Information Filtering of Very Large Databases by Using Skyline Queries
（スカイライン問合わせを利用した大規模データベースの情報選別）

Mohammad Shamsul Arefin

Department of Information Engineering
Graduate School of Engineering

Hiroshima University

Supervisor: Yasuhiko Morimoto, Assoc. Prof.

A dissertation submitted in partial fulfillment of the requirements for the Degree of

Doctor of Engineering

September 2013

To the last prophet,

Hazrat Muhammad Mustafa Sal Lal La Hu Aliahae Wa Sallam

whom Allah Subhana Hu Wata Alla sent for the benefits of mankind.

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 want to express my sincere gratitude to my supervisor Assoc. Prof. Yasuhiko Morimoto, Department of Information Engineering, Hiroshima University, Japan for giving me the opportunity to explore new ideas in the field of knowledge discovery and data mining. Without his moral courage and excellent guidelines, it was not possible for me to complete this work. His advanced knowledge, inspiration and above all diligence expertise in this field provide me many opportunities to learn new things to build my carrier as a researcher.

I acknowledge my thanks to the members of my dissertation committee Prof. Satoshi Fujita, Prof. Takio Kurita and Assoc. Prof. Junichi Miyao, for their valuable comments and suggestions those helped me a lot to improve my research.

I am thankful to the Japanese Government for providing the MEXT scholarship to pursue my study. I am indebted to Chittagong University of Engineering and Technology (CUET), Bangladesh that has granted me a study leave to attend the PhD program at HU.

Special thanks to all my co-authors and lab members for their support and contributions during my research. I am also grateful to all Bangladeshi of Hiroshima for their advice, co-operation, and warm relationships during my stay in Japan.

Last but not least, I am deeply grateful to my wife Moshammat Faijun Nessa for her comprehension, dedication, care, and support throughout the research work.

Abstract

Conventional SQL queries take exact input and produce complete result set. However, with massive increase in data volume in different applications, the large result sets returned by traditional SQL queries are not well suited for the users to take effective decisions. Therefore, there is an increasing interest in queries like top-*k* queries and skyline queries those produce a more concise result set.

Top-*k* queries rely on the scores of the objects to evaluate the usefulness of the objects. In this type of queries, users require to define their own scoring function by combining their interests. Based on the user defined scoring function, the system sorts the objects by their scores and outputs the top-*k* objects in the ranking list as the result. However, defining a scoring function by the users is a major draw of the top-k queries as in the large data sets where there are many conflicting criteria exist, it is very difficult for the users to define the scoring functions by themselves.

To overcome this disadvantage of top-*k* queries, skyline queries were proposed. Skyline queries do not rely on scoring functions to retrieve objects. Instead they use the concept of dominance relation. An object is said to dominate another object if it is not worse in any of the dimensions and is better in at least one of the dimensions. Given a set of objects with multiple dimensions, an object would not be retrieved if it is dominated by some other objects. From the result of skyline objects the user can choose promising objects for them and make further inquiries. Therefore, such skyline query functions are important for several database applications, including customer information systems, decision support, data mining and visualization, and so forth.

From the introduction of skyline queries in 2001, skyline queries are treated as an important approach for information filtering and there are many research works on skyline queries considering either a sole database or distributed databases with almost no consideration about the privacy of data. Although there are few considerations about the privacy of data while computing skyline queries from a sole database, there is no consideration about individual's privacy during the computation of skyline results from distributed databases. However, with the rapid growth of data volume and network infrastructure, in most cases data are stored at distributed databases nowadays.

Considering these facts, first part of this dissertation deals with preserving the privacy of data while computing skyline results from distributed databases. In this part, at first, we introduce an agent-based parallel computation framework for skyline sets queries from distributed databases. The computation is performed in such a way that the privacy of individual's is preserved. In addition of preserving individual's privacy, our approach is robust against the outliers and the frequently update situation.

However, in our above approach there are possibilities of disclosure of record's values from the return skyline sets in statistical compromisable situations. Considering this fact, in this part, we also introduce an efficient protection mechanism against statistical compromisable situations. In this part, we also consider the mechanism of dealing with missing values in the databases during skyline sets queries.

The second part of this dissertation focuses on selecting spatial objects from spatial databases considering environmental influences such as the presence of restaurants and supermarkets while selecting skyline hotels. Here, we utilize the concept of skyline queries. Conventional skyline queries select such spatial objects like hotels based on non-spatial attributes such as price and rating of hotels and there is no consideration of utilizing surrounding environments. In this dissertation, we propose two methods for utilizing surrounding environments. Our first approach considers the best value in each attribute of each surrounding facility, while our second method considers the number of objects of each type of facility in the surrounding environments. Our first approach is well suited for hotel recommendation systems, while our second approach can help in real estate recommendations.

The last part of this dissertation considers a problem of selecting spatial objects for a group of users located at different positions, since recent social network services can connect users and make such groups. Here, we also consider the concept of skyline queries. If a group wants to find a restaurant to hold a meeting, we have to select a convenient place for all users. Although there are many research works on spatial skyline queries, none of them can efficiently compute skyline objects in such a scenario. Considering this fact, we propose an efficient skyline query algorithm to select spatial objects, considering both spatial and non-spatial information.

To summarize, this dissertation addresses three different sophisticated information filtering methods based on the concept of skyline queries: skyline sets queries from distributed databases, skyline queries by utilizing surrounding environments, and spatial skyline queries for a group of users.

Contents

List of Figures

List of Tables

Chapter 1

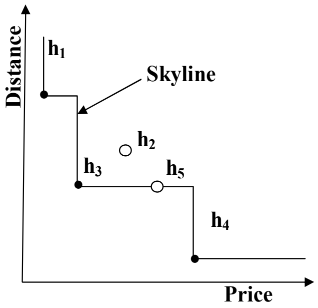
Introduction

With the increase of data volume in different applications, advanced query operators are necessary in order to help users to filter the huge amount of available data by selecting a set of promising data objects. Skyline queries can help users in this regard. Different from traditional SQL queries that return a complete result set, skyline queries return all non-dominated objects from a given data set. An object is said to be non-dominated if it is not worse than any other object in any of the attributes and is better in at least one of the attributes. If we consider the example of Figure 1.1, we can see that Figure 1.1 (a) consists a list of five hotels, each of which contains two numerical attributes "Price" and "Distance". In the list, h_2 and h_5 are dominated by h_3, while others are not dominated by any other hotel. Therefore, the skyline of the list is $\{h_1, h_3, h_4\}$. Such skyline results are important for users to take effective decisions over complex data having many conflicting criteria. From the introduction of skyline queries for information filtering by Borzonyi et al. [1] in 2001, it has attracted a lot of attention in the research community focusing different aspects and solutions of skyline queries. For information filtering, in this dissertation, we concentrate on three different problems: (i) privacy preserving skyline queries from distributed databases (ii) skyline queries considering the influences of the surrounding environments, and (iii) spatial skyline queries for a group of users.

This chapter is organized as follows. Section 1.1 contains the motivation, followed by the research question in Section 1.2. Section 1.3 covers the research contributions. Finally, in Section 1.4, we outline the organization of the dissertation.

ID	Price	Distance
h_1	3	8
h_2	5	4
h_3	4	3
h_4	9	2
h_5	7	3

(a) Hotels



(b) Skyline

Figure 1.1: Skyline example

1.1 Research Motivation

Nowadays, we are facing a flood of data due to the development of computer technologies. This flood brings us much more information than ever before and changes our lives even when we are not aware of it. The volume of data grows dramatically and comes in various forms, such as scientific tables, commercial records, stock charts, hypertexts, and multimedia. Possessing an enormous amount of data is meaningless if we cannot acquire knowledge that is helpful for the users. Skyline queries can filter out less important information and generate good information for users so that the users can take correct decisions.

However, it is necessary to consider the privacy issues while filtering data using skyline queries as privacy of data is very crucial nowadays. From the introduction of skyline queries in 2001 [1], there are many research works on skyline queries considering either a sole database [2, 3, 4, 5, 6, 7, 8, 9] or distributed databases [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24] with almost no consideration about the privacy of data. Although there is some considerations about the privacy of data while computing skyline queries from a sole database in [42, 43, 44], there is no consideration about individual's privacy during the computation of skyline results from distributed databases. However, with the rapid growth of data volume and network infrastructures, in most cases data are stored at distributed database nowadays and security of data in distributed databases are more important than the security of data in a sole database as data stored in distributed databases may belong to different data owners. Considering these facts, at first, we were motivated to compute skyline sets queries from distributed databases that can preserve individual's privacy. Let $DB_1, DB_2, \cdots, DB_m$ be the m databases with same schema. Also, let s is the number

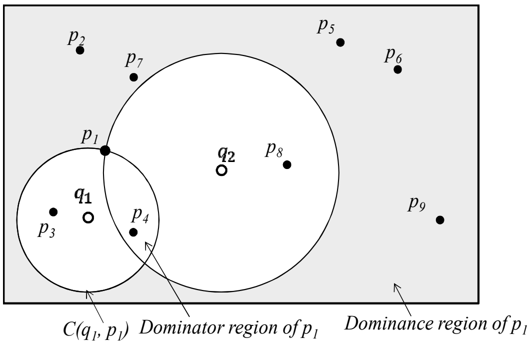


Figure 1.2: Spatial skyline example

of objects in each set and n is the total number of objects in m databases. Skyline sets queries return skyline s-sets from m such databases in such a way that the privacy of individual's is preserved.

Next, we observed that current skyline query algorithms just focuses on the attributes of the objects and cannot utilize the influences of the surrounding environments efficiently while computing skyline results. However, when we select a spatial object in a database, surrounding environments can be as important as other attributes. As for example, a hotel may be a good candidate if it has good restaurants in its surrounding area although the hotel's rating is not good. Motivated with this fact, we consider skyline queries considering the influences of surrounding environments.

Finally, let us consider that we need to recommend some good restaurants for the members of a multidisciplinary task force team located at different offices those want to put together in a restaurant to hold a lunch-on meeting. As recent social network services can connect users and make such groups easily, we need to provide useful information for such groups. To retrieve accurate skyline results for such a group, we need to consider a variation of the spatial skyline query [26] that can efficiently compute the skyline results considering spatial sub-space and can integrate the results of non-spatial sub-space. The problem of the spatial skyline query can be defined as follows. Given the two sets P of data points and Q of query points, the spatial skyline of P with respect to Q is the set of those points in P, which are not *spatially dominated* by any other point of P. A data point p_1 is said to spatially dominate another point p_2 with respect to Q iff we have $d(p_1,q_i) \leq d(P_2,q_i)$ for all $q_i \in Q$ and $d(p_1,q_j) < d(p_2,q_j)$ for some $q_j \in Q$, where $d(p,q)$ is the Euclidean distance between p and q. Figure 1.2 shows a set of nine points and two query points q_1 and q_2 in a plane. The point p_1

spatially dominates the point p_2 since both q_1 and q_2 are closer to p_1 than to p_2. The spatial skyline query returns a set of such non-dominated objects.

However, current spatial skyline query algorithms [26, 27, 28, 29, 30, 31] do not consider efficient computation of spatial skyline objects based on both spatial and non-spatial features. Considering this fact, we consider a spatial skyline queries that can compute accurate results for such groups considering both spatial and non-spatial features of objects.

1.2 Research Questions

The research in this dissertation considers several research questions related to information filtering using skyline queries and provide their efficient solutions. The list of research questions addressed in this dissertation are as follows:

- **RQ 1.** How to compute skyline sets queries from distributed databases efficiently?

  RQ 1.2 How to deal with compromisable situations during the skyline sets queries from distributed databases?

- **RQ 2.** How to select spatial objects efficiently by using skyline queries considering surrounding environments?

  RQ 2.1 What is the efficient way to compute skyline queries while considering the best values in the attributes of the surrounding facilities?

  RQ 2.2 What is the efficient way of computing skyline queries while considering the objects count of each surrounding facility?

- **RQ 3.** How to compute skyline results efficiently for the group of users considering both spatial and non-spatial features of objects?

In Section 1.3, we present the main contributions presented in this dissertation in terms of the research questions.

1.3 Research Contributions

In this section, we describe the main contributions in terms of the research questions of Section 1.2.

Information Filtering By Using Skyline Sets Queries from Distributed Databases

Let s be the number of objects in each set and n be the number of objects in the dataset distributed over m databases. The number of sets in the databases amounts to ${}_{n}C_{s}$. We propose an efficient method for skyline sets queries from ${}_{n}C_{s}$ sets to retrieve important information while filtering out less important information without disclosing individual record's values. In our approach, we provide a parallel computation framework. We also provide the solution to prevent the discloser of individual record's values due to statistical compromisable situations. We explain these approaches in Chapter 3.

Selection of Spatial Objects Considering Surrounding Environments

We develop two methods based on the concept of skyline queries those utilize the features of objects as well as surrounding environments for selecting spatial objects. Our first approach considers the best value in each attribute of each surrounding facility, while our second method considers the number of objects of each type of facility in the surrounding environment. Our first approach is well suited for hotel recommendation system while our second approach can help in real estate recommendation. For our first approach, we use a grid-based data structure, while we use the concept of *aR*-tree [57] for our second approach. Besides theoretical guarantees, our comprehensive performance studies indicate that both of our approach are very much efficient. We present these approaches in Chapter 4.

Selecting Spatial Objects for for a Group of Users

We develop an efficient algorithm for spatial skyline computation for selecting spatial objects for a group of users considering both spatial and non-spatial features of the objects. At first, we compute skyline objects based on spatial sub-space. We use the concept of *VoR*-tree [58] for computing skyline objects in spatial sub-space. Later, we use the non-spatial features of skyline objects of spatial sub-space to obtain some other skyline objects in non-spatial sub-space. Several extensive experiments on real and synthetic data set shows the effectiveness of our method. This methodology is also presented in Chapter 4.

1.4 Thesis Organization

In this section, we outline the organization of the dissertation.

Chapter 1. This chapter presents the motivation, the research questions, and the contributions achieved during the PhD program.

Chapter 2. In this chapter, at first, we describe the necessity of information filtering. Next, we elaborate the concepts of skyline queries and their variants and provide a brief review of skyline query algorithms. Finally, we describe different applications of skyline queries.

Chapter 3. In this chapter, at first, we explain the privacy related issues during skyline queries. Next, we describe basic concepts of skyline sets queries and its usefulness. Later, we explain frameworks for computing skyline sets from distributed databases.

Chapter 4. In this chapter, we provide approaches of selecting spatial objects by using skyline queries. Here, we provide three different approaches to compute spatial objects. First two methods of this chapter utilize the influences of the surrounding environments while selecting spatial objects. Third method computes spatial objects for a group of users considering both spatial and non-spatial features of the objects as well as the locations of the users.

Chapter 5. This chapter concludes the dissertation outlining the main contributions and future research directions.

Chapter 2

Information Filtering and Skyline Query

In this chapter, at first we address the necessity of information filtering. Next, we elaborate the concepts of skyline queries and their variants. Then, we provide a brief review of skyline query algorithms. Finally, we describe the applications of skyline queries.

This chapter is organized as follows. Section 2.1 describes the necessity of information filtering. Section 2.2 describes the concept of skyline queries. We also provide a brief review of existing skyline query algorithms in this section. In Section 2.3, we present different application areas of skyline queries.

2.1 Necessity of Information Filtering

We live in an information age and different applications produced an overwhelming amount of data now a days. Efficient filtering of this huge volume of data is very important to provide users accurate and desirable information within a short period of time. As for example while booking a hotel, it is very difficult to select a hotel from a list of hundreds hotels instead of selecting it from ten hotels. Consider the hotel database as shown in Table 2.1 that has three numerical attributes “Price”, “Rating”, and “Distance”. Also, consider that smaller values in each attribute is better. If anyone wants to find best hotels from Table 2.1, it is difficult for him to select such a hotel quickly. However, if we consider the records of Table 2.2, we can find that it is much

Table 2.1: Hotel database

ID	Price	Rating	Distance
h_1	7	6	8
h_2	4	5	4
h_3	2	3	3
h_4	1	7	4
h_5	10	9	3
h_6	5	2	3
h_7	9	6	5
h_8	1	4	4
h_9	4	3	9
h_{10}	5	1	7

Table 2.2: Interesting Hotels

ID	Price	Rating	Distance
h_3	2	3	3
h_6	5	2	3
h_8	1	4	4
h_10	5	1	7

easier to select interesting hotels from this table. This is because we filtered out less important information of Table 2.1 and stored only important information in Table 2.2.

Again consider that a company is trying to identify problems in its company by looking its salary database. As there are many employees in the company, it is difficult to check all employees salary and experience separately. However, if we can filter out the information of employees those have good salaries according to the companies salary policy, we can easily find the employees those cannot meet perform well for the company. As for example, if we look the records of Table 2.3, we cannot easily find the problem. However, if we look at Table 2.4 instead of Table 2.3, we can easily find that there are some problems in the company For example, in there are two employs whose salaries are very low with regards to their service period. That means these employees are not doing well for the company. Table 2.4 helps us to find such records easily because it filtered records from Table 2.3 and presents only a small number of records from which the salary personnel of the company can take the decision.

Traditional SQL queries cannot perform efficient filtering of such data as they return a complete result set. Preference queries are also not suitable for filtering such data as preference queries rely on inputs from the users, while it is often very difficult for the users to provide their preferences. Skyline queries can help in this regards as skyline queries can retrieve all interesting results from database. The records in Table 2.2 and Table 2.4 are skyline records of the dataset of Table 2.1 and Table 2.3,

Table 2.3: Salary database

ID	Salary	Experience
o_1	3	9
o_2	5	4
o_3	7	2
o_4	2	9
o_5	9	3
o_6	4	4
o_7	4	5
o_8	6	5
o_9	8	2
o_{10}	9	1

Table 2.4: Uncommon Records

ID	Salary	Experience
o_3	7	2
o_4	2	9
o_6	4	4
o_10	9	1

respectively.

2.2 Skyline Queries

The basic idea of skyline queries [1] came from some old research topics like contour problem [60], maximum vector [61], and convex hull [62].

Let us consider a k-dimensional database DB and $D_1, D_2, \cdots, D_k$ are k attributes of DB. Let $O_1, O_2, \cdots, O_r$ be r objects (tuples) of DB. Consider that $O_i.D_j$ denotes the j-th dimension of object O_i.

Definition 2.1 (Dominance) *An object O_i is said to dominate another object O_j, which we denote as $O_i \prec O_j$, if $O_i.D_s \leq O_j.D_s$ for all dimensions D_s ($s = 1, \cdots, k$) and $O_i.D_t$ $< O_j.D_t$ for at least one dimension D_t, ($1 \leq t \leq k$).*

Definition 2.2 (Skyline) *An object O_i is said to be in the skyline of DB if there is no other object O_j ($i \neq j$) in DB such that O_j dominates O_i. If there exist such an O_j, we say that O_i is dominated by O_j and O_i is not in the skyline of DB.*

The skyline query retrieves the objects those are in the skyline. For example, consider the five different vacation packages as shown in Table 2.5.

From the data of Table 2.5, we can see that P_1 dominates P_3 and P_5 and P_2 dominates P_4. P_1 and P_2 are not dominated by any other object. So, the skyline of the dataset of

Table 2.5: Skyline Example

Vacation Package	*Price*	*Hotel Rank*	*Number of Stops*
$\mathbf{P_1}$	1600	2	1
$\mathbf{P_2}$	2000	1	2
$\mathbf{P_3}$	2000	3	2
$\mathbf{P_4}$	2500	1	3
$\mathbf{P_5}$	2000	3	1

Table 2.5 is $SKY(r) = \{P_1, P_2\}$. Skyline queries retrieve such objects from the given dataset.

The notion of skyline queries can be extended to subspaces, where a subspace skyline query [2, 3] only refers to a user-defined subset of attributes. If we consider our k-dimensional database DB again, a subset u of k ($u \subseteq k$) is referred to as a subspace of k. Then, we can define the subspace skyline of u as follows.

Definition 2.3 (Subspace Skyline) *The subspace skyline of u is a set of objects from DB that are not dominated by any other object on subspace u.*

Consider our three-dimensional dataset as depicted in Table 2.5 again, where the skyline objects are $SKY(r) = \{P_1, P_2\}$. However, if we consider the subspace $u = \{Price, Number\ of\ Stops\}$, the subspace skyline object is $SKY_u(r) = \{P_1\}$.

Another variation of skyline queries is constrained skyline query [4]. A constrained skyline query returns the set of skyline points from the dataset those satisfy the given constraints.

For example, if we consider our dataset of Table 2.5 and assume that a user is interested in the packages those price are in the range of 2000-2500, then the result of this constrained skyline query is $\{P_4, P_5\}$.

Another variation of skyline queries is spatial skyline queries [26]. Different from other skyline queries, a spatial skyline query considers two sets of points: a set of data points P and a set of query points Q and each point in P and Q has spatial attributes, which are 2-dimensional coordinate attributes.

Let us consider that the distance function $d(p,q)$ returns the Euclidean distance between a pair of points p and q, where $p \in P$ and $q \in Q$.

Based on the above considerations, we can define “spatial dominance” and “spatial skyline” as follows.

Table 2.6: Data points locations

ID	x-coordinate	y-coordinate
$\mathbf{r_1}$	3	9
$\mathbf{r_2}$	7	5
$\mathbf{r_3}$	7	7
$\mathbf{r_4}$	5	1
$\mathbf{r_5}$	4	4
$\mathbf{r_6}$	4	8
$\mathbf{r_7}$	5	6
$\mathbf{r_8}$	1	3
$\mathbf{r_9}$	5	3
$\mathbf{r_{10}}$	9	3

Table 2.7: Query points locations

ID	x-coordinate	y-coordinate
$\mathbf{u_1}$	4.5	5.5
$\mathbf{u_2}$	5	6.8
$\mathbf{u_3}$	6	5
$\mathbf{u_4}$	5	3.8

Definition 2.4 (Spatial Dominance) *We say that a data point p_1 "spatially dominates" another data point p_2 if and only if $d(p_1,q) \leq d(p_2,q)$ for every $q \in Q$, and $d(p_1,q) < d(p_2,q)$ for some $q \in Q$.*

Definition 2.5 (Spatial Skyline) *A point $p \in P$ is said to be in the spatial skyline with respect to Q if and only if p is not spatially dominated by any other point of P.*

Consider Figure 2.1 that shows the pictorial representation of data points and query points of Table 2.6 and Table 2.7, respectively. From Figure 2.1, we can find that r_2, r_5, r_7, and r_9 are not spatially dominated, while all other points are spatially dominated. So, our spatial skyline result based on the data points of Table 2.6 and query points of Table 2.7 is $SPSKY = \{r_2, r_5, r_7, r_9\}$.

Since the introduction of skyline queries [1] in 2001, more than a hundred papers have been published [25] in reputed journals and conferences. These papers have not only studied efficient skyline computation in centralized or distributed systems but also proposed variations of the traditional skyline operator and studied different premises. Below we provide a brief review of skyline research.

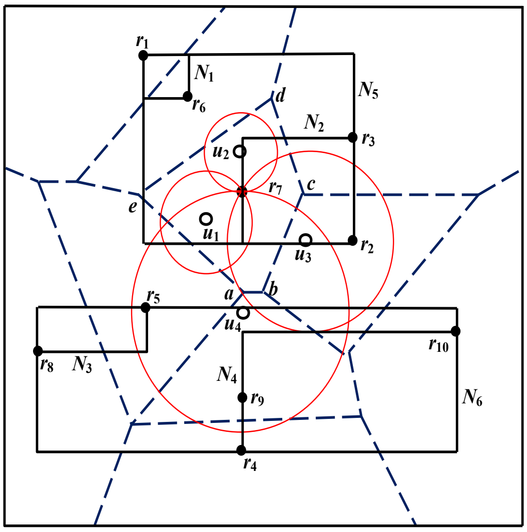


Figure 2.1: Location of data points and query points

2.2.1 Related Works on Skyline Queries

Block Nested Loops

The basic approach of computing skyline of a given data set D is to check the dominance of each data object of D against all other data objects in D. If we find that the object is not dominated by any other object, we report the object as a skyline object. The Block Nested Loops (BNL) [1] is an iterative algorithm that repeatedly scans a set of records and performs such dominance check to obtain the skyline objects. For faster dominance check, it maintains a window of candidate skyline objects in the main memory and evaluate the data objects in the data set one by one. When an object P is read from the input relation, P is compared with the objects in the window. There will be one of the three cases for P.

Case 1: P is dominated by an object in the window.

Case 2: P dominates some objects in the window.

Case 3: P is incomparable with all records in the window, i.e. P neither dominates nor being dominated.

Case 1 indicates that P cannot be in the skyline. In such a situation, P is discarded without further comparison with other candidate objects. In **Case 2** the objects those are dominated by P are removed from the window. In **Case 3** P is either inserted into the window if there is sufficient room in the window, or written to a temporary file on

Table 2.8: An example dataset

ID	x	y
$\mathbf{P_1}$	7	6
$\mathbf{P_2}$	4	5
$\mathbf{P_3}$	2	3
$\mathbf{P_4}$	1	7
$\mathbf{P_5}$	10	9
$\mathbf{P_6}$	5	2)
$\mathbf{P_7}$	9	1
$\mathbf{P_8}$	1	4
$\mathbf{P_9}$	4	2
$\mathbf{P_{10}}$	5	1

disk.

After the dataset is scanned, all the candidate objects in the window which are processed before the creation of the temporary file are output as part of the skyline. Then, BNL evaluates all data objects in the temporary file in the same way again until no temporary file is scanned. Finally, all candidate objects in the window are output as part of the skyline.

Now, consider the example dataset as shown in Table 2.8. Also consider that the window size is 3 and smaller value in each dimension is better. From Table 2.8, we can see that there are ten two-dimensional objects. BNL computes the skyline of these objects in the following way.

