume 8(7), pages 1742–1749, 2013.

[6] S. Borzsonyi, D. Kossmann, and K. Stocker. ”The skyline operator”. In *Proceedings of the 17th International Conference on Data Engineering (ICDE), April 2–6, Heidelberg, Germany*, pages 421–430, 2001.

[7] C. Y. Chan, H. V. Jagadish, K-L. Tan, A. K. H. Tung, and Z. Zhang. ”Finding k-Dominant skyline in high dimensional space”. In *Proceedings of ACM SIGMOD*, pages 503–514, 2006.

[8] L. Chen and X. Lian. ”Dynamic skyline queries in metric spaces”. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology March*, pages 333–343, 2008.

[9] H. Choi, H. Jung, K.Y. Lee, and Y.D. Chung. ”Skyline queries on keyword-matched data”. In *Information Sciences*, volume 232, pages 449–463, 2013.

[10] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. ”Skyline with presorting”. In *Proceedings of the 19th International Conference on Data Engineering (ICDE), March 5–8, Bangalore, India*, pages 717–719, 2003.

[11] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. ”Computational Geometry, Algorithms and Applications”. In *Springer Berlin Heidelberg, Germany*, pages 145–160, 1997.

[12] H. Edelsbrunner. ”Algorithms in Combinatorial Geometry”. In *Springer Science & Business Media*, volume 10, pages 293–332, 2012.

[13] D. Evangelos and B. Seeger. ”Efficient computation of reverse skyline queries”. In *Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment*, pages 291–302, 2007.

[14] R. A. Finkel and J. L. Bentley. ”Quad trees a data structure for retrieval on composite keys”. In *Acta informatica*, volume 4(1), pages 1–9, 1974.

[15] Y. Gao, Q. Liu, B. Zheng, and G. Chen. ”On efficient reverse skyline query processing”. In *Expert Systems with Applications*, volume 41(7), pages 3237–3249, 2014.

[16] X. Guo, Y. Ishikawa, and Y. Gao. ”Direction-based spatial skylines”. In *Proceedings of the 9th ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE), June 6, Indiana, USA*, pages 73–80, 2010.

[17] Z. Huang, H. Lu, B.C. Ooi, and A.K. Tung. ”Continuous skyline queries for moving objects”. In *IEEE Transactions on Knowledge and Data Engineering*, volume 18(2), pages 1645–1658, 2006.

[18] Md. I. Islam, R. Zhou, and C. Liu. ”On answering why-not questions in reverse skyline queries”. In *IEEE 29th International Conference on Data Engineering*, pages 973–984, 2013.

[19] K. Kodama, Y. Iijima, X. Guo, and Y. Ishikawa. ”Skyline queries based on user locations and preferences for making location-based recommendations”. In *Proceedings of the International Workshop on Location Based Social Networks (LBSN) November 03, Washington, USA*, pages 9–16, 2009.

[20] H.P. Kriegel, M. Renz, and M. Schubert. ”Route skyline queries: A multi-preference path planning approach”. In *Proceedings of Data Engineering (ICDE), 2010 IEEE 26th International Conference on March*, pages 261–272, 2010.

[21] I. Lazaridis and S. Mehrotra. ”Progressive approximate aggregate queries with a multi-resolution tree structure”. In *ACM SIGMOD Record*, volume 30(2), pages 401–412, 2001.

[22] Q. Lin, Y. Zhang, W. Zhang, and X. Lin. ”Efficient general spatial skyline computation”. In *World Wide Web*, volume 16(3), pages 247–270, 2013.

[23] X. Lin, J. Xu, and H. Hu. ”Range-based skyline queries in mobile environments”. In *IEEE Transactions on Knowledge and Data Engineering*, volume 25(4), pages 835–849, 2013.

[24] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. ”Selecting stars: The k most representative skyline operator”. In *Proceedings of Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on April*, pages 86–95, 2007.

[25] Y. W. Lin, E. T. Wang, C. F. Chiang, and A. L. P. Chen. ”Finding targets with the nearest favor neighbor and farthest disfavor neighbor by a skyline query”. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC), March 24–28, Gyeongju, Korea*, pages 821–826, 2014.

[26] Q. Liu, Y. Gao, G. Chen, Q. Li, and T. Jiang. ”On efficient reverse k-skyband query processing”. In *International Conference on Database Systems for Advanced Applications*, pages 544–559, 2012.