Consider that the objects are processed in sequential order. Initially, P_1 is inserted into the window as the window is empty. Then BNL considers next data object that is P_2. We can see that P_1 is dominated by P_2. So, P_1 is removed and P_2 is inserted into the window. After P_3 is processed, only P_3 is in the window. Then, P_4 is inserted into the window as it is not dominated by the data object P_3 of the window. When we process P_5 and P_6 in the same way, we can find that P_5 is dominated and P_6 is not dominated by the objects in the window. Hence, P_6 is inserted into the window. When P_7 is processed, we can see that it is not dominated by the objects in the window (i.e., objects P_3, P_4, P_6) and the window is full. So, P_7 is written to the temporary file. Let us consider that the time stamp of this temporary file is 7.

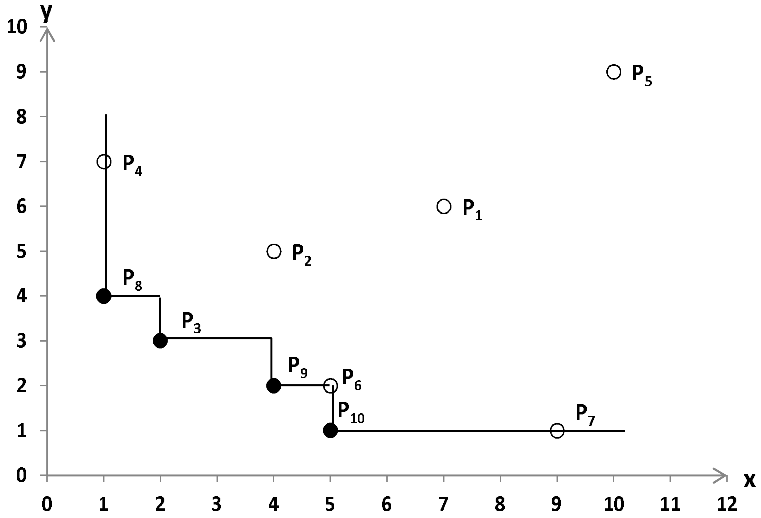


Figure 2.2: Skyline of the example dataset of Table 2.8

In the end of this iteration, the data objects in the window are P_3, P_8, and P_9, and the temporary file contains data objects P_7 and P_{10}. Then, the data objects in the window which is processed before time stamp of the temporary file (i.e., 7) are output as skyline objects. In this case, P_3 is output. After that BNL continues to process the temporary file in the same way till no data object is required in any temporary file. Finally, we obtain P_3, P_8, P_9, and P_{10} as the skyline result as shown in Figure 2.2.

To speed up the comparisons between the data objects in question and the candidate objects, the window is organized as *self-organizing list*. In this list, when a candidate object is found dominating other data objects, it is moved to the beginning of the list. Consequently, it is first compared when evaluate the next data object. This variant reduces the number of comparisons as the data objects which dominate more others may be compared first.

Divide and Conquer

To compute the skyline for a set of data, the Divide and Conquer (D&C) approach [1] divides the data objects into several parts recursively so that each partition fits in memory. After all the local skyline are computed, D&C merges theses skylines to obtain the final skyline result. Now consider the computation of skyline of the dataset of Table 2.8 using Divide and Conquer approach.

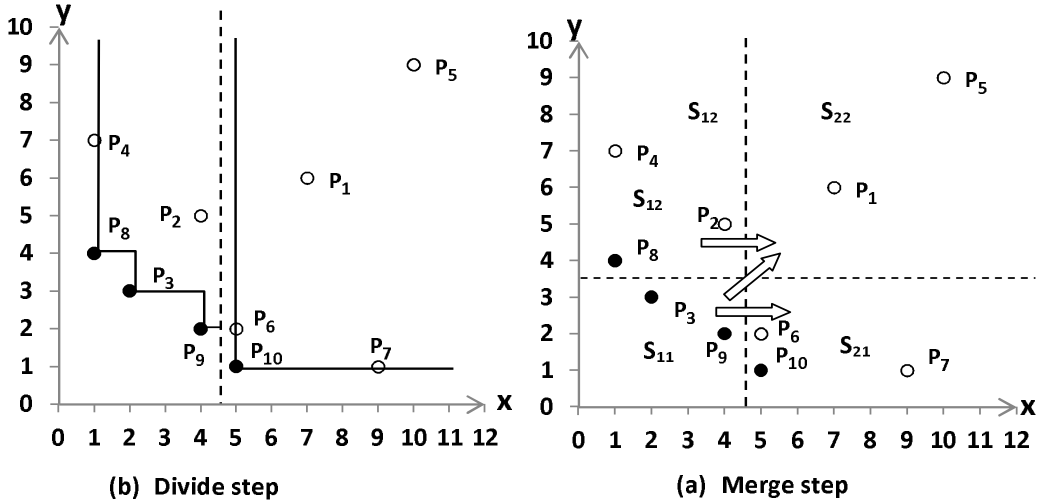


Figure 2.3: Skyline computation using divide and conquer strategy

First, D&C calculates the median of all the objects on dimension x and divides the data points into two parts, S_1 and S_2. S_1 contains the data objects whose values on dimension x are less than the median, i.e. $\{P_2, P_3, P_4, P_8, P_9\}$. S_2 contains all others, i.e. $\{P_1, P_5, P_6, P_7, P_{10}\}$. Then, skylines of S_1 and S_2 are computed respectively. This is done by recursively partitioning each part until only one data object is remained. In that case, it is easy to compute the skyline. This step of D&C is called divide step.

The local skylines of S_1 and S_2 are shown in Figure 2.3(a). After all local skylines are computed, D&C eliminates the data objects in local skyline of S_2 which are dominated by the local skyline objects of S_1 to obtain the overall skyline (merge step). In order to do elimination efficiently, the local skyline objects of S_1 and S_2 are further partitioned into 2 parts by the median of the local skyline objects of S_1 on dimension y, respectively. The further partitioning technique is shown in Figure Figure 2.3(b). Clearly, the data objects in S_{21} have smaller coordinates on dimension y than those of data objects in S_{12}. As a result, the comparison of S_{21} with S_{12} is saved. On the other hand, the data objects in S_{22} have greater coordinates on both dimensions x and y than those of data objects in S_{11}. In other words, all the data objects in S_{22} are dominated by any one in S_{11}. So, they are eliminated immediately without comparison. Now, D&C only needs to compare S_{21} with S_{11} to eliminate the non-skyline objects in S_{21}. At the end of the computation, final skyline result is $\{P_3, P_8, P_9, P_{10}\}$.

If the dataset does not fit into main memory, D&C requires to read and write the

Table 2.9: An example of SFS algorithm

ID	x	y	Normalized value of x	Normalized value of y	Entropy Value
$\mathbf{P_1}$	7	6	0.7	0.6	1.00
$\mathbf{P_2}$	4	5	0.4	0.5	0.74
$\mathbf{P_3}$	2	3	0.2	0.3	0.44
$\mathbf{P_4}$	1	7	0.1	0.7	0.63
$\mathbf{P_5}$	10	9	1.0	0.9	1.34
$\mathbf{P_6}$	5	2	0.5	0.2	0.59
$\mathbf{P_7}$	9	1	0.9	0.1	0.74
$\mathbf{P_8}$	1	4	0.1	0.4	0.43
$\mathbf{P_9}$	4	2	0.4	0.2	0.52
$\mathbf{P_{10}}$	5	1	0.5	0.1	0.50

dataset times during the partitioning process, thus incurring significant IO overhead. To improve D&C performance when main memory is limited, M-way D&C algorithm is introduced [1]. The basic idea of M-way D&C is that during each partitioning process the dataset is divided into m parts instead of 2 parts such that every part is expected to fit into main memory. This idea is also applied in merge step such that each part should occupy at most half of the available main memory. So, no additional IO cost is required during each comparison.

Sort Filter Skyline

Sort First Skyline algirithm (SFS) [5] is a variant of BNL. In order to improve BNL, SFS introduces the entropy value $E(p)$ for each data object $P = (P[1], P[2], \cdots, P[n])$. The entropy $E(p)$ is represented by the formula as shown in equation (2.1).

$$E(P) = \sum_{i=1}^{n} \ln(P'[i] + 1) \tag{2.1}$$

In equation (2.1), $P'[i]$ is the normalized value of $P[i]$. Obviously, given two data objects P_1 and P_2, P_1 cannot dominate P_2 if $E(P_1)$ is greater than or equal to $E(P_2)$. Based on this observation, SFS first sorts all the data objects in non-decreasing order of their entropy values. After that, SFS processes the sorted dataset in the same way as BNL.

Take the same dataset shown in Table 2.8 as an example. The corresponding normalized values and entropy value of the data objects are listed in Table 2.9. Suppose the window size is 3 and now after sorting on the entropy values the processing order of data objects is P_8, P_3, P_{10}, P_9, P_6, P_4, P_2, P_7, P_1, and P_5. Clearly, the first three objects P_8, P_3, and P_{10} do not dominate each other. So, all of them are inserted into the window and the window is full now. Then, P_9 is processed and it is written to the temporary file as it is not dominated by any data objects in the window. After that, all other data objects are processed and discarded due to each of them is dominated by one of P_8, P_3, and P_{10}. Now, the first iteration is done. In the beginning of the next iteration, all the data objects in the window are output as the skyline objects as they are processed before the temporary file. Since O_9 is the only data object in the temporary file SFS terminates after it is processed.

Compared with BNL, SFS has the following advantages.

1. The number of comparisons among data objects is reduced. As the data object with smaller entropy value is evaluated first, the skyline object could be found earlier. Therefore, the number of comparisons between data objects and the non-skyline objects in the window, which is unnecessary, is reduced.

2. SFS is an progressive skyline algorithm while BNL is not. When a data object P is added into the window, it is guaranteed to be a skyline object. This is because all non-processed data objects do not dominate P as they have greater entropy values than $E(P)$.

Linear Elimination Sort for Skyline

Linear Elimination Sort for Skyline algorithm (LESS) [6] improves SFS by integrating external merge sort procedure tightly into skyline computation. Similar to SFS, LESS first sorts the dataset according to entropy values of all data objects and then computes the skyline in the same way as BNL. In order to eliminate the data objects efficiently, LESS makes the following two major changes during external merge sorting procedure.

1. It maintains an elimination-flter (EF) window in pass 0 of the external merge sort procedure to eliminate some non-skyline data objects; and

2. It combines the final pass of the external merge sort procedure with the skyline examination procedure of BNL algorithm.

Specifically, a small EF window is maintained in pass 0 of the external sort routine. Copies of the data objects with the best entropy values are kept in this window. When a block of data objects is read in, these data objects are compared against those in the EF window first. The data objects dominated by those in the EF window are dropped as well. Then, the data objects in the EF window are replaced by those with the best entropy values among the surviving new data objects and those in EF window.

The merge passes of the external merge sort procedure in LESS are the same as for external merge sort, except for the final merge pass which is combined with the initial skyline examination procedure of BNL. In the final pass, skyline-filter (SF) window is maintained in which the skyline candidates are stored. Besides sorting, each data objects processed in the final pass is compared with the data objects in SF window. If the new data object is dominated by some in SF window, it is discarded immediately. If the data objects in SF window is dominated by the new data objects, they are discarded as well. If the new data object is not discarded after examination, it is inserted into SF window as a new candidate. When SF window is full, a temporary file is created. Obviously, such integration of the final merge pass and the skyline examination procedure saves a pass over the data. Note that such optimization is also implemented in many database systems for the standard two-pass sort-merge join.

Compared with SFS, LESS should consistently perform better because for the following reasons.

(1) The dataset to be processed after pass 0 in LESS is smaller than that of SFS; this may also cause that more passes are required in SFS to complete the sort;

(2) LESS saves at least one passes since it combines the final merge pass with the skyline examination procedure.

Bitmap

Bitmap algorithm [7] encodes each data object with a bitmap according to the rank of its value on each dimension. All the bitmaps enable the algorithm to efficiently determine whether a data object is a skyline object by bitwise operations (i.e., AND).

Consider a data object P = ($P[1]$, $P[2]$, $\cdots$, $P[n]$), where n is the dimensionality. Each coordinate $p[i]$, $(1 \leq i \leq n)$ is converted into m_i-bit vector, where m_i is the number of distinct values on the i-th dimension, in which the $(m_i - rank(p_i) + 1)$ most

Table 2.10: An example of Bitmap algorithm

ID	x	y	**Bit vectors for** x	**Bit vectors for** y
$\mathbf{P_1}$	7	6	1110000	11100000
$\mathbf{P_2}$	4	5	1111100	11110000
$\mathbf{P_3}$	2	3	1111110	11111100
$\mathbf{P_4}$	1	7	1111111	11000000
$\mathbf{P_5}$	10	9	1000000	10000000
$\mathbf{P_6}$	5	2	1111000	11111110
$\mathbf{P_7}$	9	1	1100000	11111111
$\mathbf{P_8}$	1	4	1111111	11111000
$\mathbf{P_9}$	4	2	1111100	11111110
$\mathbf{P_{10}}$	5	1	1111000	11111111

significant bits are 1 and others are 0. After conversion, every data object is mapped to an m-bit vector where m is represented by equation (2.2).

$$m = \sum_{i=1}^{n} m_i \tag{2.2}$$

As an example, Table 2.10 shows the corresponding bit vectors for the data objects shown in Table 2.8. After converting all data objects to bitmaps, every data object can be efficiently determined whether it belongs to the skyline by calling bitwise operations on the bitmaps. Specifically, given a data object P = ($P[1]$, $P[2]$, $\cdots$, $P[n]$), bitmap algorithm first generates d bit vectors $b_1, b_2, \cdots, b_n$, where $b_i(1 \leq i \leq n)$ is juxtaposing the corresponding rank($p[i]$) bits of every data object. The 1's in the result of b_1 & b_2 & $\cdots$ & b_n indicate the data objects which dominate O. Obviously, if there is only single 1 in the result, the considered data object is a skyline object.

Continuing the above example to check whether P_2 is a skyline object. For P_2, the corresponding bit vectors b_1 and b_2 are 0111000110, and 0010011111, respectively. Then the result of b_1 and b_2 is 0010000110, which indicates P_2 is dominated by P_3 and P_8. As a result, P_2 is not a skyline object. On the other hand, for data object P_{10}, the result of b_1 & b_2 is 0111010111 & 0000001001 = 0000000001, which has only single 1. Therefore, P_{10} belongs to the skyline. To obtain the entire skyline, Bitmap algorithm repeats the same examination for every data object in the dataset.

Table 2.11: An example of index algorithm

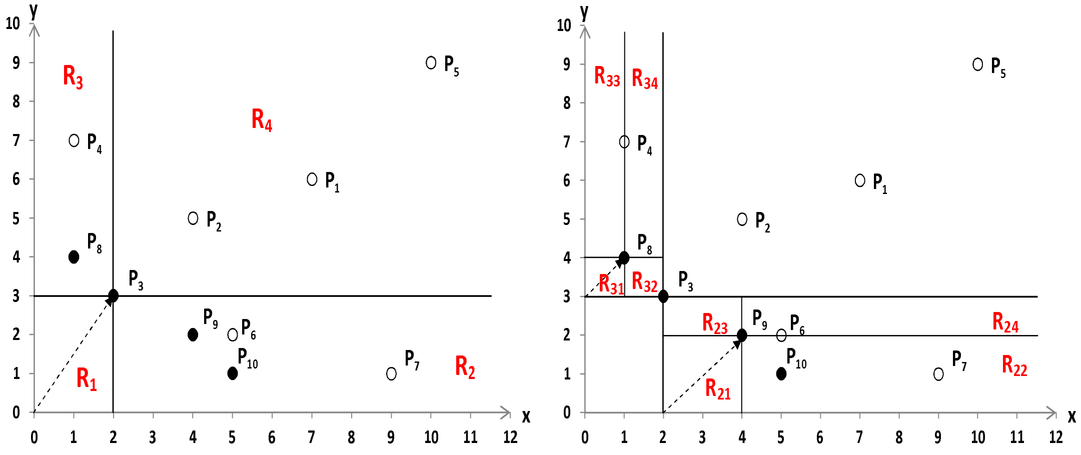
minValue	**batch**	*minValue*	**batch**
1	$P_8(1, 4)$	1	$P_{10}(5, 1)$
1	$P_4(1, 7)$	1	$P_7(9, 1)$
2	$P_3(2, 3)$	2	$P_9(4, 2)$
2	$P_6(5, 2)$	4	$P_2(4, 5)$
6	$P_1(7, 6)$	9	$P_5(10, 9)$

Index

Given a set S of n-dimensional data objects, Index algorithm [7] organizes S into n B^+-tree indices. A data object $P = (P[1], P[2], \cdots, P[n])$ is assigned to the i-th $(1 \leq i \leq n)$ B^+-tree index if and only if $P[i]$ is the minimum coordinate among all coordinates of P. The key of each B^+-tree index is the minimum coordinate (denoted by *minValue*) of each data object. The data objects in the same B^+-tree index which have same *minValue* are maintained in a batch.

To compute skyline, the maximum value of all the coordinates of the current skyline objects is maintained, which is denoted by *maxValue*. Iteratively, Index algorithm examines each B^+-tree index and processes the batch which has the smallest *minValue*. *minValue* of this batch is first compared with *maxValue*. Obviously, if *maxValue* is smaller than or equal to *minValue*, there should be some skyline object in the current skyline which dominates the data objects in this batch and all other unprocessed data objects in the same B^+-tree index as well. Therefore, the batch and all other unprocessed data objects in the same B^+ tree index can be safely discarded. Otherwise, within the batch processed, a local skyline is computed first and remove the data objects which are dominated by others. Then, the remained data objects are compared with the skyline objects computed so far. If the data object in question is dominated by some current skyline object, it is discarded. Otherwise, it is inserted into skyline as a new skyline object. Once a new skyline object is found, *maxValue* is updated. Index algorithm returns the skyline result after the batches of all B^+-tree indices are processed.

Consider the same dataset shown in Table 2.8. All the batches in the B^+-tree indices are listed as shown in Table 2.11 in the increasing order of *minValue*. Initially, Index algorithm processes the batches with *minValue* = 1 one by one. At first, data object



(a) Initial phase of NN algorithm **(b)** Recursive phase of NN algorithm

Figure 2.4: A example of NN algorithm

$P_8$is added into the skyline. After that data objects P_{10} is checked and added to the skyline as they it not dominated by the current skyline objects. After data objects P_4 and P_7 are checked, they are not added to the skyline as they are dominated by current skyline objects. After that, *maxValue* is updated to 9. Similarly, after processing P_3, P_9, and P_6, P_3 and $P_9$are added into skyline. *maxValue* is not changed after these two new skyline objects are found. Then, P_2 is examined and discarded as it is dominated by some current skyline objects (i.e., P_8). Index algorithm continues to examine the remaining batches in the increasing order of its *minValue*. When P_1 is processed, since its *minValue* = 9 is equal to *maxValue*, P_1 is discarded immediately without being evaluated with the current skyline objects. After all batches are processed, Index algorithm outputs the skyline $\{P_3, P_8, P_9, P_{10}\}$.

Nearest Neighbor Skyline

Nearest Neighbor algorithm (NN) [8] is based on the following fundamental observation.

Observation: *Given a dataset and a monotonic distance function f (e.g., Euclidean distance), the nearest neighbor of the origin is a skyline object.*

Based on the above observation, to compute the skyline for a given dataset, NN first finds the nearest neighbor P from the origin. Then, partition the data space into 3 parts with P (See Figure 2.4(a) for example):

Part 1: The hyper rectangle with the origin as lower-left corner and P as upper-right corner (i.e., R_1 shown in Figure 2.4(a)). Clearly, P is dominated by the data object in this region. According to the above observation, this part is empty as no data object dominates P.

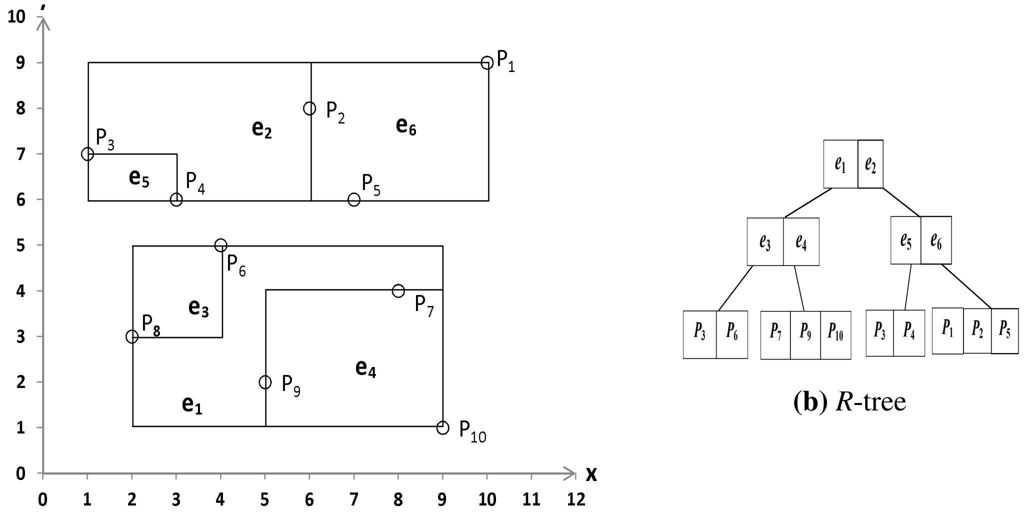
Part 2: The hyper rectangle with P as lower-left corner and the upper-right corner of the data space as upper-right corner (i.e., R_4). Obviously, all data objects in this part is dominated by P. Therefore, these data objects can be discarded safely.

Part 3: The other regions (i.e., R_2 and R_3). The property of those regions is that their local skyline objects belong to the global skyline as P does not dominate any data object in those regions. As a result, NN is recursively applied to these regions till all the data space is evaluated.

Take the same dataset shown in Table 2.8 for example. After the nearest neighbor of the origin, P_3, is found, the data space is partitioned into 4 parts, R_1, R_2, R_3, and R_4 (Figure 2.4(a)). As stated above, NN only needs to consider regions R_2 and R_3. For each region, NN recursively finds its nearest neighbor to its lower-left corner and partitions it into sub-regions. After further partitioning R_3, we find P_8, as a skyline object and search process in this region is terminated. Similarly, in R_2, as shown in Figure 2.4(b), P_9 is found as the nearest neighbor and then it further partitions R_{22}. After the data object P_{10} is found as the nearest neighbor of its sub-region, NN ends and outputs P_3, P_8, P_9, and P_{10} as the skyline result.

Branch and Bound Skyline

Like NN algorithm, Branch and Bound Skyline algorithm (BBS) [4, 9] is also based on nearest neighbor search. Assuming the dataset is indexed by an R-tree [59]. BBS is IO optimal i.e. the number of nodes of R-tree accessed is minimized. To find the skyline, BBS traverses the R-tree in a best-first manner: it always evaluates and expands the node that is closest to the origin among all unvisited nodes. To do that, BBS employs a heap in which the key of each entry (i.e., R-tree node or data object) is its minimum distance to the origin. Here, the minimum distance of an R-tree node to the origin is the summation of the coordinates of its lower-left corner. Initially, all child entries of the root node of the R-tree are inserted into heap. In each iteration, the top entry e is removed from the heap and examined against the skyline computed so far. If e is dominated by some current skyline object, e is discarded. Otherwise, e is either expanded or output as a skyline object based on its types. If e is an R-tree node, it



(b) *R*-tree

(a) Dataset

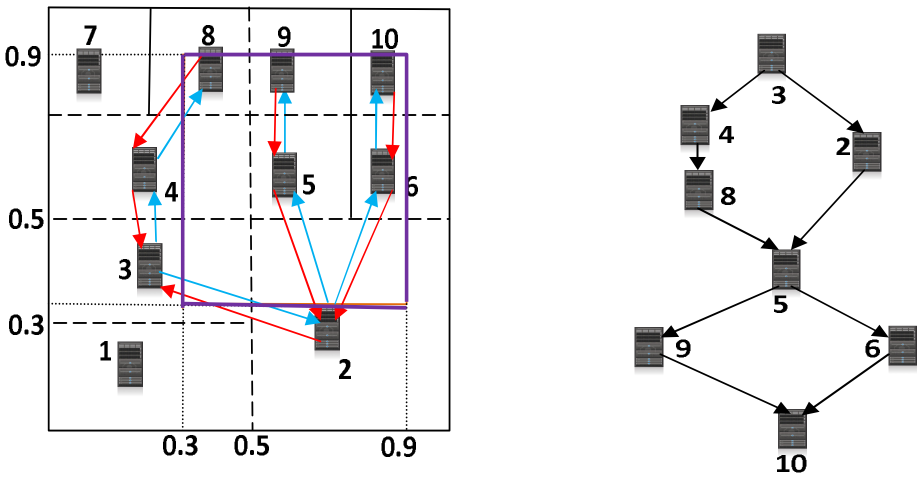
Figure 2.5: A example database and its *R*-tree

is expanded by inserting all its child entries which are not dominated by any current skyline objects into the heap. If *e* is a data object, it is output as a new skyline object. BBS ends when the heap is empty. In order to speed up examination whether the entry *e* in question is dominated by the current skyline objects, the current skyline is maintained by an in-memory *R*-tree as well.

Consider the dataset as given in Figure 2.5(a), the corresponding *R*-tree is illustrated in Figure 2.5(b). To compute skyline, as listed in Table 2.12, BBS first inserts entries e_1 and e_2 into the heap. Then, e_1, which is closer to origin than e_2, is expanded to entries e_3 and e_4. The next top entry is e_3 and its data objects P_8 and P_6 are inserted

Table 2.12: An example of BBS algorithm

Action	Heap Content	Skyline
initializing	$(e_1, 3), (e_2, 7)$	$\oslash$
expand e_1	$(e_3, 5), (e_4, 6), (e_2, 7)$	$\oslash$
expand e_3	$(P_8, 5), (e_4, 6), (e_2, 7), (P_6, 9)$	P_8
expand e_4	$(P_9, 7), (e_2, 7), (P_6, 9), (P_{10}, 10), (P_7, 12)$	P_8, P_9
expand e_2	$(e_5, 7), (P_6, 9), (P_{10}, 10), (P_7, 12)$	P_8, P_9
expand e_5	$(P_3, 8), (P_6, 9), (P_{10}, 10), (P_7, 12)$	P_8, P_9, P_3
examine P_6, P_{10}, P_7	$\oslash$	P_8, P_9, P_3, O_{10}



(a) CAN multicast-based Method with in-network pruning

(b) CAN multicast hierarchy

Figure 2.6: DSL [10] working process

into the heap after processing. After that, P_8 is processed. As it is not dominated by any current skyline object (the current skyline is empty), it is output as a new skyline object. Similarly, entry e_4 is expanded and its data objects are inserted into the heap. When e_2 is expanded, its child entry e_6 is found being dominated by one skyline object (i.e., P_8). So, it is discarded. With the same reason, after e_5 is processed, only data object P_3 is inserted into heap. Then, BBS continues to examine data objects remained in the heap one by one till all are processed. Only the data objects which are not dominated by the current skyline objects are output as new skyline objects. Finally, the skyline result is $\{P_8, P_9, P_3, P_{10}\}$.

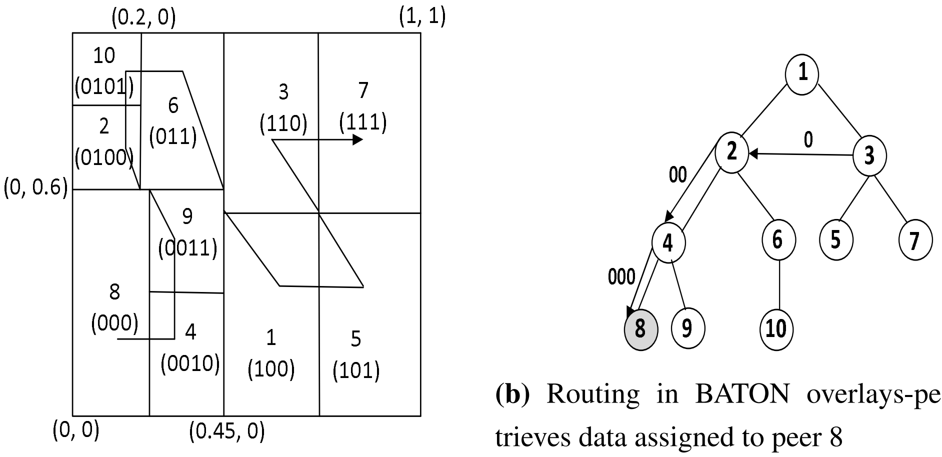
2.2.2 Skyline in Distributed Environment

As nowadays data are increasingly stored and processed in a distributed way, skyline processing over distributed data has attracted much attention recently. Below we give an overview of the existing approaches for skyline query processing in distributed environments, where each server stores a fraction of the available data. We have adapted the contents of this section from [25].

DSL

Distributed Skyline (DSL) proposed by Wu et al. [10]. DSL is used to compute constrained skyline queries in CAN networks [65]. Using CAN overlay, DSL maps data to regions and assign these regions to peers. Based on the constraints peers that do not contribute data are pruned immediately. Then an ad hoc multicast multi-level hierarchical tree is constructed with all remaining peers. Each queried peer then computes the skyline set on its local data, and intermediate peers merge result sets from their children. Figure 2.6 shows an example of a hierarchy built at runtime. The query is propagated along the edges of the hierarchy; peers perform local computation and propagate the results back on the same paths-on the way back, results are aggregated exploiting skyline additivity.

At the time of query processing, DSL builds a multicast hierarchy. In the hierarchy the peer that is responsible for the region containing the lower left corner of the constraint is the root. The data points stored at this peer are guaranteed to belong to the global skyline set because these data points cannot be dominated by points stored at any other peer. Moreover, the hierarchy is built in such away that peers whose data points cannot dominate each other are queried in parallel. In general, any peer in the CAN overlay can decide whether its local skyline points are in the global skyline set by only consulting a subset of other peers. In DSL, the hierarchy is built dynamically and each queried peer decides which neighboring peers should be queried next by using dynamic region partitioning and encoding. Thus, a peer that receives a query along with the local result set first waits to receive the local skyline sets from all neighboring peers that precede it in the hierarchy. Then, it computes the skyline set based on its local data and the received data points. Thereafter, the local skyline points are forwarded to the peers responsible for neighboring regions, in such a way that only peers whose data points cannot dominate each other are queried in parallel. In addition, neighboring peers that are dominated by the local skyline points are not queried because they cannot contribute to the global skyline set. Finally, all local result sets are collected at a peer that cannot forward the query any further, and the global skyline set is reported back to the query initiator.



(a) Peers and their assigned regions

(b) Routing in BATON overlays-peer 3 retrieves data assigned to peer 8

Figure 2.7: Skyframe example [12]

SSP and skyframe

Wang et al. [11] introduced an approach SSP (Skyline Space Partitioning) for distributed processing of skyline queries in BATON [66] networks. Peers in BATON are organized in a balanced binary tree structured overlay network, where each peer is responsible for a region in data space. Techniques for splitting and merging allow for load balancing among peers. By dynamically sampling load from random peers, load imbalance can be detected and data may be migrated to other peers in order to counteract the imbalance. As BATON networks have originally been designed for one-dimensional data, Wang et al. map the multidimensional data space to one-dimensional keys using a Z-curve. Figure 2.7 shows an example of the mapping of the data regions to the peers in the BATON network. Regions in BATON are created by successively splitting existing regions into two parts with respect to a specific dimension, and each region is represented by a binary string that identifies the region and is consistent with the Z-order of the region. Each peer knows the split history (i.e., dimension, split value) that its region originates from. Based on this information, for a given region, the identifier is determined. In addition, the routing table of each peer contains links to other peers (parents, children, adjacent peers, and other peers on the same level), so that the query is efficiently routed to a particular peer.

Figure 2.7(b) illustrates the principle of routing in BATON overlays. Assume peer 3 needs to retrieve data contained in the region assigned to peer 8. The identifier of the responsible region starts with 0 because it is contained in the lower part of the first split (at 0.45 in the x dimension). Thus, peer 3 forwards the query to a known peer (peer 2) with a leading 0 in its assigned region. Using the same strategy, peer 2 forwards the query to peer 4, which again forwards the query to the peer holding the queried data, namely peer 8.

Skyline processing in BATON networks relies on identifying relevant regions and routing the query to peers responsible for those regions. More precisely, skyline computation starts at the peer p_{start} , which is the peer responsible for the region containing the origin of the data space. Peer p_{start} computes the local skyline points that are guaranteed to be in the global skyline set. Then, p_{start} selects the most dominating point p_{md} (i.e., the point dominating the largest region [15]), which is used to refine the search space and to prune dominated regions and therefore also the responsible peers from consideration. A peer can safely be pruned if the best point (i.e., the lower left corner) of its region is dominated by p_{md}. Then, the querying peer forwards the query to the peers that are not pruned and gathers their local skyline sets. Finally, the query initiator computes the global skyline set by discarding dominated local skyline points.

In [12], Wang et al. generalize SSP by proposing Skyframe. In more detail, Wang et al. [12] propose an alternative algorithm for skyline processing without the need to determine a peer p_{start} before query processing starts. Instead, the querying peer forwards the query to a set of peers called border peers. A peer that is responsible for a region with minimum value in at least one dimension is called border peer. In our example, peers 1, 2, 4, 5, 8, and 10 are the border peers. Once the initiator receives the local skyline results, it computes p_{md} and determines whether additional peers need to be queried. Then, the querying peer queries additional peers, if necessary, and gathers the local skyline results. When no further peers need to be queried, the query initiator computes the global skyline set. Wang et al. show in [12] that Skyframe is also applicable for CAN networks [65].

iSky

Chen et al. [13] proposed the *iSky* algorithm for skyline computation on structured peer-to-peer networks. Similar to Skyframe [11, 12], *iSky* relies on the BATON [66] overlay. However, *iSky* employs a different transformation, namely *iMinMax*, to assign data to BATON peers. In *iSky*, each multidimensional tuple is mapped to a one-dimensional value (*iMinMax* value). For mapping a multidimensional tuple to

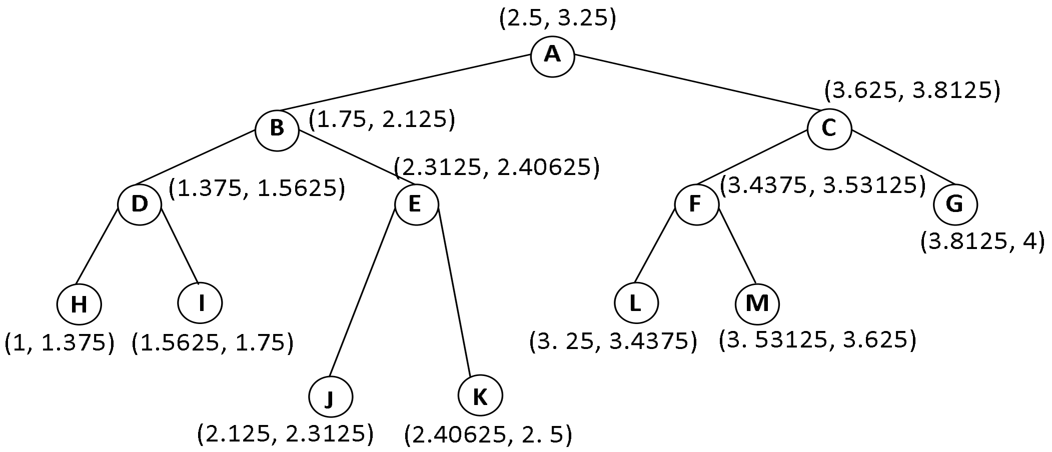


Figure 2.8: iSky example [13]: BATON network and iMinMax ranges assigned to peers

a one-dimensional value it first determines the maximum value for this tuple among all dimensions.

Assume that the range of each dimension is normalized into (0,1). Then *iMinMax* value is defined by the sum of (i) this maximum value and (ii) the number of the dimension it originates from. Each peer is responsible for a specific non-overlapping range of *iMinMax* values, so that each data point is assigned to a specific peer. Figure 2.8 shows an example of a BATON network and also for each peer the range of *iMinMax* values which each peer is responsible for. It should be noted that *iSky* assumes for each dimension, larger values are preferable.

Based on a skyline query, *iSky* first determines a set of initial skyline peers. These peers are chosen because points with maximum value in some dimension are guaranteed to be part of the global skyline set. Thereafter, the initial skyline peers are queried, and their local skyline results are merged into an initial set of skyline points by discarding dominated local skyline points. Then, the querying peer determines a threshold and a filter point. In order to define the threshold, the minimum value of all dimensions for each initial skyline point is computed. Then, the maximum value of all minimum values of all initial skyline points is selected as a threshold by using its range of *iMinMax* values. Any data point can be pruned if its maximum attribute value in all dimensions is smaller or equal to the received threshold. In addition, the most dominating point (i.e., the point dominating the largest region [15]) is chosen as filter point. The threshold and the filter point are attached to the query, which is forwarded to any neighboring peer that stores interval values larger than the threshold. When a peer receives a query, it first uses the threshold to check whether all its data are pruned. In this case, the peer just forwards the query to its neighboring peers that have interval values larger than the threshold. Otherwise, the peer processes the query on its local data and uses the filter point to discard dominated tuples. Finally, each peer refines the threshold and the filter point before forwarding the query and immediately sends the local result set to the query initiator, which then merges the local results and obtains the global skyline set after all queried peers have processed the query.

SSW

SSW (Semantic Small World) was proposed by Li et al. [14]. It is based on an underlying semantic overlay network. In such a network, the multidimensional data space is partitioned into non-overlapping regions, also known as clusters. Peers are assigned to the non-overlapping regions based on semantic labels. The semantic label of a peer corresponds to the region, which contains the centroid of its largest data cluster. For the data not contained in the region corresponding to the semantic label of a peer, foreign indexes are created at peers. A peer's foreign index holds information about data contained in its region that is stored at remote peers assigned to other regions. Each peer maintains links to other peers assigned to the same cluster and links to at least one peer in each neighboring cluster.

The computation of a skyline starts from the region that is guaranteed to contain skyline points. Then, the skyline query is evaluated over the data provided by peers of this cluster. The local skyline point that corresponds to the nearest neighbor of the origin is used as a filter point. In more detail, all regions that are entirely dominated by the filter point are not considered for further processing. After querying the remaining regions, reporting back all local result sets to the query initiator, and checking for mutual dominance, the global skyline set is returned to the user.

Apart from this algorithm, Li et al. [14] also propose an approximate algorithm, which does not require a semantic overlay network. Still, peers have semantic labels that are used to process the skyline query. As a peer is assumed to know the semantic labels of its neighbors, it forwards the query to the neighbor with the best semantic label, which is defined as the semantic label closer to the origin. Once a peer cannot find a neighbor with a better semantic label than its own semantic label, skyline computation ends and the result is returned to the user. Another variant of this strategy (multipath) forwards the query for each dimension to the peer that provides the best

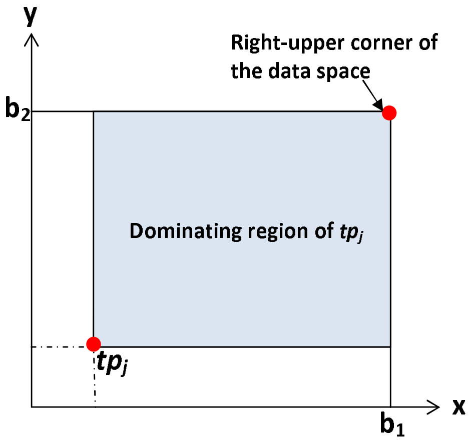

y
Right-upper corner of the data space
b_2
Dominating region of tp_j
tp_j
x
b_1

Figure 2.9: SFP example [15]: dominating region of point tp_j

data, if only this dimension is considered. The peer that initiates the query coordinates the computation and might optionally issue a stop command when a certain number of peers have been queried.

Single Filtering Point (SFP)

Huang et al. [15] assume a setting with mobile devices communicating via an ad hoc network (MANETs) and study skyline queries that involve spatial constraints.

The main feature of SFP is that it uses a point that belongs to the local skyline set as a filter to discard local skyline points of other peers. The the filter point is selected based on the volume of the dominating region (Figure 2.9). The dominating region is the area in data space that is dominated by a skyline point. If we consider a uniform distribution, a larger dominating region means that there is a higher probability to dominate other points. After receiving a query request by a peer, the peer first performs local query processing. It then propagates the query to its neighboring peers by attaching a filter point to the query. The filter point is used to discard local skyline points before sending back the local result to the query initiator. Each peer updates the filter point if a local skyline point has a larger dominating region.

DDS

Hose et al. [16, 17] proposed skyline processing based on DDS (Distributed Data Summaries)in unstructured P2P networks that uses routing indexes to identify relevant peers. Distributed data summaries are summaries of the data accessible via a peer's neighbors. This summary in a neighbor not only contains its local data but also the data of peers that are located several hops away but reachable via the neighbor. Each peer in the network is assumed to hold such summaries for its neighbors. Hose et al. consider two variants of data summaries, one based on multidimensional histograms and the other one based on the *QTree*, a combination of *R*-trees and multidimensional histograms.

Processing skyline queries based on DDS in works as follows: First, the query initiator computes the skyline set based on its local data. Then, based on its data summaries, it decides the relevancy of its neighbors. This helps the initiator to decide which neighbors can be pruned. The neighbors those provide only data that are dominated by local skyline points of the query initiator can be pruned without any further processing. A data summary based on histograms can be regarded as a set of regions represented by rectangles. Given a skyline query and a rectangular region, then the best point that might be contained with respect to domination is the rectangle's lower left corner. The best point dominates all data points possibly contained in the region. If the best point is dominated by a local skyline point, then the region can be pruned. If all regions that summarize data of a specific neighbor are pruned, then the query is not forwarded to this neighbor. Otherwise, the query is forwarded and the local skyline points are also forwarded to the neighboring peer in order to prune its neighbors. To minimize load at the initiator, local skyline points are routed on the same path the query was propagated on; a peer merges the local result sets received from its neighbors with its own local skyline set, checks for mutual dominance, and sends the obtained result to the peer that it received the query from.

DDS also supports approximate skyline queries mentioned as relaxed skylines. A relaxed skyline query aims at reducing the computational costs of skyline processing by representing regions of a peer's data by a single local skyline point. A region describing the data of a neighboring peer is represented by only one local skyline point if any point of the region has a distance to the representative point less than a given threshold, so a neighbor is pruned if all the regions describing its data are either dominated or represented by representatives, i.e., local skyline points. Thus, the query result set does not contain all skyline points, but a subset of skyline points and additionally representative data points that represent regions that are nearby and possibly contain further skyline points.

SKYPEER and SKYPEER+

Vlachou et al. [18] proposed SKYPEER, a distributed framework for efficient computation of subspace skyline processing over a super-peer architecture. In this regard, the notion of domination is extended by defining the extended skyline set, which contains all data points that are sufficient to answer a skyline query in any arbitrary subspace. Each super-peer pre-computes and stores extended skyline sets of its associated peers. When a super-peer receives a subspace skyline query, SKYPEER propagates the query to all super-peers and gathers the local skyline sets. In order to facilitate pruning of dominated data across the peers, SKYPEER also utilizes an efficient thresholding mechanism. In order to handle threshold-based query processing, data are transformed into one-dimensional values. Then, during query processing, a threshold value is defined based on already computed subspace skyline points. Then, the threshold is attached to the query before it is propagated in the network. Vlachou et al. explore different strategies for (i) threshold propagation and (ii) result merging over the P2P network aiming to reduce both computational time and volume of transmitted data. For threshold propagation, they considered two strategies, namely fixed threshold and refined threshold. In case of fixed threshold, the query initiator sets the threshold. On the other hand, in case of refined threshold each super-peer can update the threshold based on its local result. For merging results, they employed two strategies, namely merging the local result sets by the query initiator or progressive merging by intermediate super-peers.

SKYPEER was then extended in [19] leading to SKYPEER+. The goal of SKYPEER+ algorithm is to reduce the number of contacted super-peer during skyline queries. For achieving this goal, SKYPEER+ establishes a routing mechanism in order to contact only those super-peers that may contribute to the global skyline set. It works as follows:

In the preprocessing phase, each super-peer additionally applies a clustering algorithm on its locally stored extended skyline set. Then, the extended skyline set is stored based on the one-dimensional mapping, as depicted in Figure 2.10. The clusters are represented by MBRs and each point is mapped to a one-dimensional value, while all points that belong to the same dashed line have the same one-dimensional value. Then, based on the threshold employed by SKYPEER+, point p prunes the shadowed area. The cluster descriptions are broadcast over the super-peer network. Each super-peer

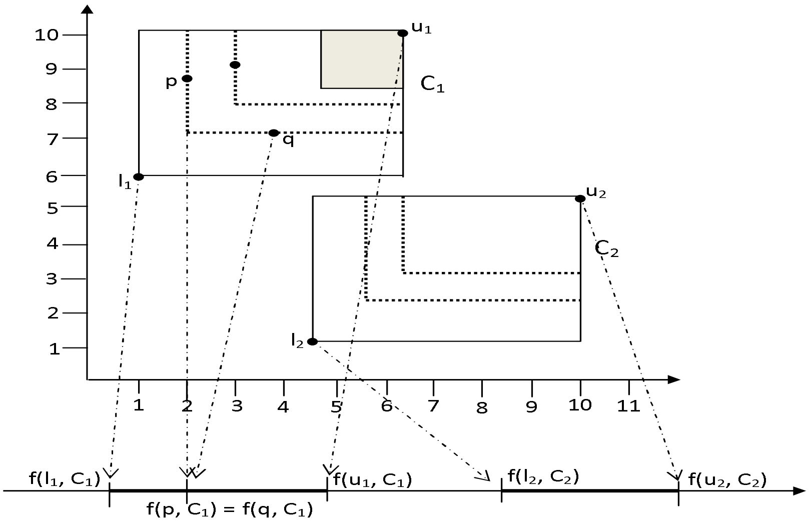
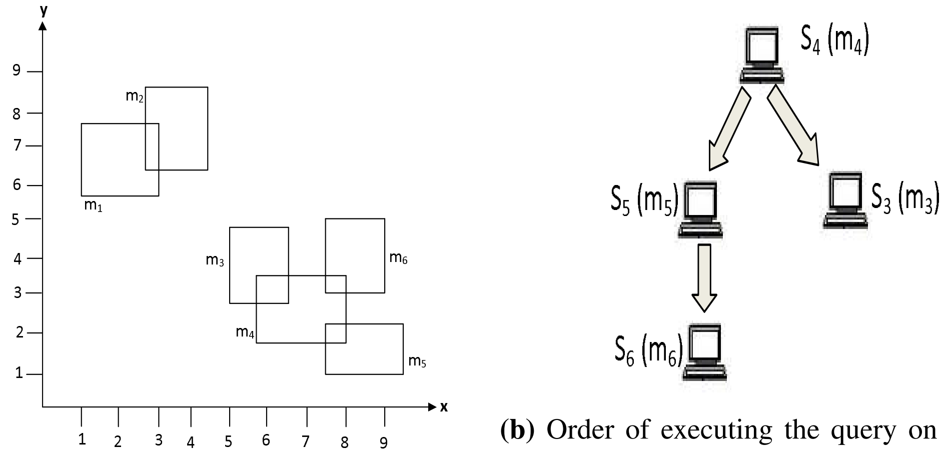


Figure 2.10: SKYPEER+ example [19]: Indexing of extended skyline points with one-dimensional mapping

collects the cluster information of all super-peers and builds routing indexes based on them. The one-dimensional mapping is combined with the clustering information, and a novel indexing technique is proposed for building the routing indexes, which support efficiently the thresholding scheme of SKYPEER. During query processing, the routing indexes are used to propagate the query only to network paths with super-peers storing data points that may contribute to the skyline set. In addition, the routing information is used to refine the threshold. Therefore, SKYPEER+ further improves the thresholding scheme and significantly reduces the amount of transferred data.

BITPEER

Fotiadou et al. [20] proposed BITPEER for subspace skyline queries over a super-peer architecture. Similar to SKYPEER, each super-peer stores the extended skyline of its peers. Differently to SKYPEER, Fotiadou et al. focus on distributed skyline computation based on BITMAP [7]. Therefore, BITPEER uses a bitmap representation that summarizes all extended skyline points. Given a subspace skyline query, the query is flooded in the super-peer network, and local results are sent back to the querying super-peer by using progressive merging at intermediate super-peers. The authors also discuss caching of subspace skyline points and continuous skyline queries. In order to enable the reusability of subspace skyline results, during query processing, the query-



(a) Incomparable groups of MBRs

(b) Order of executing the query on different peers

Figure 2.11: PaDSkyline example [21]

ing super-peer gathers the extended subspace skyline [18] instead of the subspace skyline. Therefore, BITPEER is able to use a cached skyline set also for subspace skyline queries that refer to a subspace of the query in the cache.

PaDSkyline

Cui et al. [21] proposed PaDSkyline (Parallel Distributed Skyline query processing) algorithm. The main idea of PaDSkyline is to determine the peers those can process the query in parallel under the assumption that the data points of each peer lie only in a part of the data space.

The querying peer first gathers a set of minimum bounding regions (MBRs) from each peer that summarizes the data stored at each peer. Subsequently, the querying peer processes the collected MBRs and groups them into one or more incomparable groups such that any data point summarized by an MBR of one group cannot be dominated or dominate any data point captured by an MBR of another group. Figure 2.11(a) depicts a set of MBRs gathered by the querying peer. We can see that there are two incomparable groups, namely MBRs m_1 and m_2 form the first group, while the second group consists of the remaining MBRs. These two incomparable groups can be queried in parallel without requiring merging of local results. For each incomparable group, a specific plan is constructed with the aim at maximizing the gain achieved by the filter points and defines a beneficial order to query the peers. An example of an execution order is depicted in Figure 2.11(b). In general, dominated MBRs are discarded, while MBRs (for example m_6) that are partially dominated (for example by m_5) are executed after the partially dominating MBR. For each group, the plan is sent to the peer (head group) responsible for the head MBR of the plan. The peer that receives a plan processes the query locally. Once a peer has processed the query, it removes itself from the query plan and forwards the query to the next peer indicated by the plan. In order to reduce network traffic, each peer attaches a set of K filter points to the query for discarding local skyline points of the peers that belong to the same group. The goal is to select as filter points the local skyline points that are more likely to dominate many other points. Two different strategies are studied. The first strategy is to select the K points with the largest volume of their dominating region [15]. The proposed alternative is to pick the K points with the maximal distance between them. The aim of this strategy is to minimize the overlap between the dominating regions of different points. After the peer has processed the query locally, the results are sent back directly to the head group, and after discarding all dominated points, the results are sent back to the querying peer.

AGiDS

Rocha-Junior et al. [22] propose a grid-based approach for distributed skyline processing (AGiDS), which shares assumptions similar to [21]. Differently, AGiDS assumes that each peer maintains a grid-based data summary structure for describing its data distribution.