[27] H. Lu and M.L. Yiu. ”On computing farthest dominated locations”. In *IEEE Transactions on Knowledge and Data Engineering*, volume 23(6), pages 928–941, 2011.

[28] M. Magnani, I. Assent, and M.L. Mortensen. ”Taking the Big Picture: representative skylines based on significance and diversity”. In *VLDB Journal*, volume 23(5), pages 795–815, 2014.

[29] K. Mullesgaard, J.L. Pedersen, H. Lu, and Y. Zhou. ”Efficient skyline computation in MapReduce”. In *Proceedings of 17th International Conference on Extending Database Technology (EDBT) March*, pages 37–48, 2014.

[30] T. Ohya, M. Iri, and K. Murota. ”A fast voronoi diagram algorithm with quaternary tree bucketing”. In *Information Processing Letters*, volume 18(4), pages 227–231, 1984.

[31] D. Papadias, Y. Tao, G. Fu, and B. Seeger. ”An optimal and progressive algorithm for skyline queries”. In *Proceedings of the ACM SIGMOD June 9–12, California, USA*, pages 467–478, 2003.

[32] Y. Park, J. Min, and K. Shim. ”Parallel computation of skyline and reverse skyline queries using mapreduce”. In *Proceedings of the VLDB Endowment 6, no. 14*, pages 2002–2013, 2013.

[33] N. Roussopoulos, S. Kelley, and F. Vincent. ”Nearest neighbor queries”. In *ACM sigmod record*, volume 24(2), pages 71–79, 1995.

[34] D. Sacharidis, P. Bouros, and T. Sellis. ”Caching dynamic skyline queries”. In *Proceedings of International Conference on Scientific and Statistical Database Management July 9*, pages 455–472, 2008.

[35] M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB), September 12–15, Seoul, Korea*, pages 751–762, 2006.

[36] M. Sharifzadeh and C. Shahabi. ”VoR-tree: R-trees with Voronoi Diagrams for Efficient Processing of Spatial Nearest Neighbor Queries”. In *Proceeding of VLDB Endowment*, volume 3(1-2), pages 1231–1242, 2010.

[37] J. Shi, D. Wu, and N. Mamoulis. ”Textually relevant spatial skylines”. In *IEEE Transactions on Knowledge and Data Engineering*, volume 28(1), pages 224–237, 2016.

[38] K. L. Tan, P.K. Eng, and B. C. Ooi. ”Efficient Progressive Skyline Computation”. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), September 11–14, Rome, Italy*, pages 301–310, 2001.

[39] Y. Tao, L. Ding, X. Lin, and J. Pei. ”Distance-based representative skyline”. In *Proceedings of Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on March*, pages 892–903, 2009.

[40] G. Valkanas, A.N. Papadopoulos, and D. Gunopulos. ”Skyline Ranking la IR”. In *Proceedings of EDBT/ICDT Workshops March*, pages 182–187, 2014.

[41] A. Vlachou and M. Vazirgiannis. ”Ranking the sky: Discovering the importance of skyline points through subspace dominance relationships”. In *Data and Knowledge Engineering*, volume 69(9), pages 943–964, 2010.

[42] G. Wang, J. Xin, L. Chen, and Y. Liu. ”Energy-efficient reverse skyline query processing over wireless sensor networks”. In *IEEE Transactions on Knowledge and Data Engineering*, volume 24(7), pages 1259–1275, 2012.

[43] W.C. Wang, E.T. Wang, and A.L. Chen. ”Dynamic skylines considering range queries”. In *Proceedings of International Conference on Database Systems for Advanced Applications April*, pages 235–250, 2011.

[44] R.C.W. Wong, A.W.C. Fu, J. Pei, Y.S. Ho, T. Wong, and Y. Liu. ”Efficient skyline querying with variable user preferences on nominal attributes”. In *Proceedings of the VLDB Endowment August*, pages 1032–1043, 2008.

[45] W. Xiaobing, Y. Tao, R. C. Wong, L. Ding, and J. X. Yu. ”Finding the influence set through skylines”. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, ACM*, pages 1030–1041, 2009.

[46] G. W. You, M. W. Lee, H. Im, and S. W. Hwang. ”The farthest spatial skyline queries”. In *Information Systems*, volume 38(3), pages 286–301, 2013.

[47] L. Zhu, C. Li, and H. Chen. ”Efficient computation of reverse skyline on data stream”. In *Computational Sciences and Optimization, International Joint Conference on, IEEE*, volume 1, pages 735–739, 2009.