AGiDS assumes that all peers share common cell boundaries for the grid structure that leads to non-overlapping cells, which increases the probability of domination between cells and enables efficient merging of local skyline set. The set of cells of a peer that contain at least one data point and that are not dominated by other cells is called region-skyline set of the peer. Only these cells of the grid contain data that belong to the local skyline set. At query time, the query initiator first contacts all peers and gathers the region-skyline sets of all peers. Then, the query initiator merges the collected cells into a new region-skyline set by discarding dominated cells.

An example of this process is depicted in Figure 2.12. Finally, queries are forwarded only to peers that correspond to at least one cell in the region-skyline set. The query initiator requests only a subset of local skyline points, namely those that belong to the cells of the region-skyline set. After having gathered all relevant points, the

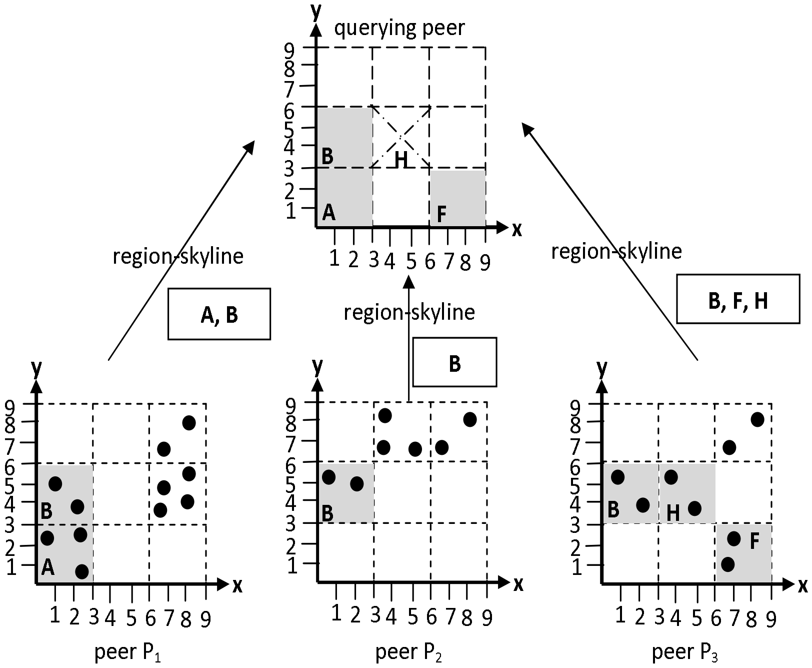


Figure 2.12: AGiDS example [22]: finding the non-dominating regions of different peers

querying peer computes the global skyline set by testing only the necessary regions for dominance.

FDS

Zhu et al. [23] propose a feedback-based distributed skyline (FDS) algorithm, which assumes no particular overlay network. FDS aims at minimizing the network bandwidth consumption, measured in the number of tuples transmitted over the network. FDS requires a scoring function that is used by all peers, and each query is processed in multiple round trips. In each round trip, all peers send to the querying peer k local skyline points with the lowest score based on the scoring function. Then, the querying peer computes the maximum score of all transferred local skyline points and requests from all peers the remaining local skyline points that have scores smaller than the maximum score. Finally, the querying peer merges the local result sets and selects a subset of the current skyline points as a feedback that is sent to all peers. Peers receiving the feedback remove from their local data points all points that are dominated by the points of the feedback.

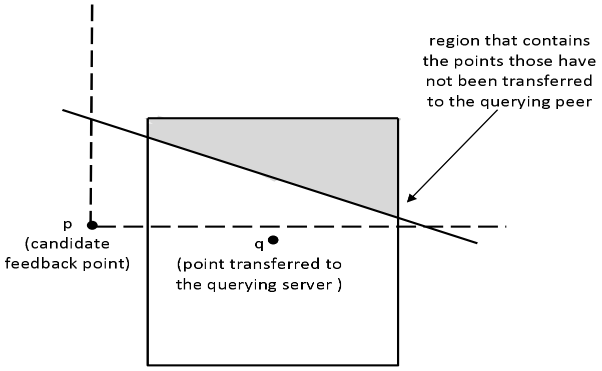
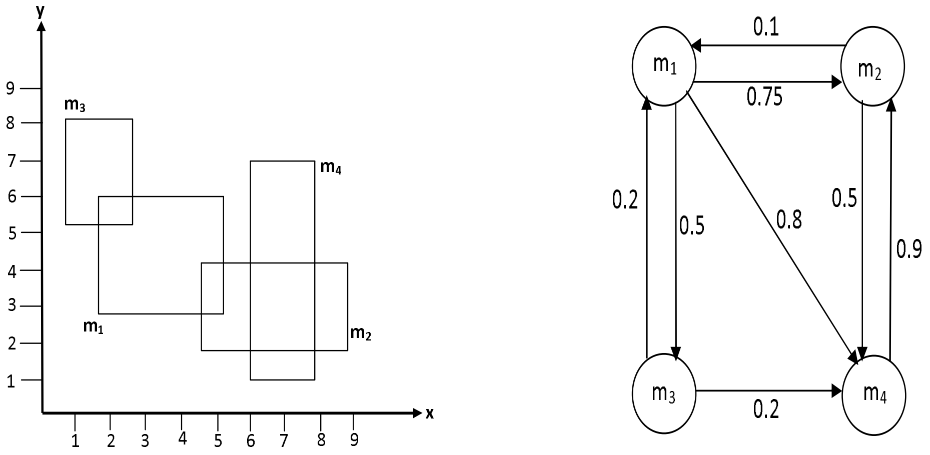


Figure 2.13: FDS example [23]: example of the feedback algorithm

In the feedback phase, FDS selects filter points for each peer; these are skyline points that are guaranteed to dominate at least l local data points. To this end, for each local skyline point, the distance of the l-nearest neighbor is computed and attached to it before sending it to the querying peer. The distance is combined with the score of the scoring function in such a way that FDS can decide whether a skyline point satisfies the condition. An example of the feedback algorithm is depicted in Figure 2.13. The depicted rectangle is defined by the distance of the l-nearest neighbor based on L_∞. The score of the scoring function used for sorting the data points defines the region that encloses the points that have not been transferred to the querying peer. Then, if the dominating region of a skyline point covers this region, it will dominate at least l points. FDS is efficient in terms of bandwidth consumption. However, several round trips are required to compute the skyline set. Thus, it may incur high response time.

SkyPlan

For improving the performance of PaDSkyline [21], SkyPlan was proposed in [24] . Like PaDSkyline, during query processing, each peer reports a set of minimum bounding rectangles (MBRs) to the querying peer as a summarization of its data. SkyPlan defines the order the query is executed that improves the performance of skyline query processing. In SkyPlan, it is possible that some peers are not contacted at all. This situations happens if all points of a peer are dominated by a point stored locally at another peer. Moreover, there is a significant reduction in data transfer. However, if the filter points fail to prune any point of a peer, then there is no gain from querying



(a) Example of MBRs **(b)** Constructed graph

Figure 2.14: SkyPlan example [24]

the peers consecutively. In this case, the parallelism should be preserved, in order to minimize the latency and therefore also the response time.

In SkyPlan, a weighted directed graph (*SD*-graph) is created by the query originator that can represent the dominance relationships between the collected MBRs. Each vertex of the graph represents a non-dominated MBR, while an edge between two vertices means that one MBR dominates partially the other MBR. The weights on the graph represent the pruning power those are used to quantify the potential gain through filtering. For example, consider the MBRs and the graph depicted in Figure 2.14. MBR m_1 partially dominates m_2 because the lower left corner of m_1 dominates the upper right corner of m_2. Thus, a directed edge from m_1 to m_2 is added to the graph. Before executing the query, SkyPlan transforms the *SD*-graph into an execution plan (one or more directed trees) that maximizes the total pruning power while preserving the parallelism when no significant gain can be obtained from processing the queries on different peers consecutively. It has been shown that SkyPlan supports also multi-objective executions plans, in case that additional objectives need to be fulfilled simultaneously, such as additionally restricting the number of hops that the query is forwarded.

Finally, the distributed skyline query is processed based on the execution plan. The querying peer sends the query to the root of every directed tree in the execution plan. Each queried peer processes the skyline query locally, refines the execution plan, and selects a set of filter points. The refinement of the execution plan produces a new

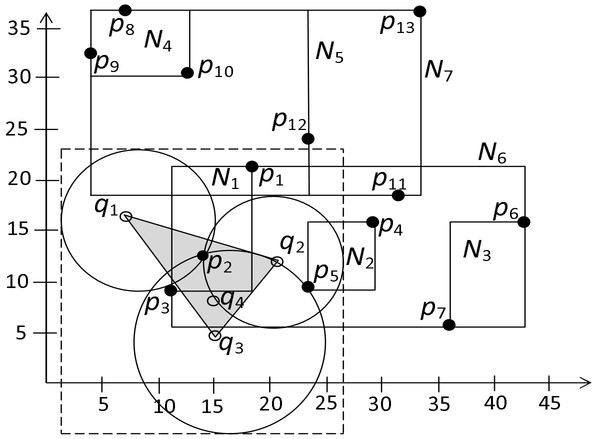


Figure 2.15: B^2S^2 [26]: Example of data points and query points

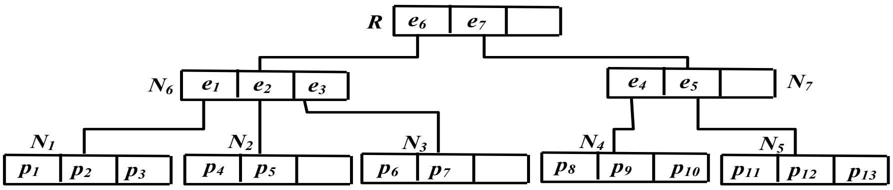


Figure 2.16: B^2S^2 example [26]: Data points indexed by an R-tree

execution plan that does not contain the MBRs that are dominated by the local skyline points. The filter points are selected based on the dominating region [15]. Since the volume of the dominating region does not necessarily relate to the area within the MBR that is dominated by a filter point, SkyPlan takes into account the MBRs of the execution plan while selecting the filter points. Eventually, each peer gathers the local result sets of the peers that had received the query through and merges them by discarding dominated points. Local processing on peers terminates by returning the merged skyline points to the previous peer based on the execution plan

2.2.3 Spatial Skyline Computation

Given a set of data points P and a set of query points Q spatial skyline queries retrieve all data points from P those are not dominated with respect to query points in Q. Below, we provide a brief review of well known spatial skyline query algorithms.

Table 2.13: B^2S^2 for the example of Figure 2.15

Step	**Heap Contents (entry e, $mindist(.,.)$)**	$S(Q)$
1	$(e_6, 8)$, $(e_7, 52)$	$\oslash$
2	$(e_1, 18)$, $(e_2, 49)$, $(e_7, 52)$, $(e_3, 115)$	$\oslash$
3	$(p_2, 38)$, $(p_3, 42)$, $(e_2, 49)$, $(e_7, 52)$, $(p_1, 70)$, $(e_3, 115)$	$\oslash$
4	$(e_7, 52)$, $(p_5, 53)$, $(p_1, 70)$, $(e_3, 115)$	p_2, p_3
5	$(p_5, 53)$, $(p_1, 70)$, $(e_3, 115)$	p_2, p_3
6	$(p_1, 70)$, $(e_3, 115)$	p_2, p_3, p_5

B^2S^2: Branch-and-Bound Spatial Skyline Algorithm

B^2S^2 algorithm was introduced by Sharifzadeh et al. [26]. It is an improved customization of BBS [4, 9] algorithm. In B^2S^2 algorithm, data points are indexed by a data partitioning method such as R-tree. Let us consider that $mindist(p,A)$ 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 rectangle e and the points of Q.

Now, consider the set of data points $P = \{p_1, p_2, \cdots, p_{13}\}$ and set of query points $Q = \{q_1, q_2, q_3, q_4\}$ as shown in Figure 2.15. The indexing of the data points in P by an R-tree is shown in Figure 2.16.

B^2S^2 computes spatial skyline points as follows.

At first B^2S^2 computes the convex hull of Q and determines the set of its vertices $CH_v(Q)$. In our running example vertices on the convex hull are q_1, q_2 and q_3. i.e., $CH_v(Q) = \{q_1, q_2, q_3\}$. Subsequently, B^2S^2 begins to traverse the R-tree from its root R down to the leaves. It maintains a minheap H sorted based on the $mindist$ values of the visited nodes. Table 2.13 shows the contents of H at each step of the computation procedure. First, B^2S^2 inserts $(e_6, mindist(e_6, CH_v(Q)))$ and $(e_7, mindist(e_7, CH_v(Q)))$ corresponding to the entries of the root R into H. Then, e_6 with the minimum $mindist$ is removed from H and its children e_1, e_2, and e_3 together with their $mindist$ values are inserted into H. Similarly, e_1 is removed and the children of e_1 are added to H. In the next iteration, the first entry p_2 is inside $CH(Q)$ and hence is added to $S(Q)$ as the first skyline point.

After obtaining the first skyline point, any entry e must be checked for dominance before insertion into and after removal from H. If e is dominated by any skyline point in $S(Q)$, B^2S^2 discards e. B^2S^2 applies two simple tests to determine the dominance of

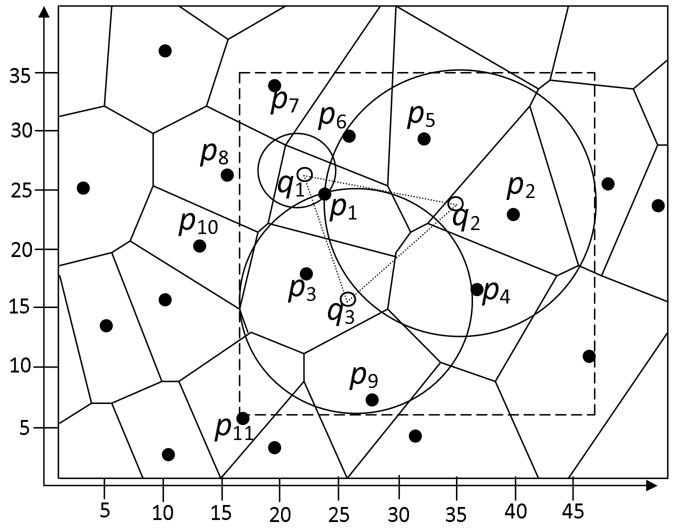


Figure 2.17: VS^2 [26]: Example points for VS^2

e. (1)If e does not intersect with the intersection of all MBRs each corresponding to a current skyline points in $S(Q)$, e is dominated by a point in $S(Q)$. (2) If e is completely inside the convex hull $CH(Q)$, e is not dominated. If e does not pass either of the above tests, B^2S^2 requires to check e against the entire $S(Q)$. B^2S^2 maintains a rectangle B corresponding to the intersection area and updates it when a new skyline point is found (see dotted box in Figure 2.15.

Considering the example of Figure 2.15, B^2S^2 removes p_3 from H. As p_3 is inside B and is not dominated by the current skyline point p_2, p_3 is added to the skyline points and the rectangle B is updated. The next step examines e_2 which is not dominated by current skyline points p_2 and p_3. Among e_2's children, p_5 is inserted to H and p_4 is is discarded as p_4 is dominated by p_3. Then, B^2S^2 removes e_7 and extracts its children e_4 and e_5 as e_7 is not dominated. Here, we can see that e_4 does not intersect with B and e_5 is dominated by p_2. Hence, B^2S^2 discards both entries. At this step, p_5 is removed and added to the skyline points. The remaining steps discard both dominated entries p_1 and e_3. Finally, we obtain the points p_2, p_3, and p_5 as the final spatial skyline result.

VS^2: Voronoi-based Spatial Skyline Algorithm

VS^2 algorithm was also proposed [26]. Instead of R-tree data structure, VS^2 algorithm utilizes the Voronoi diagram (i.e., the corresponding Delaunay graph) of the data points to answer spatial skyline query. In VS^2, the adjacency list of the Delaunay graph of the

Table 2.14: VS^2 for the example of Figure 2.17

Step	Heap Contents (entry e, $mindist(.,.)$)	$S(Q)$
1	$(p_1, 24)$	$\emptyset$
2	$(p_1, 24)$, $(p_3, 28)$, $(p_6, 32)$, $(p_5, 34)$, $(p_4, 38)$, $(p_8, 44)$	$\emptyset$
3	$(p_3, 28)$, $(p_6, 32)$, $(p_5, 34)$, $(p_4, 38)$, $(p_8, 44)$, $(p_9, 49)$, $(p_10, 49)$, $(p_{11}, 63)$	p_1
4	$(p_6, 32)$, $(p_5, 34)$, $(p_4, 38)$, $(p_8, 44)$, $(p_7, 46)$, $(p_9, 49)$, $(p_10, 49)$, $(p_{11}, 63)$	p_1, p_3
5	$(p_5, 34)$, $(p_4, 38)$, $(p_8, 44)$, $(p_7, 46)$, $(p_9, 49)$, $(p_{10}, 49)$, $(p_{11}, 63)$	$p_2, p_3, p_6, p_5, p_4, p_2$
...	...	$p_2, p_3, p_6, p_5, p_4, p_2$

points in P is stored in a flat file. To preserve locality, points are organized in pages according to their Hilbert values.

VS^2 starts traversing the Delaunay graph from a data point closest to a query point q_i). The traversal order is determined by the monotone function $mindist(p,CH_v(Q))$. VS^2 maintains two different lists $Visited$ and $Extracted$ to track the traversal. $Visited$ contains all visited points and $Extracted$ contains those visited points whose Voronoi neighbors have also been visited. Similar to B^2S^2, VS^2 also maintains the rectangle B which includes all candidate skyline points.

For the explanation of VS^2 algorithm consider the example as shown in Figure 2.17.

In Figure 2.17, three query points q_1, q_2, and q_3 and the data points are shown as white and black dots, respectively. Like B^2S^2, VS^2 maintains a heap data structure. Table 2.14 shows the contents of heap H. First, VS^2 adds $(p_1,midist(p_1,CH_v(Q)))$ to H and marks p_1 as visited. B is also initialized accordingly to the dotted box as shown in Figure 2.17. The first iteration visits p_3, p_4, p_5, p_6, and p_8 as p_1's Voronoi neighbors and adds their corresponding entries to H. It also adds p_1 to the $Extracted$ list. The second iteration removes p_1 from H as p_1 and its neighbors have been already visited. It also adds p_1 to $S(Q)$ as p_1 is inside $CH(Q)$. The third iteration adds p_9, p_{10}, and p_{11} as the only unvisited neighbors of p_3. The next two iterations immediately remove p_3 and then p_6 from H and add them to $S(Q)$ as their neighbors have been already visited. The subsequent iterations add p_5, p_4, and p_2 to the skyline and eliminate the remaining entries of H as they are all dominated.

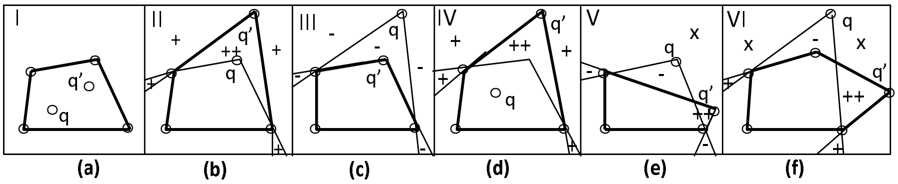


Figure 2.18: VCS^2 example [26]: Change patterns of convex hull of Q when the location of q changes to q'

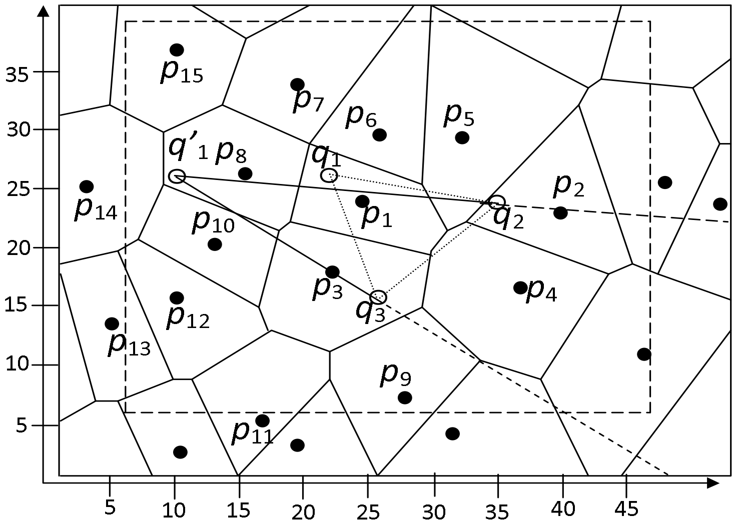


Figure 2.19: VCS^2 example [26]: Example points for VCS^2

VCS^2: Voronoi-based Continuous SSQ

For handling the change in locations of query points, Sharifzadeh et al. [26] introduced VCS^2 algorithm. The VCS^2 traverse traverses the Delaunay graph of the data points similar to VS^2. It first computes $CH(Q')$, the convex hull of the latest query set. Then, it compares $CH(Q')$ with the old convex hull $CH(Q)$. Depending on how $CH(Q')$ differs from $CH(Q)$, VCS^2 decides which specific portions of the graph is required to traverse if necessary and how to update the old skyline $S(Q)$.

It is observed that there are only six change in patterns of the query points as shown in Figure 2.18(a)- Figure 2.18(f). Each figure illustrates a case where the location of a query point q changes to q'. The grey and the thickedged polygons show $CH(Q)$ and $CH(Q')$, respectively. VCS^2 tries to recognize specific simple patterns and subsequently updates the skyline accordingly. In the patterns of Figure 2.18, the region labeled by "++" includes the points inside $CH(Q')$ and outside $CH(Q)$ and any point in this region is a skyline point with respect to Q' and hence must be added to the skyline. The points in the regions labeled by ”+” might be skyline points and must be examined. The regions labeled by ”-” contain the points whose dominator regions have been expanded and hence might be deleted from the old skyline. The points in the regions specified by ”” must be examined for inclusion in or exclusion from the skyline as their dominator regions have changed. Finally, neither q nor q' affects the dominance of the points in the unlabeled white region.

Now consider the explanation of VCS^2 using the example of Figure 2.19 where we can see that the query point q'_1 is the new location of q_1. First, VCS^2 computes the convex hull of $Q' = q'_1, q_2, q_3$ and compares it with $CH(Q)$. The change pattern matches pattern V in Figure 2.18. Therefore, the update to S(Q) involves both insertion and deletion. Then, VCS^2 initializes $S(Q')$ to the old $S(Q)$ resulted from applying VS^2. It also adds $(p_8, mindist(p_8, CH_v(Q')))$ into the heap as p_8 is the closest point to q'_1, the new location of q_1. The second iteration extracts p_8 and adds all of its Voronoi neighbors except p_1 into the heap as p_1 is not in the candidate region of pattern V of Figure 2.18 . Similarly, p_3 is extracted and all its neighbors except p_4 is added in added to the heap. Next, as p_3 is not dominated it remains in $S(Q')$. In the next two iterations p_8 and p_{10} are to $S(Q')$. The sixth iteration, visits p_6, the only unvisited Voronoi neighbor of p_7 in B which is subsequently removed from $S(Q')$ as it is dominated by $p_1 \in S(Q')$. The final four iterations of VCS^2 also eliminate remaining points from heap as they are all dominated. No point in the final skyline set is dominated and hence VCS^2 returns p_1, p_3, p_5, p_4, p_2, p_8, and p_{10} as the skyline result. Note that in this example VCS^2 avoids dominance checks for points such as p_4 outside the candidate region of the identified change pattern and p_1 inside the convex hull of query points.

Enhanced Spatial Skyline (ES)

ES [27] algorithm use R*-tree [56] to store the data points. ES algorithm computes the Voronoi diagram and the delaunay graph of the data points, and store them in the form of the file. To find the data point closest to a query point, ES computes the Voronoi cells intersecting the boundary of the query convex hull and finds all the Voronoi cells lying in the convex hull by traversing the delaunay graph. As it is necessary to look each Voronoi cell at most once during traversing the Delaunay graph of the data points, it can be done by reading it from the file when it is required and deallocate it from memory after passing it by.

In ES process, the region to search is restricted to the bounding box containing $|Q|$ circles for $|Q|$ query points. More precisely, the bounding box is set as the intersection of all bounding boxes defined by the skyline subset found so far. After that, we obtain a list of the candidates in this bounding box by using R*-tree. The list is sorted in ascending order of the candidates' distances to a query point and process them one by one in this order. When a new skyline point is found, the size of the size of the bounding box is reduced by taking the intersection of the current bounding box with the bounding box of this new skyline point. During the process, if some candidate point is not contained in the bounding box, we can simply skip the dominance test.

Direction-Based Spatial Skylines (DSS)

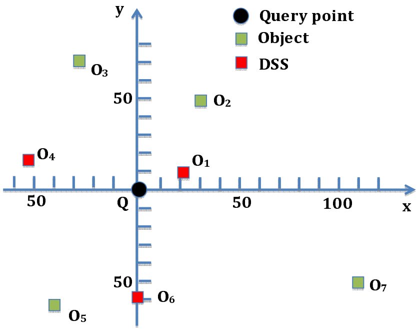
Direction-based spatial skylines (DSS) was proposed by Guo et al. [28] where they proposed a technique to retrieve nearest objects around the user from different directions. The DSS query compares objects in terms of two spatial properties distances and direction. The closer object is better than or dominates the further object if they are in the same direction. The objects that cannot be dominated by any other object are included in the direction-based spatial skyline(DSS). They proposed *snapshot DSS queries* to find spatial skyline objects for DSS query according to the user's current position. We can define Snapshot DSS Query as follows.

Definition 2.6 (Snapshot DSS Query [28]) *Given a DSS query* (Q, θ)*, an snapshot DSS query finds all the objects on the DSS.*

Snapshot DSS query computes spatial skyline objects based on the following two observations.

*O*bservation 1: (Limiting the scope to adjacent objects) The first observation is that when we determine whether an object is on the DSS, we do not need to check its dominance relationships with all the other objects. To explain the idea, they introduce the notion of adjacent objects. The objects are adjacent to each other if they are adjacent in the circular list sorted by the order of directions. When we check an object, we only need to consider the dominance relationships between its adjacent objects.

*O*bservation 2: (Early termination) The second observation is that we can finish the process by only checking a subset of the whole object set. This is based on the concept of partition angle [28]. The partition angle between two objects O_i and O_j, where O_j is the successor of O_i is defined by the equation $\varphi_{ij} = (\omega_j - \omega_i) \ mod \ 60^\circ$



object	distance to query point Q	the angle between O_i and Q
O_1	22	27°
O_2	50	53°
O_4	54	158°
O_6	60	270°
O_3	67	117°
O_5	72	236°
O_7	112	333°

Figure 2.20: Example of a DSS query ($\theta = \pi/3$) [28]

Based on this observation 1, the following property of DSS query has been derived.

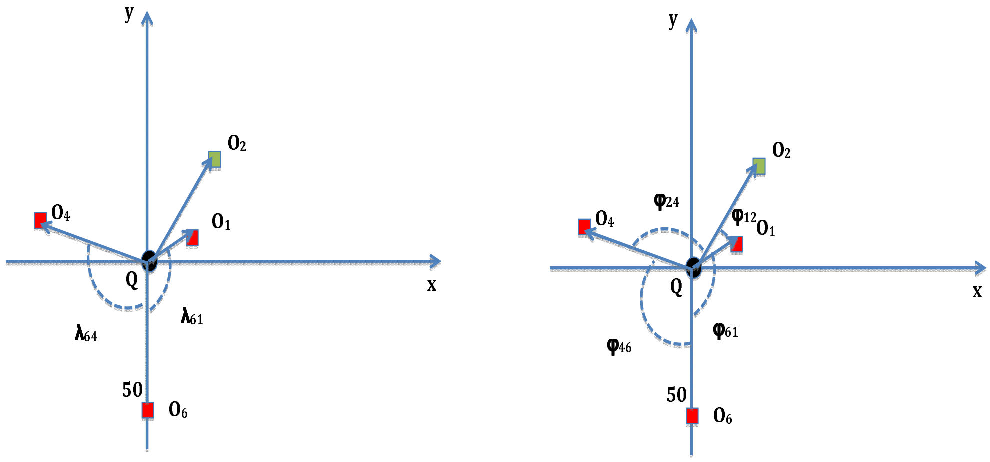
Property 1 [28]: Let the current target object be O_i and O_i's adjacent objects among the checked objects be O_j and O_k. If both of the included angles λ_{ij} and λ_{ik} satisfy the properties $\lambda_{ij} \geq \theta$ and $\lambda_{ik} \geq \theta$, O_i is a DSS object. Otherwise, p_i is not a DSS object.

Based on this observation 2, Property 2 has derived as follows.

Property 2 [28]: If all the partition angles for the checked objects are smaller than 2θ, the process can be terminated and we can say that we have obtained all the spatial skyline objects based on DSS.

Let us consider the computation of DSS objects based on the example of Figure 2.20 and **observation 1** and **observation 2** as shown in Figure 2.21 (a) and Figure 2.21 (b), respectively.

The starting object in Figure 2.20 is the nearest object O_1. Here the partition angle is $\varphi_{O_1O_1} = 2\pi$. Next, we check the second nearest object O_2. The object O_2 is not on the DSS as $\lambda_{O_1O_2} < \theta$. The partition angles are $\varphi_{O_1O_2} < 2\theta$ and $\varphi_{O_2O_1} > 2\theta$. Since the termination condition is not satisfied, the procedure continues. Then, we examine the third nearest object O_4. It is on the DSS due to $\lambda_{O_4O_2} > \theta$ and $\lambda_{O_4O_1} > \theta$. As the partition angles are $\varphi_{O_1O_2} < 2\theta$, $\varphi_{O_2O_4} < 2\theta$, and $\varphi_{O_4O_1} > 2\theta$, the procedure proceeds. Next, we check the fourth nearest object O_6. It is on the DSS because $\lambda_{O_4O_6} > \theta$



(a) Observation 1 **(b)** Observation 2

Figure 2.21: Two observations [28]

and $\lambda_{O_6O_1} > \theta$. The partition angles are $\varphi_{O_1O_2} < 2\theta$, $\varphi_{O_2O_4} < 2\theta$, $\varphi_{O_4O_6} < 2\theta$ and $\varphi_{O_6O_1} < 2\theta$. Thus, the procedure terminates and we have found out all the DSS objects $\{O_1, O_4, O_6\}$.

Guo et al. [28] also proposed a modified version of *Snapshot DSS query* to handle the situations where the positions of query point changes frequently.

Spatial Skyline Queries in Distributed Environment [29]

Yoon en al. [29] first considered the computation of distributed spatial skyline query. They introduced distributed spatial skyline query for wireless sensor networks. There distributed spatial skyline algorithm consists three different phases.

In the first phase, the algorithm computes query convex hull $CH(Q)$, which is the convex hull of query points Q and rendezvous point R, which is the centroid of $CH(Q)$ for partitioning. It then transmits the $< CH(Q), R >$ to each triangle home, $t \in T$, (T is the set of triangle homes) which is the node closest to each point in $CH(Q)$. In phase 2, the algorithm searches the internal skylines in each t, which are definite skylines located within or near $CH(Q)$. In phase 3, the algorithm searches external skylines, the skylines that are located outside of $CH(Q)$. Finally, it returns all spatial skyline objects.

Other Spatial Skyline Queries

All of spatial skyline queries discussed above depend on Euclidean distance. As a result they can not well applicable in road networks as in road networks there are many constraints. Considering this fact Son et al. [30] introduced a variation of spatial skyline queries known as *MSSQ* that is well applicable to road networks. *MSSQ* depends on Manhattan distance instead of Euclidean distance for Spatial Skyline computation.

There are several situations where we are interested about non existence of facilities of the same type. As for example, for selecting a location for establishing a park or a supermarket, it is more likely that the place where there is no such facilities will be the interesting location. Considering this fact, You et al. [31] introduced a spatial skyline query mechanism that can easily select such interesting places.

2.2.4 Other Variants of Skyline Queries

So far all the algorithms discussed above focus on computing skyline from numerical databases with a fixed set of dimensions. Below we discuss different variants of skyline queries as well as skyline queries in different domains.

Subspace Skyline

The recent papers on skyline computation in subspaces [2, 3]. Yuan et al. [2] proposed two methods to compute skylines in all the subspaces by traversing the lattice of subspaces either in a top-down or bottom-up manner. In the bottom- up approach, the skylines in a subspace are partly derived by merging the skylines from its child subspaces at the lower level. In the top-down approach, the sharing- partition-and-merge and sharing-parent property of the DC algorithm [1] is exploited to recursively enumerate the subspaces and compute their skylines from the top to bottom level, which turns out to be much more efficient than the bottom-up approach. Since we can get the skyline frequencies if the skylines in every subspace is available, we compare their top-down approach with our top-k method in the performance study.

Another study on computing skylines in subspaces is by Pei et al. [3]. They introduced a new concept called skyline group, every entry of which contains the skyline objects sharing the same values in a corresponding subspace collection. They also proposed an algorithm skyey, which visits all the subspaces along an enumeration tree, finds the skylines by sorting and creates a new skyline group if some new skyline objects are inserted into an old group. The skyline groups found are maintained in a quotient cube structure for queries on subspace skyline. Their study tries to answer where and why an object is part of skyline without any accompanying coincident objects. However, their scheme can not help to solve the skyline frequency problem since an object can be in exponential number of skyline groups in high dimensional space.

Top-k Frequent Skyline

A team of Chan, et al. [32] proposed a finding of top-k frequent skyline objects in multidimensional subspaces. The authors introduce the concept of the skyline frequency of a data object, which compares and ranks the interestingness of data objects based on how often they are returned in the skyline when different numbers of dimensions (i.e., subspaces) are considered. Intuitively, an object with a high skyline frequency is more interesting as it can be dominated on fewer combinations of the dimensions. Approximate algorithms are developed to address issues involved in high dimensional spaces, namely the dimensionality curse and the fact that frequent skyline computation requires that skylines be computed for each of an exponential number of subsets of the dimensions.

k-Dominant Skyline

In general, skyline query often retrieves too many objects to analyze. With the presence of a possibly large number of skyline objects, the full skyline may be less informative. To reduce the number of returned objects and to find more important and meaningful objects, Chan, et al. considered k-dominant skyline query [33], where we can control the selectivity by changing k. They relaxed the definition of domination so that an object is more likely to be dominated by another. Given an n-dimensional database, an object P_i is said to k-dominates another object P_j $(i \neq j)$ if there are k $(k \leq n)$ dimensions in which P_i is better than or equal to P_j . A k-dominant skyline object is an object that is not k-dominated by any other object. In Table 2.15, for example, if $k = 5$, the 5-dominant skyline query returns two objects: P_5 and P_7. Objects P_1, P_2, P_3, P_4, and P_6 are not in 5-dominant skyline because they are 5-dominated by P_7. The 4-dominant skyline query returns only one object, P_7, and the 3-dominant skyline query returns empty. Conventional skyline objects are n-dominated objects.

Table 2.15: Symbolic Dataset

ID	D_1	D_2	D_3	D_4	D_5	D_6
P_1	7	3	5	4	4	3
P_2	3	4	4	5	1	3
P_3	4	3	2	3	5	4
P_4	5	3	5	4	1	2
P_5	1	4	1	1	3	4
P_6	5	3	4	5	1	5
P_7	1	2	5	3	1	2

Table 2.16: Ordered Domination table

ID	D_1	D_2	D_3	D_4	D_5	D_6	DP	SUM	DC	IDX
P_7	$\underline{1}$	$\underline{2}$	5	3	$\underline{1}$	$\underline{2}$	4	14	3	P_5
P_5	$\underline{1}$	4	$\underline{1}$	$\underline{1}$	3	4	3	14	4	P_7
P_4	5	3	5	4	$\underline{1}$	$\underline{2}$	2	20	6	P_7
P_2	3	4	4	5	$\underline{1}$	3	1	20	5	P_7
P_6	5	3	4	5	$\underline{1}$	5	1	23	5	P_7
P_3	4	3	2	3	5	4	0	21	5	P_7
P_1	7	3	5	4	4	3	0	26	6	P_7

The k-dominant skyline has following property [33].

Any object in $Sky_{k-1}(DB)$ must be an object in $Sky_k(DB)$ for any k such that $1 < k \leq n$. Any object that is not in $Sky_k(DB)$ cannot be an object in $Sky_{k-1}(DB)$ for any k such that $1 < k \leq n$. Similarly, every object that belongs to the k-dominant skyline also belongs to the skyline, i.e., $Sky_k(DB) \subseteq Sky_n(DB)$.

However, computing k-dominant skyline queries using conventional algorithm [33] is time consuming. The algorithm proposed by Siddique et al. [34] can compute k-dominant skyline queries quickly. In order to compute k-dominant skyline efficiently, Siddique et al. [34] used the concept of domination power of each object. The framework of their computation procedure is explained below.

An object P_i is said to have δ-domination power if there are δ dimensions in which P_i is better than or equal to all other objects of DB. After obtaining the domination power of each object, the objects are sorted in descending order by domination power.

If their is a tie in domination power among several objects, sorting is done in ascending order of the sum value to break the tie. This order reflects how likely an object k-dominates other objects. Table 2.16 is the sorted object sequence of Table 2.15, in which the column "DP" is the domination power and the column "Sum" is the sum of all values. In the sequence, object P_7 has the highest domination power 4. Note that object P_7 dominates all objects lie below it in four attributes, D_1, D_2, D_5, and D_6. After computing the sorted object sequence, we compute dominated counter (DC) and dominant index (IDX). The dominated counter (DC) indicates the maximum number of dominated dimensions by another object in DB. The dominant index (IDX) is the strongest dominator. That means "IDX" keeps the record of the corresponding strongest dominator for each object.

$Sky_k(DB)$ is a set of objects whose DC is less than k. In Table 2.16, for example, according to the dominated counter, we can see that $Sky_6(DB) = \{P_7, P_5, P_2, P_6, P_3\}$, $Sky_5(DB) = \{P_7, P_5\}$, and $Sky_4(DB) = \{P_7\}$. Since there is no object whose DC value is less than 3, thus $Sky_3(DB) =$. Using this table, we can quickly answer the k-dominant skyline query for a given k.

Skyline Computation over Partial Order Domain

The skyline queries with partially-ordered attributes are different from traditional skyline evaluation algorithms. This is because such attributes lack a total ordering. Chan et al. [35] first address the problem of skyline query evaluation involving partially-ordered attribute domains. The basic idea of their approach is to (a) transform each partially-ordered attribute domain into two integer-domain attributes, (b) organize the transformed attributes in an existing indexing method, and compute the skyline answers via the index.

Based on the above framework, they proposed three algorithms: BBS+, SDC, and SDC+. BBS+ is a straightforward adaptation of BBS. Because of false positives, BBS+ is no longer progressive, i.e., it needs to find all skyline points before answers can be returned. The second scheme, SDC (Stratification by Dominance Classification) exploits the properties of domain mappings to avoid unnecessary dominance checkings. In particular, it organizes the data into two strata at runtime - points that are definitely in the skyline and those that may be false positives. In the third scheme, SDC+, the data is partitioned into two or more strata offline so that points at stratum i cannot dominate points at stratum *i*-1. In this way, skyline points obtained from stratum *i* - 1 can be returned before points in stratum *i* are examined.

There are also several considerations [36, 37, 38] about computing skyline queries from data with partial order attributes. Although each of these works employed a different technique, the main focus of all of these works is to convert the partial order domain to its equivalent total order domain efficiently and then computing skyline from the total order domains.

Skyline Computation over Road Networks

There are several works about skyline computation in road networks. Deng et al. [39] first proposes multi-source skyline query processing in road network. They proposed three different skyline query processing algorithms for the computation of skyline points. These algorithms are the Collaborative Expansion algorithm (CE), the Euclidean Distance Constraint algorithm (EDC) and the Lower Bound Constraint algorithm (LBC). CE is a straightforward algorithm using an underlying paradigm that identifies network skyline points by expanding the search space centered around each query point incrementally. EDC is an approach to control the search directions for network skyline points by using Euclidean skyline points as a guide. LBC is based on network nearest neighbor algorithms and uses a novel concept of path distance lower bound to minimize the cost of network distance computation. The LBC algorithm is an instance optimal algorithm [39] and it is more than four times more efficient than CE. In [40], Safar et al. considered nearest neighbour based approach for calculating skylines over road networks and claimed that their approach performs better than the approach presented in [39]. Huang et al. [41] proposed two distance-based skyline queries techniques those can efficiently compute skyline queries over road networks.

Privacy Aware Skyline Computation

Till now there is very little consideration about preserving privacy during skyline computation. Qiao et al. [42] proposed skyline queries that are taking into account of privacy. They proposed Range to Ranges skyline queries and Point to Ranges skyline queries for a database containing location information, in which they used cloaking region of each data to compute skyline queries in stead of using exact location. However, they did not preserve privacy of non-spatial attributes.

Su et al. considered top-k combinatorial skyline queries[43]. Their top-k combinatorial skyline problem is to compute the skyline of all s-sets ($s = 1,...,k$). Note that our skyline k-sets are not a subset of their top-k combinatorial skyline. Their results can preserve privacy in a sense if they eliminate combinatorial skyline objects with small cardinality. However, their efficient algorithm is not suitable for privacy-aware distributed databases since it is an incremental algorithm and requires individual record's values to prune unnecessary search.

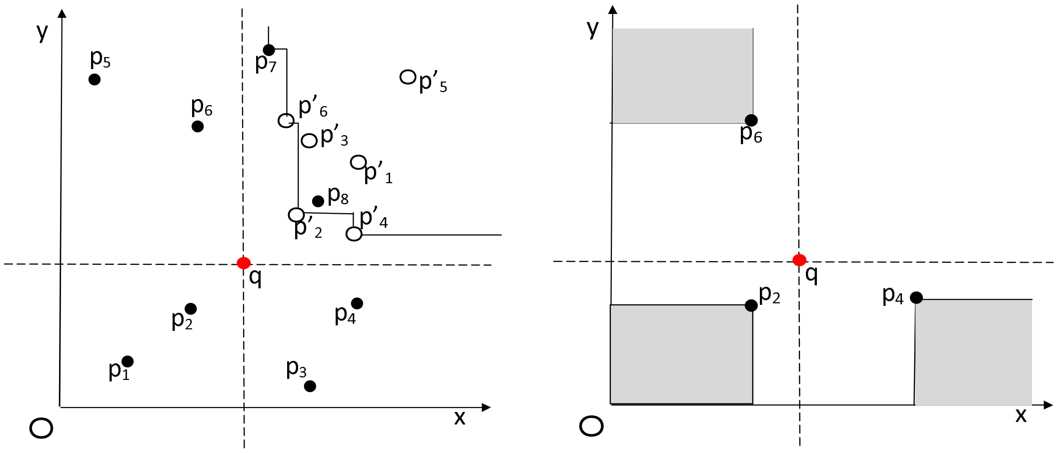
Siddique et al. [44] introduce the concept of skyline sets queries that can preserve individual's privacy while computing skyline sets from a sole database. Later, they extend the idea for spatio-temporal databases [45] . However, their approach is vulnerable against statistical compromisable situations. Moreover, their approach can not compute skyline sets from distributed databases.

Reverse Skyline

Dellis et al. [46] introduced the idea of reverse skyline queries. The concept of Reverse Skyline Queries (RSQ) is based on the concept of Dynamic Skyline Queries (DSQ). Skyline Query becomes dynamic, where the domination is defined with respect to the users query point. Dynamic skyline corresponds to the skyline of a transformed data space where the query point becomes the origin and all points are represented by their coordinate-wise distances to the query point. The reverse skyline query returns the objects whose dynamic skyline contains the query object. We can formally define dynamic skyline queries and reverse skyline queries as follows.

Definition 2.7 (Dynamic Skyline Query [46]) *Given a query point q & a data set P, a Dynamic Skyline Query according to q retrieves all data points from P those are not dynamically dominated. A point $p_1 \in P$ dynamically dominates another point $p_2 \in P$ with regard to the query point q if (1) for all i (i =1 to d): $|q^i$-$p_1^i| \leq |q^i$-$p_2^i|$, & (2) at least one j ($j \in (1,\cdots,d)$):$|q^j$-$p_1^j| < |q^j$-$p_2^j|$.*

Now, consider the example of Figure 2.22(a) where all data points are transformed in the new data space with origin at point q. The absolute distance of the data points to q are used as mapping functions. The dynamic skyline according to q consists of points p_7, p_6, p_2, and p_4 as shown in Figure 2.22(a). It is observed that point p_8 is not part of the dynamic skyline because it is dynamically dominated by point p_2.



(a) Dynamic skyline of q

(b) Reverse skyline of q

Figure 2.22: Dynamic and reverse skyline [46]

Definition 2.8 (Reverse Skyline Query [46]) *Let P be a d-dimensional data set. A Reverse Skyline Query (RSQ) according to the query point q retrieves all points $p_1 \in P$ where q is in the dynamic skyline of p_1. Frmally, a point $p_1 \in P$ is a reverse skyline point of point $q \in P$ iff $\neg\exists$ $p_2 \in P$ such that (1) for all i (i =1 to d): $|p_2^i - p_1^i| \leq |q^i$-$p_1^i|$, & (2) at least one j ($j \in (1, \cdots, d)$):$|p_2^j$-$p_1^j| < |q^i$-$p_1^i|$.*

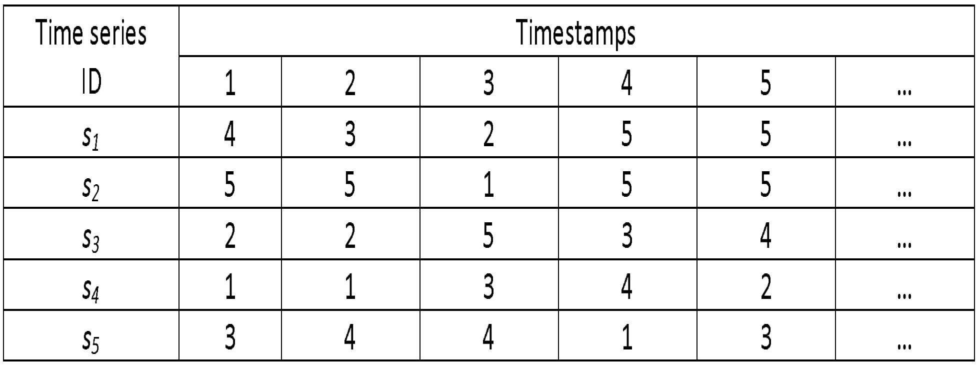
Now, considering the example of Figure 2.22(b) that depicts the RSQ of point q. As illustrated, point p_2 is a reverse skyline point of q since, according to above definition, point q is part of the dynamic skyline of point p_2.

In order to compute the reverse skyline of an arbitrary query point, Dellis et al. [46] proposed several algorithms. Their first algorithm is is Branch and Bound Reverse Skyline (BBRS) algorithm, which is an improved customization of the original BBS [4, 9] algorithm. To reduce the computational cost of BBRS, they developed an enhanced algorithm known as RSSA (Reverse Skyline using Skyline Approximations) that is based on accurate pre-computed approximations of the skylines. These approximations are used to identify whether a point belongs to the reverse skyline or not.

Interval Skyline Queries

In many applications such as environment surveillance, telecommunication, and finance market analysis, we need to analyze a large number of time series. Considering this fact Jiang et al. [47] developed interval skyline queries, a novel type of time series

Table 2.17: A set of time series data [47]



Time series ID	Timestamps					
	1	2	3	4	5	...
s_1	4	3	2	5	5	...
s_2	5	5	1	5	5	...
s_3	2	2	5	3	4	...
s_4	1	1	3	4	2	...
s_5	3	4	4	1	3	...

Table 2.18: The sorted list of time series

Time series	s_2	s_3	s_5	s_1	s_4
max	5	5	4	4	3
min[2 : 3]	1	2	4	2	1

analysis queries. For a set of time series and a given time interval [i : j], an interval skyline query returns the time series which are not dominated by any other time series in the interval.

Jiang et al. [47] proposed two different algorithms for computing interval skyline queries from a set of time series. Their first approach is on-the-fly method that keeps the minimum and the maximum values for each time series and computes the interval skyline at query time.

We can explain on-the-fly method with the help of the example as shown in Table 2.17 that shows 5 time series. Suppose currently the data in interval $W = [1 : 3]$ is maintained. Let us compute the skyline in interval [2 : 3]. Table 2.18 shows the sorted list L of the 5 time series along with their *max* and *min*[2 : 3] values. We first put s_2 in the skyline candidate list and update *maxmin* to 1. Then, we process s_3. As s_2 and s_3 do not dominate each other in [2 : 3]. Thus, s_3 is pushed into the candidate list. Consequently, $maxmin = 2$. Next, s_5 is processed and becomes a candidate. At this stage *maxmin* is updated to 4. The next time series in list L is s_1, which is dominated by s_5 and thus discarded. As *maxmin* is larger than $s_4 : max$, the algorithm terminates and s_2, s_3, and s_5 is returned as skyline in the interval[2 : 3].

Table 2.19: An example of non-redundant skylines [47]

Interval	non-redundant skyline
[2 : 2]	$\{s_2\}$
[3 : 3]	$\{s_3\}$
[4 : 4]	$\{s_1, s_2\}$
[2 : 3]	$\{s_5\}$
[3 : 4]	$\{s_4\}$
[2 : 4]	$\{\oslash\}$

However, above approach cannot provide efficient solution for online Interval Skyline Query. To handle such an issue, the authors used a radix priority search tree[] for each time series. To process a time series, binary tree dimension of a radix priority search tree i.e. dimension x is used as the time dimension (i.e., the timestamps)and the heap dimension i.e dimension Y is used for the data values. Then, it is possible to provide online query answering efficiently.

In addition of on-the-fly method method Jiang et al. [47] also proposed a view-materialization method. The view-materialization method works as follows. (1) First, it computes non-redundant interval skylines in all sub-intervals for the set of time series in the base interval. A time series s is called a non-redundant skyline time series in interval [i : j] if (a) s is in the skyline in interval [i : j]; and (b) s is not in the skyline in any subinterval $[i' : j'] \subset [i : j]$. (2) Next, it takes the union of all non-redundant interval skylines within the desired interval (3) Finally, it performs false positive check to obtain the final skyline result.

Now, consider the computation of interval skyline in [3 : 4] based on the base interval is $W = [2 : 4]$ from Table 2.17 using view-materialization method.

Table 2.19 shows the non-redundant interval skylines in all sub-intervals of W for the set of time series in Table 2.17. To compute the interval skyline in [3 : 4], we union the non-redundant interval skylines in [3 : 3], [4 : 4] and [3 : 4] and obtain the result $\{s_1, s_2, s_3, s_4\}$. As we can see that s_2 is a false positive since it is dominated by s_1 in [3 : 4], we remove s_2 to obtain $\{s_1, s_3, s_4\}$ as the interval skyline result for the interval [3:4].

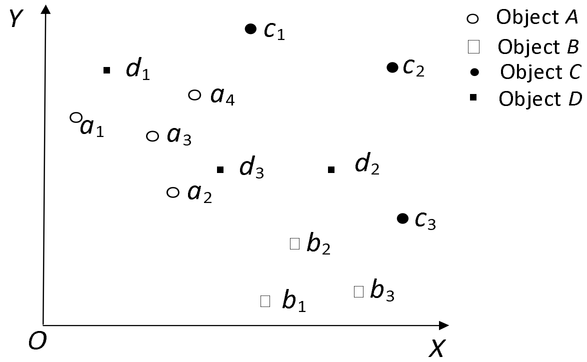


Figure 2.23: A set of uncertain objects [48]

Probabilistic Skyline

There are many situations where the result of conventional skyline queries are not meaningful. As for example, consider the skyline analysis on NBA players. We can say that a player U is in the skyline if there exists no another player V such that V is better than U in one aspect, and is not worst than U in all other aspects. Such skyline results depend on the annual statistics of certain data and attribute of each players represents by aggregate function (mean or median). However, players may have different performances in different games based on different situations. We cannot get benefit from such statistical information using conventional skyline queries. Considering this fact, Pei et al. [48] introduced probabilistic nature of uncertain objects into the skyline analysis. The probability of an object being in the skyline is the probability that the object is not dominated by any other objects. Probabilistic skyline queries can be defined as follows:

Definition 2.9 (Probabilistic Skyline Query [48]) *For a set of uncertain objects S and a probability threshold p* $(0 \leq p \leq 1)$*, the p-skyline is the subset of uncertain objects in S, each of which takes a probability of at least p to be in the skyline. i.e.,* $Sky(p) = \{U \in S | Pr(U) | geqp\}$

Consider the set of uncertain objects as shown in Figure 2.23 and we want to compute skyline objects with a probability threshold ($p \geq 0.8$).

For object A, we can see that every instance of A is not dominated by any instances of objects B, C or D. Thus, the probability that A is dominated by any object is 0, and the probability that A is in the skyline is 1. Similarly, the probability that B is in the skyline is also 1. For object D, instance d_1 is dominated by a_1, d_2 is dominated by a_2, b_1 and b_2, and d_3 is dominated by a_2. Thus, the probability that D is not dominated by any other object can be calculated as $1/3((1-1/4)+(1-1/4)*(1-2/3)+*1-1/4)) = 7/12$. Thus, D takes a probability of 0.58 to be in the skyline. Hence, we can say that object A and object B are in the result of probabilistic skyline.

Pei et al. proposed two different approaches for probabilistic skyline queries. These are bottom-up method and top-down method. Both of these approaches utilize the concept of $bounding - pruning - refining$ iteratively. Their bottom-up approach is well suited for large data sets while the top-down approach has good scalability with respect to cardinality of the data sets.

Preference Based Skyline Computation

Till now there is very little consideration about preference based skyline query processing. There is some consideration about preference based skyline queries in [49, 50]. As for the preference issue, authors in [49] presented an approach to recommend items such as restaurants to a mobile user taking into account his current location and preferences. In this framework, a user initially provides a profile, which records preferences as relative orders within predefined categories such as food types and prices. Then, the system selects skyline items from the database based on the user's profile as well as the current location. Wong et al. [50] provides a skyline computation framework considering dynamic preferences on nominal attributes. In order to compute such preference-based skyline queries efficiently, they proposed a semi-materialization method called the IPO-tree search which stores partial useful results only. These partial results can return skyline results related to each possible preference efficiently.

Keyword-Matched Skyline Query

Consider a situation where a customer wants to find a used car on a shopping web site. He / she is looking for cars with both lower mileage and lower price and wants a car equipped with air bags for safety. Consider another situation where a tourist is looking for a hotel that is cheap and close to the beach having wireless internet and a baby sitter service facility. Conventional skyline queries cannot handle such type of skyline queries. Considering this fact Choi et al. [51] introduced the concept of keyword-matched skyline query. Given a set of query keywords W and a d-dimensional dataset

D^d, a keyword-matched skyline query retrieves the set of skyline tuples whose each textual attribute contains all words of W.

They introduces an inverted-index-based keyword skyline search (INKS) method for retrieving the result of a keyword-matched skyline. INKS consists of two phases: (1) a keyword search and (2) a skyline search. In the keyword search phase INKS performs simple keyword matching through the inverted index [52, 53, 54], so it obtains all docIds for each query keyword by fetching both the index file and the posting file. Next, INKS intersects all docIds according to the given query and then it fetches data objects corresponding to intersected results from database. In skyline search phase, INKS passes the fetched data objects to BNL [1]. Then, INKS computes the skyline tuples by using BNL.

2.3 Skyline Applications

Skyline queries are useful to multi-criteria decision making as they represent the set of all solutions that the user can safely take without fear that something better is out there. It can act as a filter to discard sub-optimal objects. The user can then interactively look at the (smaller) set of skyline objects and further select the ones that fits his/her needs. Below we provide some such examples.

Salesman

The concept of dominance is very useful from the perspective of customer while selecting the products they like as well as for the salesman is whether his products are popular with customers compared to their competitors' products. For example a salesman has to visit promising customers without have any knowledge about customers preferences. However, he can not carry all products. In this situation, he pick up skyline products and carry them. Then he visit customers and find their preferences and pick up recommended products from skyline and sell them.

Cell Phone Finder

Consider a person looking for a suitable cell phone at a website. He/she may care about a large number of features including weight, size, talk time, standby time, screen size, screen resolution, data rate, and camera quality in order to pick one that suits him / her.

There are too many phones at the website for the user to examine them all manually. Computing the skyline over these cell phone features may remove a large number of cell phones whose features are “worse” than those in the skyline. This will reduce the load of manually evaluating a large number of cell phones.

Flight Ticket Reservation

Consider that after completing PhD degree, foreign students need to go back to their home country. In this situation, they need to select their preferable flight. In general, the preference of flights varies from student to student. One student may prefer a flight with low price, while another student may choose a flight having short travel time, yet another student may make their choice based on minimum number of stops. Even some other students can make their choice based on more than one of the above criteria. With the help of skyline queries we can fulfil the needs of all such students by returning all non-dominated routes to them.

Teaching Doctors

Consider the medical records of patients database of a medical hospital. A professor of the hospital wants to find rare medical records of heart attack to include in his lecture. In order to obtain such rare medical records, the Professor can execute a skyline query looking for records on heart attack in young patients (minimizing age), with small number of occurrence of heart attack in the immediate relatives (minimizing number of occurrences), and with high weight (maximizing weight). In this case the skyline returns the records of patients that are not dominated (worse) by any other in all attributes of interest.

Chapter 3

Privacy Preserving Information Retrieval

Given a set of objects, a skyline query retrieves objects those are not dominated by another object. A skyline query helps us to filter unnecessary information efficiently and gives us clues for various decision making tasks. However, most of the existing skyline algorithms cannot preserve individual's privacy and are not well suited for data with outliers and frequently updated data. Considering the issue of privacy, earlier Siddique et al. [44] proposed skyline sets queries from a sole database. However, with the increase of data volume, in most cases data are stored in distributed databases. Considering this fact, we expand the idea of skyline sets queries from distributed databases. We propose an efficient computation framework that can preserve individual's privacy while computing skyline set queries from distributed databases. The proposed method utilizes an agent-based parallel computation framework.

This chapter is organized as follows. In Section 3.1, we describe the privacy problem and some other related issues related to skyline queries. Section 3.2 describes the concept of skyline sets queries and its applicability in preserving individual's privacy. We also describe its usefulness in some other applications. In Section 3.3, we provide a review of works related to our work in this chapter. Section 3.4 describes the preliminary concepts related to the work of this chapter. In Section 3.5, we detail the privacy preserving computation process of skyline sets queries from distributed databases. In Section 3.6, we present an approach of dealing with missing values during skyline sets queries. Section 3.7 presents the experimental results. Finally, we conclude the chapter in Section 3.8.

3.1 Privacy Problem

Skyline queries retrieve a set of skyline objects so that a user can choose promising objects or eliminate unnecessary objects. Given a k-dimensional database DB, an object p is said to be in skyline of DB if there is no object q in DB such that q is better than p in all k-dimensions. If there exists such an object q, then we say that p is dominated by q or q dominates p. Consider the example of Figure 3.1. The table in Figure 3.1(a) is a list of five records of a hotel database, each of which contains two numerical attributes "Price" and "Distance". In the list, the best choice usually comes from the skyline, i.e., one of $\{h_1, h_3, h_4\}$ (See Figure 3.1 (b)). A number of efficient algorithms for computing skyline have been reported in the literature [1, 4, 5, 7, 8, 11, 14, 15, 16, 18, 20, 24, 63, 64].

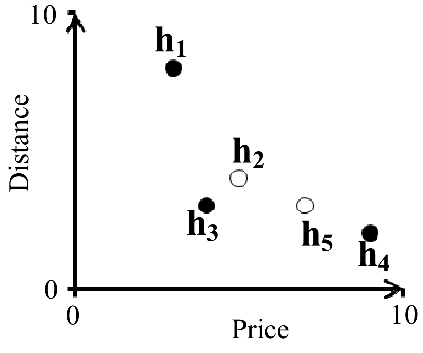
However, there is no consideration about the issue of individual's privacy in these works. As for example, if we look at the skyline result in Figure 3.1(b), we can see individual values in each of the three records. However, this type of discloser of records' values are not allowable as individual record may contain very sensitive and secure information and discloser of such a record can create problems for the record owner. Considering such situations, preserving individual's privacy becomes an important issue in data management. In a database, it might be necessary to hide individual information to preserve privacy. People often do not want to disclose their records' values during the computation procedure although they want to take benefit from the computation process. In such privacy aware environments, we cannot use conventional skyline queries. In addition to the privacy issue, there are some other problems of conventional skyline queries.

First, they are not robust in case of databases with outliers, though the existence of outliers is a common problem and it is very much important to minimize the effect of outliers in the computation. In different field of computing i.e. ubiquitous computing and sensor computing, we need to collect data via various sensor devices. In such situations, there are strong possibilities that some data may be outside of the general distribution of the data. Use of any conventional skyline query algorithm in such a data set may retrieve some outliers as skyline. Taking decision based on such skyline results can create serious problems in decision-making.

Consider the example of Figure 3.2. The table in Figure 3.2 is a list of five records of a sensor device, each of which contains two numerical attributes "Temperature" and

ID	Price	Distance
h_1	3	8
h_2	5	4
h_3	4	3
h_4	9	2
h_5	7	3

(a) Hotels data

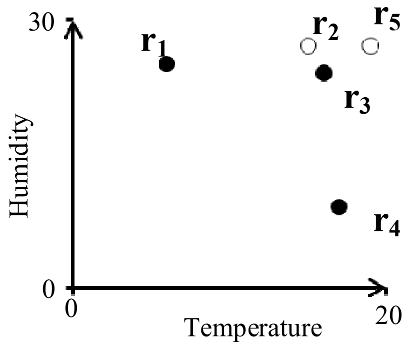


(b) Skyline

Figure 3.1: Hotels data and corresponding skyline

ID	Temperature	Humidity
r_1	6	25
r_2	15	27
r_3	16	24
r_4	17	9
r_5	19	27

(a) Sensor data



(b) Skyline

Figure 3.2: Sensor data and corresponding skyline

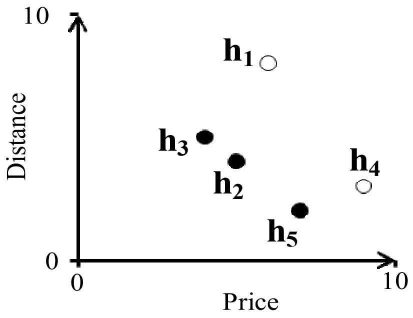
"Humidity". From the data of Figure 3.2(a), we can see that there are two records r_1 and r_4 those have wrong values, which we call outliers, in their attributes "Temperature" and "Humidity", respectively. If we perform a conventional skyline query, it returns r_1, r_3, and r_4 (see Figure 3.2(b)) as skylines. Note that in the result two records r_1 and r_4 have been included. As these two records have outliers in their attributes, taking decision based on either r_1 or r_4 will create a great problem.

Second, conventional skyline query algorithms are not stable in case of update operation. Since skyline takes time to compute, we usually use precompute skyline results to answer a query quickly. If a record in a database is updated, we have to recompute the skyline results. However, if records in a database are frequently updated, we cannot prepare the precomputed results in time.

Consider that the hotel database in Figure 3.1 has been updated and some of its

ID	Price	Distance
h_1	6	8
h_2	5	4
h_3	4	5
h_4	9	3
h_5	7	2

(a) Hotels data after update

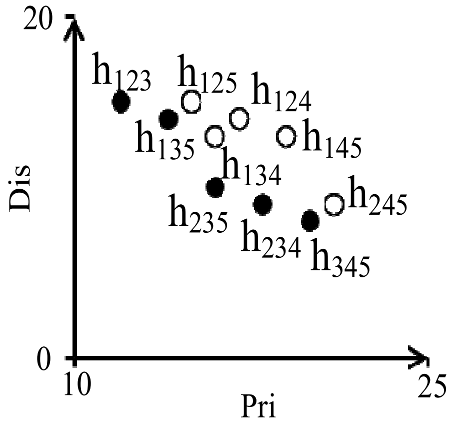


(b) Skyline after update

Figure 3.3: Hotels data and corresponding skyline after update

ID	Pri	Dis
h_{123}	12	15
h_{124}	17	14
h_{125}	15	15
h_{134}	16	13
h_{135}	14	14
h_{145}	19	13
h_{234}	18	9
h_{235}	16	10
h_{245}	21	9
h_{345}	20	8

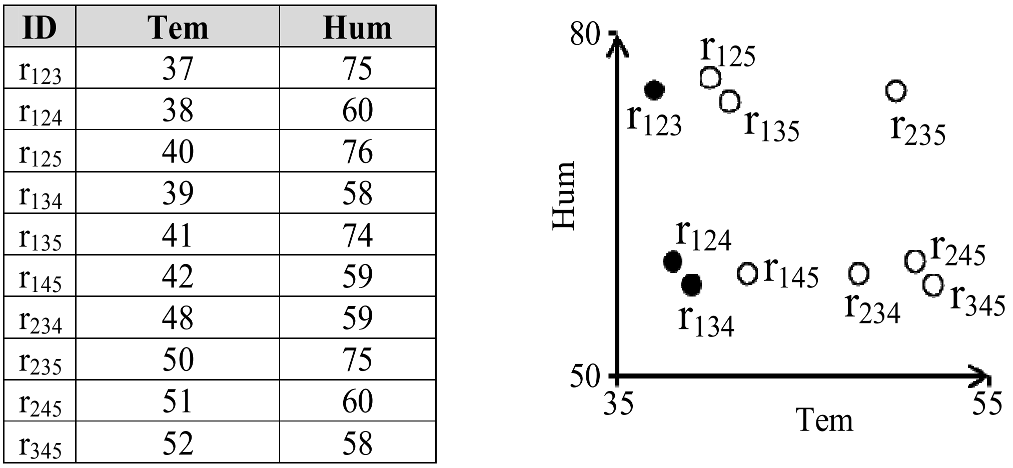
(a) Sets of three hotels before update



(b) Skyline of 3- hotels before update

Figure 3.4: Three sets of hotels and corresponding skyline 3-sets

records' values are changed as shown in Figure 3.3. From Figure 3.3, we can easily find that the result of any conventional skyline query is h_2, h_3 and h_5, which is different from the result before update. We can see that precomputed skyline $\{h_1, h_3, h_4\}$ is meaningless for users who are interested in the cheapest price unless it is recomputed. This problem has been an important research issue of skyline query.



ID	Tem	Hum
r_{123}	37	75
r_{124}	38	60
r_{125}	40	76
r_{134}	39	58
r_{135}	41	74
r_{145}	42	59
r_{234}	48	59
r_{235}	50	75
r_{245}	51	60
r_{345}	52	58

(a) 3-sets of sensor data

(b) Skyline of 3-sets

Figure 3.5: Three sets of sensor data and corresponding skyline 3-sets

3.2 Skyline Sets

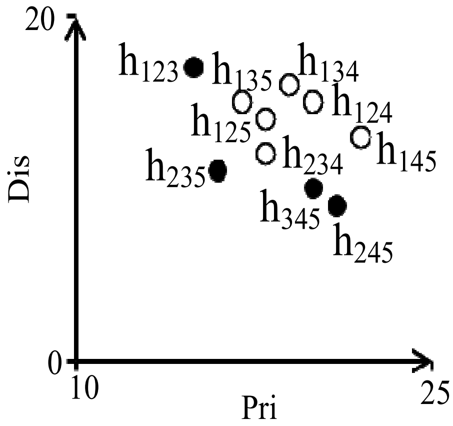
Focusing on the privacy issue, for the first time, Siddique et al. [44] proposed the concept of skyline sets queries from a sole database .

Figure 3.4 is a list of 3-sets, in which all of the combinations of three records of Figure 3.1(a) are listed. In Figure 3.4(a), “ID” denotes a set of three records and the attributes “Pri” and “Dis” represent the sums of price and distance of three records of Figure 3.1(a)), respectively. In the example, h_{123} denotes a set of three records r_1, r_2, and r_3. “Pri” and “Dist” of h_{123} are the sums of the “Price” and “Distance” of records in the set, respectively. The skyline of the combinations of three records are $\{h_{123}, h_{135}, h_{235}, h_{234}, h_{345}\}$ as shown Figure 3.4(b). Note that if one wants to know the cheapest hotel, she/he can find that the cheapest set is h_{123} from the skyline and can easily imagine that the price of the cheapest hotel is around 4, since price of the cheapest 3-set h_{123} is 12. Similarly, if one prefers cheaper and closer, she/he may choose h_{235} from the skyline and can easily imagine the value of the preferable choice from the aggregated values. In such a way skyline sets queries proposed in [44] can preserve individual’s privacy in most of the cases while computing skyline sets from a sole database.

The work in [44] is also robust to a certain level against data outliers and for frequently updated data.

ID	Pri	Dis
h_{123}	15	17
h_{124}	20	15
h_{125}	18	14
h_{134}	19	16
h_{135}	17	15
h_{145}	22	13
h_{234}	18	12
h_{235}	16	11
h_{245}	21	9
h_{345}	20	10

(a) Sets of three hotels after update



(b) Skyline of 3-hotels after update

Figure 3.6: Three sets of hotels and corresponding skyline 3-sets after update

As for example, if we look at the result in Figure 3.5(b), we can find that the skyline 3-sets correspond to the sensor data of Figure 3.2 are r_{123}, r_{124}, and r_{134}. Note that in temperature sensitive situations if anyone takes decision based on the result of conventional skyline queries as shown in Figure 3.2, she / he will choose r_1 and the decision will be a wrong one as the value “6” in “Temperature” attribute is an outlier. On the other hand, it is observed that if the decision is taken based on the result of skyline sets queries, we can see that the chosen value is around “12” and the decision is much more accurate. In this way, using skyline set queries we can reduce the effect of outliers at a certain level.

If we consider the update situation and skyline sets queries with $s = 3$ from our examples of Figure 3.1 and Figure 3.3, we can find that the results are $\{h_{123}, h_{135}, h_{235}, h_{234}, h_{345}\}$ and $\{h_{123}, h_{235}, h_{245}, h_{345}\}$ as shown in Figure 3.4 and Figure 3.6, respectively. From the results of skyline sets of Figure 3.4 and Figure 3.6, we can see that the result is almost same before and update operation. That means skyline sets queries are more robust against update than conventional skyline queries.

However, there is no is still consideration of computing skyline sets queries from distributed databases although due to exponential growth in data volume, data are often store in distributed databases nowadays. Considering this fact, in this chapter, at first, we introduce an agent-based parallel computation framework for skyline sets queries from distributed databases that shows very good computation performance.

ID	a_1	a_2
o_1	5	5
o_2	1	6
o_3	6	4
o_4	9	7
o_5	6	8

DB_1

ID	a_1	a_2
o_6	1	3
o_7	7	2
o_8	7	9
o_9	4	5
o_{10}	6	7

DB_2

ID	a_1	a_2
o_{11}	4	2
o_{12}	8	4
o_{13}	4	1
o_{14}	8	7
o_{15}	1	5

DB_3

Figure 3.7: Three databases with same schema

Moreover, the method of [44] cannot protect individual record's values under statistically compromisable situations. For example, let us consider the three databases with same schema (ID, a_1, a_2) as shown in Figure 3.7. In this example, we assume that value domain of both a_1 and a_2 is set to $[1 \cdots 10]$.

Skyline sets queries with $s = 3$ returns the results as shown in Table 3.1 as the skyline of 3-sets. Notice that the aggregated values in Table 3.1 are disclosed to the public including database owners.

From the a_1 of R_1, any user can easily find that individual record's value in a_1 is 1 because the minimum value in a_1 is 1. Therefore, the owner of DB_1 can find there are two records whose a_1 is 1 in other databases. Similarly, the owner of DB_2 and DB_3 can find the fact.

From the value of a_2 of R_2, one can find that one of the three records contains 1 in a_2 since aggregated value of 3 records is 5. Therefore, one can infer that the remaining two records have values 1 and 3 or 2 and 2 in a_2. In this case, the owner of DB_3 happens to know that one of others has 2 in a_2 and no other has 1 in a_2 since DB_3 has 1 and 2.

Considering this fact, later on, we provide a framework for detecting such compromising skyline s-sets and propose a protection mechanism for such s-sets. We also introduce an efficient agent-based parallel computation framework that improves the overall computation performance significantly.

Table 3.1: Results of skyline sets queries for $s = 3$ from the databases of Figure 3.7

ID	a_1	a_2
R_1	3	14
R_2	15	5
R_3	6	9
R_4	9	6

3.3 Related Work on Privacy Problem

3.3.1 Skyline Query

Borzonyi et al. [1] first introduced the skyline operator into database systems and proposed Block Nested Loop (BNL), Divide-and-Conquer, and B-tree based algorithms. As a variant of BNL, Chomicki et al. [5] improved BNL algorithm with the help of a sort-Filter-Skyline (SFS) algorithm. In SFS, data needs to be pre-sorted using a monotone scoring function, which can simplify the selection of skyline objects. Tan et al. [7] proposed two progressive algorithms: Bitmap and Index. The bitmap algorithm represents points in bit vectors and performs bit-wise operations. On the other hand, the index approach uses data transformation and B+-tree indexing. Kossmann et al. [8] proposed a Nearest Neighbor (NN) method. It selects skyline points by recursively invoking R*-tree based depth-first NN search over different data portions. Papadias et al. [4] proposed a Branch-and-Bound Skyline (BBS) method based on the best-first nearest neighbor algorithm. Godfrey et al. [63] provided a comprehensive analysis of previous skyline algorithms without indexing supports and proposed a new hybrid method with improvement. However, none of these works can preserve privacy of records.

3.3.2 Skyline Query in Distributed Environment

In [10], Wu et al. first address the problem of parallelizing skyline queries over a shared-nothing architecture. They provided two mechanisms: recursive region partitioning and dynamic region encoding for the execution of skyline queries. In their approach, a server starts the skyline computation on its data after receiving the results of other servers based on the partial order. It causes a waiting delay of the servers.

In our work, such problem is localized within the cluster. Park et al. [68] introduced two parallel skyline algorithms in multicore architectures. The first one is a parallel version of BBS algorithm. The second one is known as pskyline, which is based on skeletal parallel programming [72]. Gao et al. [69] proposed parallel computation of skyline queries in multi-disk environment using parallel R-trees. The core of their scheme is to visit more entries simultaneously and to enable effective pruning strategies. Cui et al. [70] introduced skyline queries in large-scale distributed environments without the assumption of any overlay structures and propose PaDSkyline algorithm. PaDSkyline is an algorithm that significantly reduces the response time by performing parallel processing over site groups produced by a partition algorithm. Within each group, it locally optimizes the query processing. It also improves the network transmission efficiency by performing early reduction of skyline candidates. Vlachou et al. [71] proposed an angle-based space partitioning scheme for parallel computation of skylines of data points using the hyperspherical coordinates of the data points. In their approach, data point are almost equally spread among the partitions that increases the average pruning power of data points.

Huang et al. [15] considered a setting with mobile devices communicating via an ad-hoc network (MANETs) and studied skyline queries that involved spatial constraints. In this approach, queries are forwarded through the whole MANET without routing information. They proposed a filtering based data reduction technique to reduce the data transfer among devices. However, in our work, we assume a wired large-scale distributed environment in which query results from each cluster are sent to the coordinator for computing skyline sets. In [18], Vlachou et al. studied the problem of subspace skyline processing in a super-peer network. In this approach, peers hold their data in an autonomous manner and collectively process skyline queries on subspaces. Hose et al. [16] introduced relaxed skylines in Peer Data Management Systems (PDMS). They proposed a strategy for processing relaxed skylines in distributed environments using distributed data summaries. For efficient computation of skyline, Wang et al. [11] use the z-curve method to map the multidimensional data to one dimensional values. The one dimensional values are then assigned to peers connected in a tree overlay like BATON [66]. In this approach, the problem of load balancing arises. In particular, in their approach the peers near the origin of the axes need to process most of the queries. Li et al. [14] use a space partitioning method that is based on an underlying semantic overlay. Their approach shares the same drawbacks as [11]. Balke et al. [64] addressed skyline operation over web databases where different dimensions are stored in different data sites. Their algorithm first retrieves values in every dimension from remote data sites using sorted access in round-robin fashion on all dimensions. This continues until all dimension values of an object, called the terminating object, have been retrieved. Then, all non-skyline objects are filtered from all those objects with at least one dimension value retrieved. In [20], Fotiadou et al. proposed a bitmap approach for efficient subspace skyline computation in a distributed setting. The bitmap approach computes extended skylines that includes all points necessary for computing the skyline at any subspace. They presented an algorithm for computing extended skylines using a bitmap representation along with a storage efficient bucket-based variation of bitmap representation. Rocha et al. [24] introduced an efficient execution plans for distributed skyline query processing. In this paper, the authors proposed SkyPlan, a mechanism for querying servers consecutively. Their approach reduces the amount of transferred data and the number of queried servers.

Most research papers on parallel and distributed skyline query processing so far have the problem of load balancing. As a result few peers need to carry almost all the processing burden, while most other peers remain idle. Moreover, there is no consideration about individuals privacy. In contrast, we introduce a parallelizing mechanism in which every server takes part in the computation simultaneously and preserves individual's privacy.

3.3.3 Privacy Preserving Skyline Query

Qiao et al. [42] proposed skyline queries that are taking into account of privacy. They proposed Range to Ranges skyline queries and Point to Ranges skyline queries for a database containing location information, in which they used cloaking region of each data to compute skyline queries in stead of using exact location. However, they did not preserve privacy of non-spatial attributes. Su et al. [43] considered top-k combinatorial skyline queries. Their top-k combinatorial skyline problem is to compute the skyline of all s-sets ($s = 1, ..., k$). Their results can preserve privacy in a sense if they eliminate combinatorial skyline objects with small cardinality. However, their efficient algorithm is not suitable for privacy-aware distributed databases since it is an incremental algorithm and requires individual record's values to prune unnecessary search. The work in [44] can preserve privacy to some extent while computing skyline queries from a sole database. They introduced the concept of skyline sets queries. However, they did not consider the computation of skyline sets queries from distributed databases.

In this chapter, we extend the idea of [44] for skyline sets queries from distributed databases. Our framework can compute skyline sets queries from distributed databases efficiently without disclosing individual record's.

3.4 Preliminaries

We consider a database DB having k attributes and n objects. Let $a_1, a_2, \cdots, a_k$ be the k attributes of DB. Without loss of generality, we assume that smaller values in each attribute are better and each attribute contains positive integer values.

3.4.1 Skyline Queries

Let p and q be objects in DB. Let $p.a_l$ and $q.a_l$ be the l-th attribute values of p and q, respectively, where $1 \leq l \leq k$. An object p is said to dominate another object q, if $p.a_l \leq q.a_l$ for all the k attributes a_l, $(1 \leq l \leq k)$ and $p.a_j < q.a_j$ on at least one attribute a_j, $(1 \leq j \leq k)$. The skyline is a set of objects which are not dominated by any other object in DB.

3.4.2 Skyline Sets Problem

Let $|S| = {}_nC_s = \frac{n!}{s!(n-s)!}$ be the number of s-sets that can be composed from n objects. We assume a virtual database of S on the k dimensional space of DB. Each object of the database is an s-set whose value of each attribute (dimension) is the sum of s values of corresponding s objects. An s-set $p \in S$ is said to dominate another s-set $q \in S$, denoted as $p \leq q$, if $p.a_l \leq q.a_l$, $1 \leq l \leq k$ for all k attributes and $p.a_j < q.a_j$, $1 \leq j \leq k$ for at least one attribute. We call such p as dominant s-set and q as dominated s-set between p and q.

An s-set $p \in S$ is said to be a skyline s-set if p is not dominated by any other s-set in S.

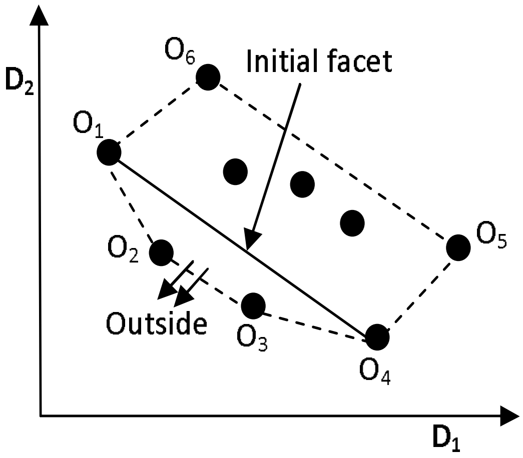


Figure 3.8: Convex hull and convex skyline

3.4.3 Convex Skyline Sets

Each object in S is a point in k-dimensional vector space. Convex hull for the set of S points is the minimum convex solid that encloses all of the objects of S. The dotted line polygon of Figure 3.8 is an example of convex hull in two-dimensional space. In the Figure 3.8, O_1 and O_4 are the objects that have the minimum values in D_1 and D_2, respectively. Notice that such objects must be in the convex hull. We call the line between O_1 and O_4 "the initial facet". Among all objects in the convex hull, objects that lie outside of the initial facet are skyline objects and we call such objects "convex skyline objects". In k-dimensional space, we compute such initial hyperplane surrounded by k objects as the initial facet. Then, we compute convex skyline objects that lie in the convex hull and outside the initial facet.

The definition of convex skyline sets problem can be simplified as follows:

Given a natural number s, find all s-sets those lay in both the convex hull and the skyline of S.

3.4.4 Algorithm for Computing Convex Skyline Sets

If we compute all of the s-sets from the original database and make a dataset containing $|S|$ records, the problem can be solved by conventional skyline query algorithms. However, $|S|$ is unacceptably large when the original database size is large. Therefore, in [44] we proposed an efficient alternative.

Table 3.2: Inner product with tangent lines

$\mathbf{o}$	$(\Theta_1, \mathbf{o})$	$(\Theta_2, \mathbf{o})$	$(\Theta_{1,2}, \mathbf{o})$
$\mathbf{h_1}$	$\underline{-3}$	-8	-85
$\mathbf{h_2}$	$\underline{-5}$	-4	$\underline{-67}$
$\mathbf{h_3}$	$\underline{-4}$	$\underline{-3}$	$\underline{-52}$
$\mathbf{h_4}$	-9	$\underline{-2}$	-79
$\mathbf{h_5}$	-7	$\underline{-3}$	$\underline{-73}$

Each s-set in S can be represented as a k-dimensional point $x = (x_1, x_2, \cdots, x_k)$ where x_i, $(1 \leq i \leq k)$ is the sum of the i-th attribute value of the s objects in DB. "Touching oracle" function proposed in [44] is a method to compute an s-set on the convex hull without generating S. It computes the tangent object of the convex hull of S and a (k-1)-dimensional hyperplane directly from the original n records in DB.

3.4.4.1 Touching Oracle

Assume there is a hyperplane whose normal vector is Θ. In order to find the tangent point with the hyperplane and the convex hull without precomputing S, we compute (Θ, o), i.e., inner products of the normal vector and each object o in the database. We choose s objects whose inner products are the top s. These top s objects compose the s-set of the tangent point. Consider the hotel example as shown in Figure 3.1. There are five records in the original database DB as in Figure 3.1(a). Each of the five records is represented as a two-dimensional point, which we call an atomic point.

Now consider the line whose normal vector is Θ_1 = (-1, 0). In order to find the tangent point corresponding to the line with Θ_1, we compute inner products of the normal vector and each of the five atomic points as shown in the second column of Table 3.2. Then, we choose the top three inner products, i.e., $\{h_1, h_2, h_3\}$ if s = 3. These top three inner products compose the tangent point (12, 15), which is the 3-set, h_{123}. Similarly, for a line with Θ_2 = (0, -1), we can find $\{h_3, h_4, h_5\}$ as the top three. Those three points compose the tangent point (20, 8).

As mentioned above, we can compute a tangent point, which is a point on the convex hull, by giving the normal vector of a tangent line. In k-dimensional case, we can find a tangent point with a tangent (k -1)-dimensional hyperplane by giving the normal vector of the tangent (k -1)-dimensional hyperplane.

The touching oracle function chooses the top-s points from n atomic points in DB. Since s is a negligible small constant compared with n, we can compute the tangent point by scanning n atomic points only once, which is $O(n)$.

3.4.4.2 Convex Hull Search

First, we compute initial k tangent objects that can be computed by touching oracle with initial k vectors $\Theta_x = (\theta_1, \theta_2, \cdots, \theta_k)$, where θ_i = -1 if $i = x$, otherwise $\theta_i = 0$ for each $x = 1, \cdots, k$. Note that those k initial tangent objects are on the horizon of the initial facet ((k-1)-dimensional hyperplane). Convex skyline s-sets are objects lie outside of the initial facet and are in the convex hull.

Next, we compute the normal vector of the initial facet. Using the computed normal vector, we try to find new tangent point. The new tangent point expands the initial facet into k facets. We recursively compute the touching oracle for each of the expanded facets until we can find a new tangent object outside the facet.

From Table 3.2, for example, we have obtained two initial tangent points $p_1 = (12, 15)$ and $p_2 = (20, 8)$ with normal vectors $\theta_1 = (-1, 0)$ and $\theta_2 = (0, -1)$, respectively. These two initial tangent points constructs the initial facet. Using the facet containing the two initial points, we can compute the normal vector of the facet as $\theta_{1,2} = (-(15-8), (12-20)) = (-7, -8)$, which directs outside of the facet. Using this normal vector, we can find new tangent point h_{235}, which is (16, 10). The new tangent point expands the initial facet into two facets, which are the facet surrounded by $p_1 = (12, 15)$ and (16, 10) and the facet surrounded by (16, 10) and $p_2 = (20, 8)$. We can apply this operation for higher k-dimensional space analogically as follows.

Three Dimensional Case:

Assume we have a facet surrounded by following three points:

$$P1 = (p1_1, p1_2, p1_3)$$

$$P2 = (p2_1, p2_2, p2_3)$$

$$P3 = (p3_1, p3_2, p3_3)$$

We assume that P1, P2 and P3 are clockwise order when we look the facet from outside of the convex hull. Now, we can compute two edge vectors by using the three points as follows. Suppose the edges are following vectors.

$$V1 = (v1_1, v1_2, v1_3) = (p2_1, p2_2, p2_3) - (p1_1, p1_2, p1_3)$$

$$V2 = (v2_1, v2_2, v2_3) = (p3_1, p3_2, p3_3) - (p1_1, p1_2, p1_3)$$

The outside normal vector of this facet is computed as the expansion of the following symbolic determinant.

$$V1 \otimes V2 = \begin{vmatrix} e_1 & e_2 & e_3 \\ v1_1 & v1_2 & v1_3 \\ v2_1 & v2_2 & v2_3 \end{vmatrix}$$

In the formula, e_1, e_2, and e_3 are the elementary vectors (1,0,0), (0,1,0) and (0,0,1) respectively. Using this normal vector, we can divide this facet into three facets if we can find a new tangent point outside of the facet by the touching oracle function. If P is found outside of the facet, then the three new facets are as follows:

$$(P1, P, P3)$$

$$(P1, P2, P)$$

$$(P, P2, P3)$$

The normal vectors of these three facets are

$$(P - P1) \otimes (P3 - P1)$$

$$(P2 - P1) \otimes (P - P1)$$

$$(P2 - P) \otimes (P3 - P)$$

if points in each facet are clockwise order when we look the facet from outside of convex hull.

Four Dimensional Case:

We can use the idea into higher dimensional case analogically.

Assume that we have a facet surrounded by four points as follows:

$$P1 = (p1_1, p1_2, p1_3, p1_4)$$

$$P2 = (p2_1, p2_2, p2_3, p2_4)$$

$$P3 = (p3_1, p3_2, p3_3, p3_4)$$

$$P4 = (p4_1, p4_2, p4_3, p4_4)$$

Using similar operations of 3D case, we can compute three vectors as follows:

$$V1 = (v1_1, v1_2, v1_3, v1_4) = P2 - P1$$

$$V2 = (v2_1, v2_2, v2_3, v2_4) = P3 - P1$$

$$V3 = (v3_1, v3_2, v3_3, v3_4) = P4 - P1$$

Then, the normal vector can be computed as the expansion of the following determinant.

$$V1 \otimes V2 \otimes V3 = \begin{vmatrix} e_1 & e_2 & e_3 & e_4 \\ v1_1 & v1_2 & v1_3 & v1_4 \\ v2_1 & v2_2 & v2_3 & v2_4 \\ v3_1 & v3_2 & v3_3 & v3_4 \end{vmatrix}$$

In the determinant, the value of e_1, e_2, e_3 and e_4 are (1,0,0,0), (0,1,0,0), (0,0,1,0), and (0,0,0,1), respectively. If P is found, then the four new facets are as follows:

$$(P1, P2, P3, P)$$

$$(P1, P2, P, P4)$$

$$(P1, P, P3, P4)$$

$$(P, P2, P3, P4)$$

The normal vectors of these four facets are as follows:

$$(P2 - P1) \otimes (P3 - P1) \otimes (P - P1)$$

$$(P2 - P1) \otimes (P - P1) \otimes (P4 - P1)$$

$$(P - P1) \otimes (P3 - P1) \otimes (P4 - P1)$$

$$(P2 - P) \otimes (P3 - P) \otimes (P4 - P)$$

k-Dimensional Case:

We can expand above operations for k-dimensional case. Assume we have a facet surrounded by following k points.

$$P1 = (p1_1, p1_2, ..., p1_k)$$

$$P2 = (p2_1, p2_2, ..., p2_k)$$

$$\ldots$$

$$Pk = (pk_1, pk_2, ..., pk_k)$$

We can calculate $(k-1)$ vectors like $V1$, $V2$, $\cdots$, $V(k-1)$. Then, the normal vector of the facet that directs outside can be computed as the expansion of the following determinant.

$$V1 \otimes \cdots \otimes V(k-1) = \begin{vmatrix} e_1 & \cdots & e_k \\ v1_1 & \cdots & v1_k \\ \cdots & \cdots & \cdots \\ v(k-1)_1 & \cdots & v(k-1)_k \end{vmatrix}$$

If P is found, then the k new facets are as follows:

$$(P, P2, \cdots, Pk-1, Pk)$$

$$(P1, P, \cdots, Pk-1, Pk)$$

$$\ldots$$

$$(P1, P2, \cdots, Pk-1, P)$$

The normal vectors of these k facets are as follows:

$$((P2-P) \otimes \cdots \otimes (Pk-1-P) \otimes (Pk-P))$$

$$((P-P1) \otimes \cdots \otimes (Pk-1-P1) \otimes (Pk-P1))$$

$$\ldots$$

$$((P2-P1) \otimes \cdots \otimes (Pk-1-P1) \otimes (P-P1))$$

Performance of these higher dimensional computation can be found in [67].

ID	a_1	a_2
o_1	6	3
o_2	3	5
o_3	7	5
o_4	5	8
o_5	4	6

DB_1

ID	a_1	a_2
o_6	7	8
o_7	8	3
o_8	4	9
o_9	3	7
o_{10}	8	5

DB_2

ID	a_1	a_2
o_{11}	5	5
o_{12}	2	6
o_{13}	6	4
o_{14}	9	7
o_{15}	6	8

DB_3

ID	a_1	a_2
o_{16}	8	3
o_{17}	9	4
o_{18}	6	3
o_{19}	7	6
o_{20}	6	6

DB_4

ID	a_1	a_2
o_{21}	1	3
o_{22}	8	4
o_{23}	7	9
o_{24}	4	5
o_{25}	6	7

DB_5

ID	a_1	a_2
o_{26}	3	5
o_{27}	4	1
o_{28}	7	2
o_{29}	2	8
o_{30}	5	3

DB_6

ID	a_1	a_2
o_{31}	9	3
o_{32}	4	8
o_{33}	3	3
o_{34}	1	5
o_{35}	8	7

DB_7

ID	a_1	a_2
o_{36}	9	8
o_{37}	3	4
o_{38}	6	6
o_{39}	4	2
o_{40}	6	7

DB_8

Figure 3.9: Distributed database example

3.5 Secure Parallel Computation of Skyline Sets

3.5.1 Problem Formulation

We assume that there are m databases in a network. Let $DB_1, DB_2, \cdots, DB_m$ be the databases. Each database has a view table whose schema has following columns: ID, a_1, a_2, $\cdots$, a_k, where ID is the primary key attribute and a_i ($i = 1, \cdots, k$) are k-dimensional numerical attributes. Assume that we have to compute skyline sets for the union of m such databases in such a way that the privacy of individual's is preserved. Figure 3.9 is an example of a distributed database that consists of eight databases, DB_1, DB_2, $\cdots$, and DB_8, each of which lies in a different server.

3.5.2 Agent-based Parallel Computation

We assume there is a coordinator who is responsible for performing the convex hull search, which is mentioned in subsection 3.4.4. The coordinator computes the touching oracle function, which is to find the top-s inner product values from distributed databases, by the divide-and-conquer strategy.

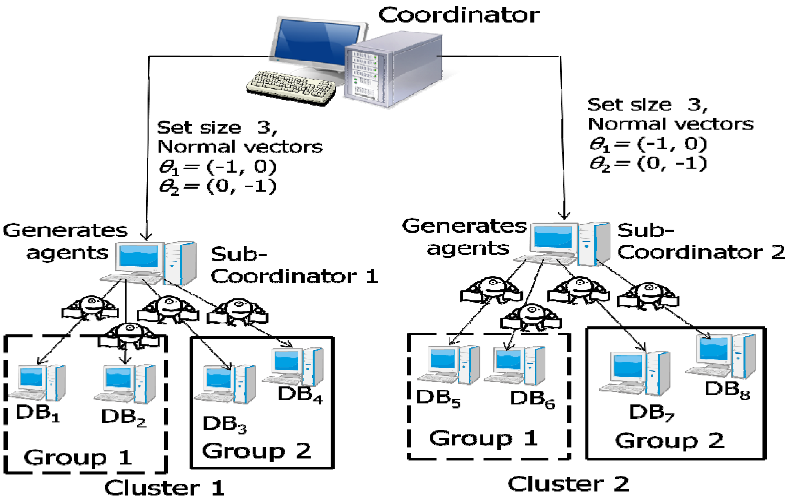


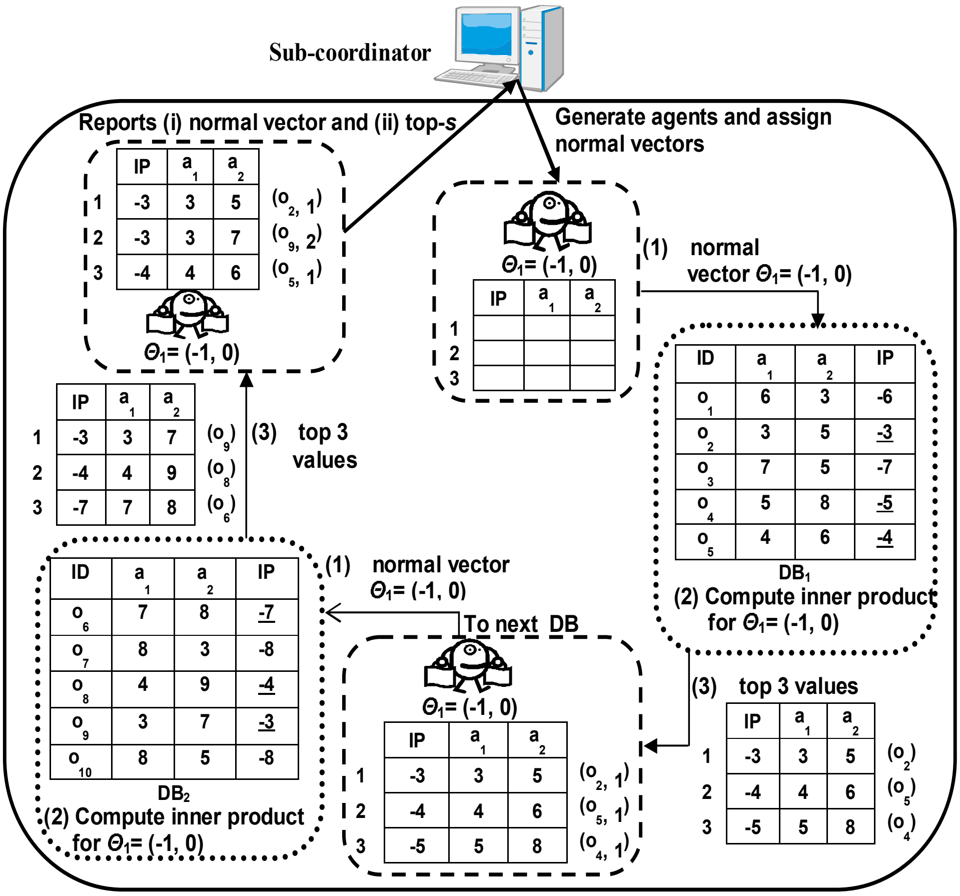
Figure 3.10: Example of divide-and-conquer computation

The coordinator divides the distributed databases into several clusters and creates sub-coordinators for each cluster. For each cluster, the sub-coordinator computes the “local” top-s among the databases in the corresponding cluster. After computing all “local” top-s, the coordinator merges all the “local” top-s and finds “global” top-s. During the process, agents are used to preserve privacy of all “local” databases. Note that all the “local” computations are performed simultaneously. Now, consider the secure computation of skyline 3-set query from the distributed databases of Figure 3.9. For each cluster, the coordinator asks the sub-coordinator to compute touching oracles for the two initial normal vectors, i.e., (-1, 0) and (0, -1).

3.5.2.1 Computation in Each Cluster

In order to minimize idle time, the sub-coordinator divides databases into several groups. In general, the number of groups depends on the number of different normal vectors, which are in the process. However, if the number of databases in a cluster is less than the number of normal vectors, we set the number of groups to the number of databases. For example, if we are processing the two initial normal vectors, databases of the cluster are divided into two groups as in Figure 3.10.

For each group, the sub-coordinator creates two agents one of which is for $(-1,0)$ and the other is for $(0,-1)$. Each agent has a normal vector and a priority queue,


Sub-coordinator
Reports (i) normal vector and (ii) top-s
Generate agents and assign normal vectors
Θ1= (-1, 0)
(1) normal vector Θ1= (-1, 0)
DB1
(2) Compute inner product for Θ1= (-1, 0)
(3) top 3 values
To next DB
(1) normal vector Θ1= (-1, 0)
DB2
(2) Compute inner product for Θ1= (-1, 0)
(3) top 3 values

	IP	a1	a2	
1	-3	3	5	(o2, 1)
2	-3	3	7	(o9, 2)
3	-4	4	6	(o5, 1)

	IP	a1	a2
1			
2			
3			

ID	a1	a2	IP
o1	6	3	-6
o2	3	5	-3
o3	7	5	-7
o4	5	8	-5
o5	4	6	-4

	IP	a1	a2	
1	-3	3	7	(o9)
2	-4	4	9	(o8)
3	-7	7	8	(o6)

ID	a1	a2	IP
o6	7	8	-7
o7	8	3	-8
o8	4	9	-4
o9	3	7	-3
o10	8	5	-8

	IP	a1	a2	
1	-3	3	5	(o2, 1)
2	-4	4	6	(o5, 1)
3	-5	5	8	(o4, 1)

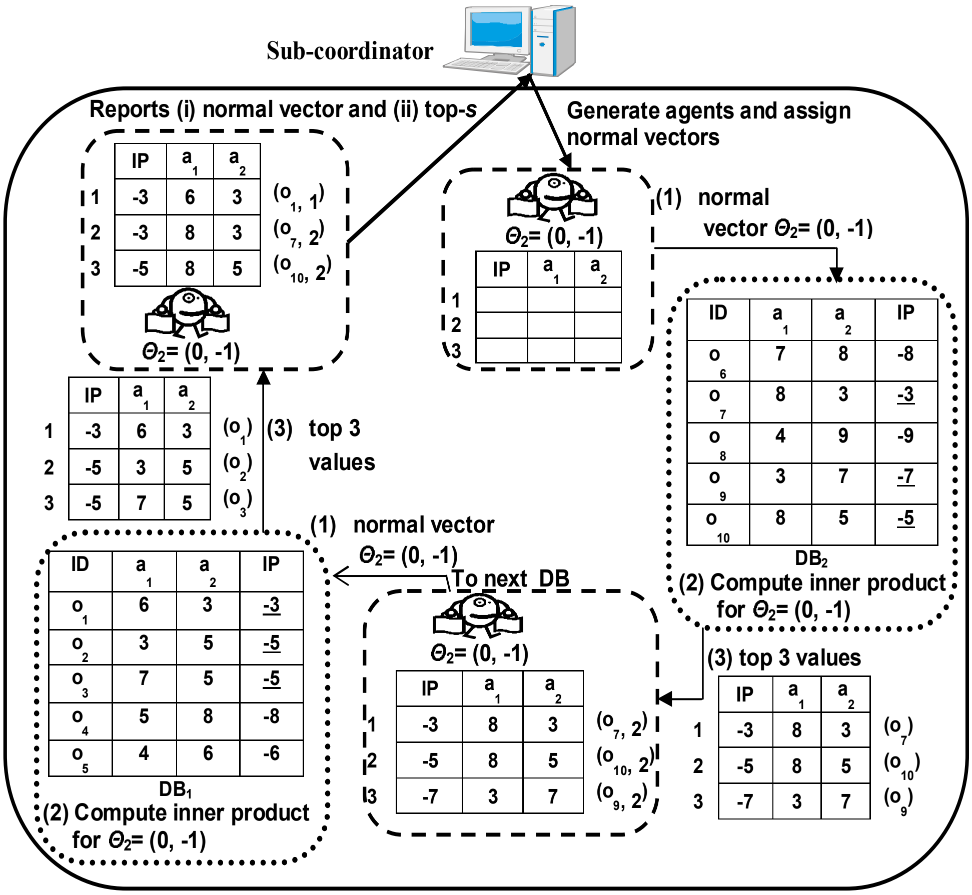
	IP	a1	a2	
1	-3	3	5	(o2)
2	-4	4	6	(o5)
3	-5	5	8	(o4)

Figure 3.11: Computation in group one of cluster one with $\Theta_1 = (-1, 0)$

also known as "heap data structure" that keeps the top-3 inner product values and their corresponding record values.

Figure 3.11 shows the computation process of the agent with normal vector $\Theta_1 = (-1, 0)$ in group 1 of cluster 1. When an agent arrives at a database of a group, it sends the normal vector of the database. Next, the database computes inner product for each record of the database. Finally, the database pushes the local top-3 records along with inner product values to the agent. Note that during this computation the database cannot see the contents of the priority queue of the agent.

In the example of Figure 3.11, it is observed DB_1 pushes three triplets, i,e., inner product value (IP), a_1 value, and a_2 value, (IP, a_1, a_2) = ((-3, 3, 5), (-4, 4, 6), (-5, 5, 8)) to the agent. Next, the agent goes to DB_2. DB_2 pushes (IP, a_1, a_2) = ((-3, 3, 7), (-4, 4, 9), (-7, 7, 8)) to the agent. After visiting all databases in the cluster, agent with normal


Sub-coordinator
Reports (i) normal vector and (ii) top-s
Generate agents and assign normal vectors
Θ_2= (0, -1)
(1) normal vector Θ_2= (0, -1)
(2) Compute inner product for Θ_2= (0, -1)
(3) top 3 values
To next DB
DB$_1$
DB$_2$

	IP	a_1	a_2	
1	-3	6	3	(o_1, 1)
2	-3	8	3	(o_7, 2)
3	-5	8	5	(o_{10}, 2)

	IP	a_1	a_2
1			
2			
3			

ID	a_1	a_2	IP
o_6	7	8	-8
o_7	8	3	-3
o_8	4	9	-9
o_9	3	7	-7
o_{10}	8	5	-5

	IP	a_1	a_2	
1	-3	6	3	(o_1)
2	-5	3	5	(o_2)
3	-5	7	5	(o_3)

ID	a_1	a_2	IP
o_1	6	3	-3
o_2	3	5	-5
o_3	7	5	-5
o_4	5	8	-8
o_5	4	6	-6

	IP	a_1	a_2	
1	-3	8	3	(o_7, 2)
2	-5	8	5	(o_{10}, 2)
3	-7	3	7	(o_9, 2)

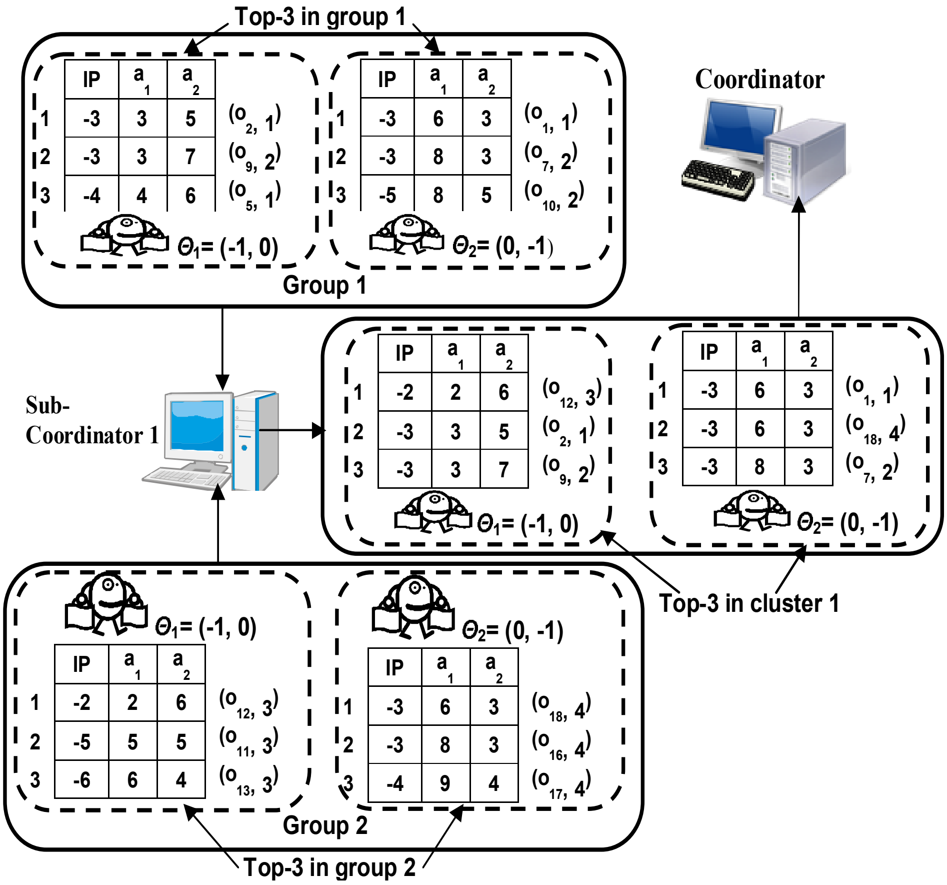
	IP	a_1	a_2	
1	-3	8	3	(o_7)
2	-5	8	5	(o_{10})
3	-7	3	7	(o_9)

Figure 3.12: Computation in group one of cluster one with $\Theta_2 = (0, -1)$

vector $\Theta_1 = (-1, 0)$ contains (IP, a_1, a_2) = ((-3, 3, 5), (-3, 3, 7), (-4, 4, 6)) in its priority queue and returns back to the sub-coordinator. During the top-3 computation the agent keeps track about the owner of each of the three objects. As for example, we can see that two objects are from DB_1 and one object from DB_2 in Figure 3.11.

Figure 3.12 shows similar computation in the group 1 of cluster 1 with normal vector $\Theta_2 = (0, -1)$. Here, in order to minimize idle time, the agent travels from DB_2 and goes to DB_1. After the computation, the agent contains (IP, a_1, a_2) = ((-3, 6, 3), (-3, 8, 3), (-5, 8, 5)) in its priority queue and goes to the sub-coordinator and reports the results.

During these processes, the sub-coordinator computes the local top-3 in the group 2 simultaneously. After the agent-based computation in two groups, the sub-coordinator merges the local top-3 priority queues for each normal vector. Figure 3.13 shows the



	IP	a_1	a_2	
1	-3	3	5	(o_2, 1)
2	-3	3	7	(o_9, 2)
3	-4	4	6	(o_5, 1)

	IP	a_1	a_2	
1	-3	6	3	(o_1, 1)
2	-3	8	3	(o_7, 2)
3	-5	8	5	(o_{10}, 2)

	IP	a_1	a_2	
1	-2	2	6	(o_{12}, 3)
2	-3	3	5	(o_2, 1)
3	-3	3	7	(o_9, 2)

	IP	a_1	a_2	
1	-3	6	3	(o_1, 1)
2	-3	6	3	(o_{18}, 4)
3	-3	8	3	(o_7, 2)

	IP	a_1	a_2	
1	-2	2	6	(o_{12}, 3)
2	-5	5	5	(o_{11}, 3)
3	-6	6	4	(o_{13}, 3)

	IP	a_1	a_2	
1	-3	6	3	(o_{18}, 4)
2	-3	8	3	(o_{16}, 4)
3	-4	9	4	(o_{17}, 4)

Figure 3.13: Merging process at cluster one

merge process of cluster 1.

Algorithm 3.1 shows the procedure for local top-*s* computation within a cluster. First, it creates necessary groups and divides the databases among the groups (lines 5-12). Next, it computes top-*s* values for each normal vector from the groups (lines 13-16). Finally, it calculates local top-*s* for the normal vectors in the cluster (line 17).

3.5.2.2 Global Merging

After the computation of local top-*s* for a normal vector in a cluster, the result is sent to the coordinator for global merging. In global merging, local top-*s* results from the clusters are merged to obtain global top-*s* results. For example, local top-3 results corresponds to each normal vector from two clusters of Figure 3.10 are merged into global top-3 as shown in Figure 3.14. These merge processes are carried out among

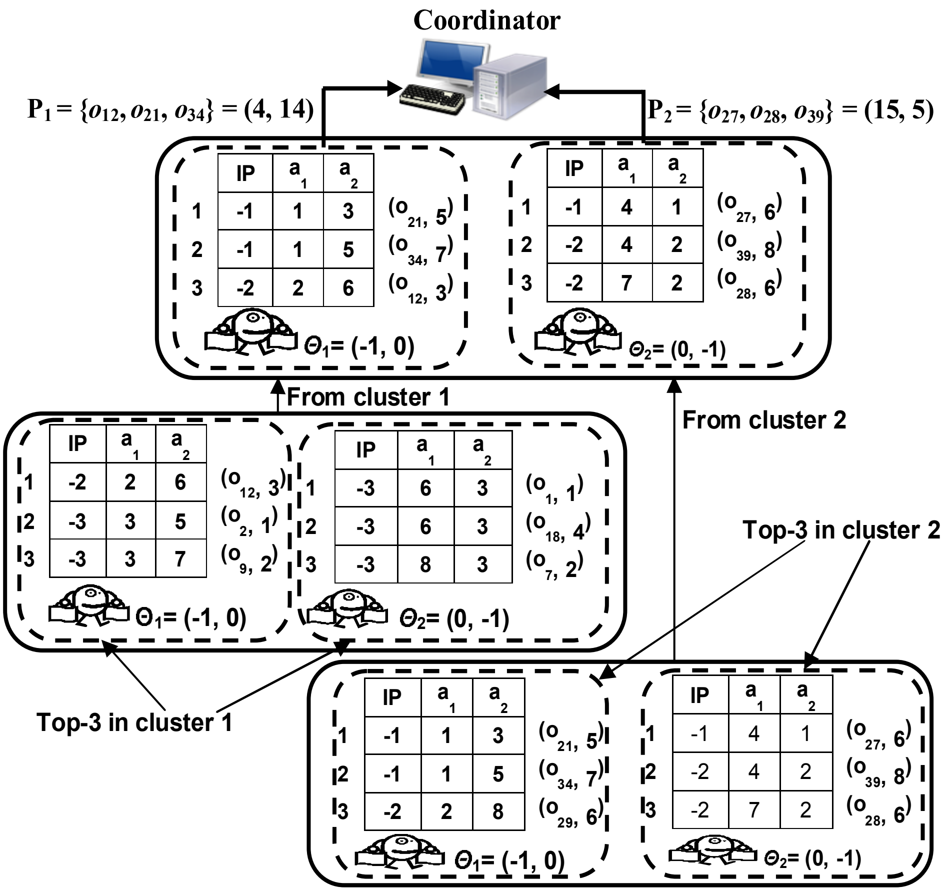
Algorithm 3.1 ComputationWithinCluster

```
Input: Normal vectors $\theta$s, set size $s$
Output: Local top-$s$ for each normal vector
begin
Let $(\theta_1, \theta_2, \cdots, \theta_v)$ be the given $v$ normal vectors, $(DB_1, DB_2, \cdots, DB_z)$ be $z$ databases in the cluster
if $z < v$ then
    Create $z$ groups
else
    Create $v$ groups
end if
for each $i$ ($i$ = 1 to $z$) do
    Assign $DB_i$ to group ($i$ % $v$)
end for
for each group do
    Create agents $ag(\theta_t)$, $(1 \le t \le v)$ // Each $ag(\theta_t)$ is an agent with $\theta_t$
    $ag(\theta_t)$ travels $DBs$ in the group and compute top-$s$ of the group
end for
Sub-Coordinator receives top-$s$ of $\theta_t$, $(1 \le t \le v)$ for each group and computes top-$s$ of $\theta_t$ in the cluster
end
```

the agents so that the coordinator does not see the individual record's values of the agents. The agent then returns the aggregated values to the coordinator.

From the example of Figure 3.9, we get 3-set corresponds to normal vector (-1, 0) is $\{o_{12}, o_{21}, o_{34}\}$, whose coordinate values in the two-dimensional space is (4, 14) and the 3-set corresponds to normal vector (0, -1) is $\{o_{27}, o_{28}, o_{39}\}$ whose coordinate values is (15, 5).

The global merging procedure is given in **Algorithm 3.2**. The algorithm first computes global top-s from local top-s of the clusters (lines 4-6). Next, it calculates aggregated values of attributes of top-s and returns the result to the coordinator (lines 7-10).



	IP	a_1	a_2	
1	-1	1	3	(o_{21}, 5)
2	-1	1	5	(o_{34}, 7)
3	-2	2	6	(o_{12}, 3)

	IP	a_1	a_2	
1	-1	4	1	(o_{27}, 6)
2	-2	4	2	(o_{39}, 8)
3	-2	7	2	(o_{28}, 6)

	IP	a_1	a_2	
1	-2	2	6	(o_{12}, 3)
2	-3	3	5	(o_2, 1)
3	-3	3	7	(o_9, 2)

	IP	a_1	a_2	
1	-3	6	3	(o_1, 1)
2	-3	6	3	(o_{18}, 4)
3	-3	8	3	(o_7, 2)

	IP	a_1	a_2	
1	-1	1	3	(o_{21}, 5)
2	-1	1	5	(o_{34}, 7)
3	-2	2	8	(o_{29}, 6)

	IP	a_1	a_2	
1	-1	4	1	(o_{27}, 6)
2	-2	4	2	(o_{39}, 8)
3	-2	7	2	(o_{28}, 6)

Figure 3.14: Merging process at the coordinator

3.5.2.3 Facet Expansion

After receiving all surrounding points of a facet, the coordinator computes the normal vector of the facet by using the surrounding points. In the example, $P_1 = (4,14)$, which is found by normal vector $\theta_1 = (-1,0)$ and $P_2 = (15,5)$, which is found by normal vector $\theta_2 = (0,-1)$, are two surrounding points of the initial facet (line segment between P_1 and P_2). The coordinator computes the normal vector $\theta_{1,2} = (-9,-11) = (-(14-5),(4-15))$ from the facet as shown in Figure 3.15. Then, the normal vector $\theta_{1,2} = (-9,-11)$ is sent to the sub-coordinator of each cluster and the agent-based parallel touching oracle finds $P_{1,2} = (9,6)$, which is composed of $\{o_{21}, o_{27}, o_{39}\}$. This point expands the initial facet (line segment between P_1 and P_2) into two facets, which are the line segment between P_1 and $P_{1,2}$ and the line segment between $P_{1,2}$ and P_2 as shown in Figure 3.15.

Algorithm 3.2 GlobalMerging

```
Input: Local top-$s$ of $\theta_t$, $(1 \le t \le v)$ for each cluster
Output: Aggregated values correspond to each $\theta_t$, $(1 \le t \le v)$
begin
for each $t$ ($t$ = 1 to $v$) do
    Compute "global" top-$s$ of $\theta_t$ from "local" top-$s$ for each cluster
end for
for each $\theta_t$, $(1 \le t \le v)$ do
    Compute the summation of values of each dimension of top-$s$ objects
    return the aggregated values to the coordinator
end for
end
```

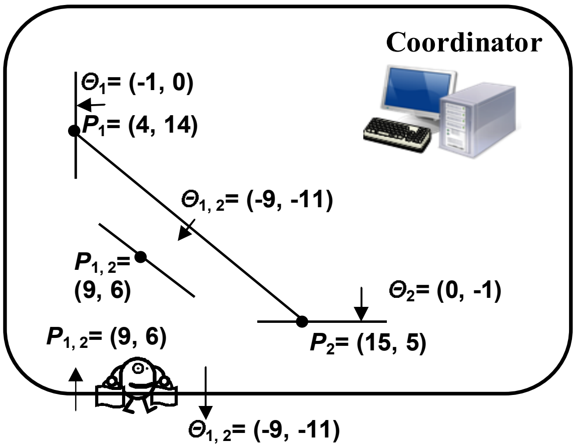
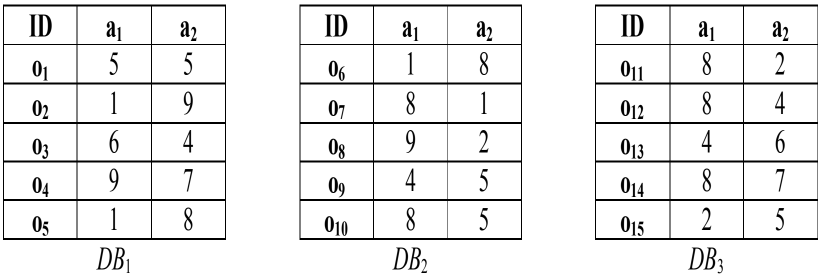


Figure 3.15: Facet expansion

We recursively compute tangent points for each of the expanded facet. If we find a new point outside the facet, we expand the facet further. We continually adopt the recursive operation while we can find new tangent point outside the facet. Finally, we can find all the points of convex skyline s-sets.

3.5.3 Compromisable Situations in Skyline Sets Queries

Let us assume that domains of the numerical attributes are specified. Let min_i and max_i be the minimum and maximum values of an attribute a_i, $(i = 1, \cdots, k)$. Let $value_i$ be the aggregated value of a_i in an s-set. We can say that a skyline s-set is a compromised



ID	a_1	a_2
o_1	5	5
o_2	1	9
o_3	6	4
o_4	9	7
o_5	1	8

DB_1

ID	a_1	a_2
o_6	1	8
o_7	8	1
o_8	9	2
o_9	4	5
o_{10}	8	5

DB_2

ID	a_1	a_2
o_{11}	8	2
o_{12}	8	4
o_{13}	4	6
o_{14}	8	7
o_{15}	2	5

DB_3

Figure 3.16: Example of three databases having compromisable 3-sets

Table 3.3: Some skyline 3-sets in the databases of Figure 3.16

ID	$\mathbf{a_1}$	$\mathbf{a_2}$
$\mathbf{Q_1}$	3	25
$\mathbf{Q_2}$	25	5
$\mathbf{Q_3}$	14	11
$\mathbf{Q_4}$	18	8
$\mathbf{Q_5}$	4	21

s-set if it contains at least one aggregated value $value_i$ such that we can find exact a_i's value of a member record of an s-set from $value_i$. Formally, we can say that a skyline s-set is a compromised if $(value_i - s * min_i) < s$ or $(s * max_i - value_i) < s$, in an attribute a_i, $(i = 1, \cdots, k)$. For example, assume that s = 3 and min_1 = 1. If we have an s-set whose $value_1$ is 4, one can find that three member values in a_1 are 1, 1, and 2. Note that this example satisfies the condition $(value_i - s * min_i) < s$. Similarly, if s = 4, max_1 = 5, $value_1$ = 17, we can find that there is at least one record whose a_i's value is 5 in the s-set.

Now, consider an example of three databases as shown Figure 3.16. We assume that domain of a_1 and a_2 are $[1 \cdots 9]$, i.e., $min_1 = min_2 = 1$ and $max_1 = max_2 = 9$. Table 3.3 shows partial results of skyline sets queries with s = 3. According to the definition of compromised s-sets, we can find that skyline 3-sets $Q_1 = (3, 25)$, $Q_2 = (25, 5)$, and $Q_5 = (4, 21)$ are compromised 3-sets. However, Q_3 and Q_4 are not compromised 3-sets.

We considered a perturbation method to prevent the compromised skyline s-sets. The coordinator performs the detection of compromised s-sets and perturbation of them using **Algorithm 3.3**. **Algorithm 3.3** first checks whether there is any com-

Algorithm 3.3 Perturbation

```
1:  Input: Skyline $s$-sets
2:  Output: Perturbed $s$-sets
3:  begin
4:  for each skyline $s$-set do
5:      for each attribute $a_i$ do
6:          if $((value_i - s * min_i) < s)$ then
7:              return $value_i = (s + s * min_i)$
8:          else if $((s * max_i - value_i) < s)$ then
9:              return $value_i = (s * max_i - s)$
10:         else
11:             return $value_i$
12:         end if
13:     end for
14: end for
15: end
```

promised value (line 6 and line 8) in the corresponding skyline s-set. If the algorithm detects a compromised value, it replaces the value with a new value (line 7 and line 9).

Now, consider compromised 3-sets Q_1, Q_2, and Q_5 of Table 3.3. From Q_1, we can see that it has a value 3 at attribute a_1 that satisfies line 6 of **Algorithm 3.3**. So, **Algorithm 3.3** replaces the value 3 with a value $(3 + 3 * 1) = 6$ (line 7). Like this, **Algorithm 3.3** replaces any value $value_i$ that satisfies the condition of line 6 with a value equal to $(s + s * min_i)$.

We can further observe that the value 25 of Q_1 at attribute a_2 satisfies line 8 of the **Algorithm 3.3**. So, the algorithm replaces the value 25 with a value (3 $*$ 9 - 3) = 24 (line 9). **Algorithm 3.3** performs the replacement of any such value with a value equal to $(s * max_i s)$. After such replacements Q_1 has values 6 and 24 in its attributes a_1 and a_2 respectively. Note that after replacement, from Q_1 no one can infer a member value exactly. Similarly, after perturbation, Q_2 and Q_5 are modified to (24, 6) and (6, 21), respectively and no one can exactly identify a member value from Q_2 or Q_5.

If we consider the distributed database example of Figure 3.9, we can find that skyline 3-sets correspond to normal vectors $\theta_1 = (-1, 0)$ and $\theta_2 = (0, -1)$ are $P_1 = (4, 14)$ and $P_2 = (15, 5)$, respectively. We can see that both P_1 and P_2 are compromised 3-sets.

Algorithm 3.4 Data Preprocessing

Input: Incomplete Data Set S, value $value_j$ that will replace missing values in attribute j. Here, $j = 1$ to k

Output: Complete Data Set T

```
begin
repeat
    read point $p_i$, $(1 \leq i \leq n)$ from input $S$
    for each $p_i$ do
        check every dimension $j$, $j = 1$ to $k$.
        replace each missing value with $value_j$.
    end for
until end of input $S$
end
```

ID	a_1	a_2
o_1	-	1
o_2	1	3
o_3	2	-
o_4	-	3
o_5	6	2

DB_1

ID	a_1	a_2
o_6	1	-
o_7	-	1
o_8	3	1
o_9	2	-
o_{10}	-	1

DB_2

Figure 3.17: Example of two databases with missing values

So, **Algorithm 3.3** modifies the values of P_1 and P_2 to (6, 14) and (15, 6), respectively. For preserving individual's privacy, the coordinator sends the perturbed 3-sets (6, 14) and (15, 6) to the user instead of original 3-sets (4, 14) and (15, 5). The coordinator uses the original values of skyline s-sets for facet expansion. As for example, the coordinator uses (4, 14) and (15, 5) for facet expansion instead of perturbed 3-sets (6, 14) and (15, 6).

3.6 Dealing with Missing Values

Our above approach of skyline sets queries consider that there are no missing values in the attributes of the database. However, such an assumption of completeness is not practical in many cases. For example, consider a movie rating application where each

ID	**a$_1$**	**a$_2$**
o_1	**10**	1
o_2	1	3
o_3	2	**10**
o_4	**10**	3
o_5	6	2

DB$_1$

ID	**a$_1$**	**a$_2$**
o_6	1	**10**
o_7	**10**	1
o_8	3	1
o_9	2	**10**
o_{10}	**10**	1

DB$_2$

Figure 3.18: Example of two databases without missing values

Table 3.4: Convex Skyline 3-sets in the databases of Figure 3.18

ID	**a$_1$**	**a$_2$**
P$_1$	6	23
P$_2$	23	6
P$_3$	10	6
P$_4$	6	14
P$_5$	14	6

user rates movies from thousands of movies. It is highly unlikely that every single user will rate all movies. Instead, a user will rate only the movies that interest her / him. As a result, each movie will be represented as a D-dimensional point with several blank (i.e., missing) dimensions. Another example is from the hotel application where some hotels may not disclose some of their properties. These undisclosed properties are represented as incomplete entries within the hotel multi-dimensional point representation. In such a situation, we need some preprocessing tasks before applying skyline sets query algorithm. This is described below.

In data preprocessing phase, we first search the databases for obtaining missing values of the records in each attribute. Then, we replace the missing values of the records in an attribute with a value outside the domain values. The choice of such a value for an attribute depends on the nature of the data in that attribute. If smaller values are better for an attribute, we shall replace missing values in that attribute with a value larger than the domain value and vice versa. **Algorithm 3.4** shows such replacement procedure.

Figure 3.17 shows two databases, each of which contains two attributes with some missing values in each attribute. Let us assume that smaller values in each attribute are preferable and the value domain in each attribute is 0 to 9. Using **Algorithm 3.4**, we can replace the missing values of both attributes of both databases with a value, say 10 as it is out side the value domain of each attribute and it is larger than any value of the attributes.

After replacement, we obtain the databases with no missing values as shown in Figure 3.18. From Figure 3.18, we can see that now there is no missing values. After obtaining such a table, we can apply the skyline sets query and perturbation algorithms and can return the results as shown in Table 3.4 to the user.

Note that in our result of Table 3.4, some of the skyline 3-sets contain inserted value in an attribute. We do not need any postprocessing of such s-sets to remove the inserted value. This is because if someone wants to hide some attributes' values of some records they should not get priority in those attributes. Moreover, from Table 3.4 we can find that we perform perturbation before returning the results to the users.

3.7 Experiments

We have implemented the proposed parallel computation of the skyline sets queries in a distributed database using Java Agent Development Framework. We have performed the experiment in a simulation environment of fifty databases created by ten PCs running on windows OS and are connected by an Ethernet switch. Each of the PCs has an Intel(R) Core2 Duo, 2 GHz CPU, and 3 GB main memory. We evaluate our proposed privacy preserving skyline sets queries algorithm in distributed environment on synthetic datasets. As benchmark databases, we use the databases proposed by Borzsonyi, et al. [1], in which there are three types of synthetic data distributions: "correlated", "anticorrelated", and "independent". We consider data dimensionality between 2 to 5.

We first evaluate the effect of set size. Figure 3.19 shows the results of 2D, 3D, 4D, and 5D cases for datasets with 2500k data distributed among fifty databases. Databases are distributed among five clusters and each database contains around 50k data. We observe that with the increases of s, query time also increases. This is because as s increases, the number of sets in convex skyline also increases.

In the next experiment, we evaluate the effect of data size. We used data with cardinality 500k, 1000k, 1500k, 2000k, and 2500k. Same as the previous experiment fifty databases are distributed among five clusters and each cluster contains ten databases. In case of 500k, each database contains at least 10k data. Similarly, for datasets 1000k,

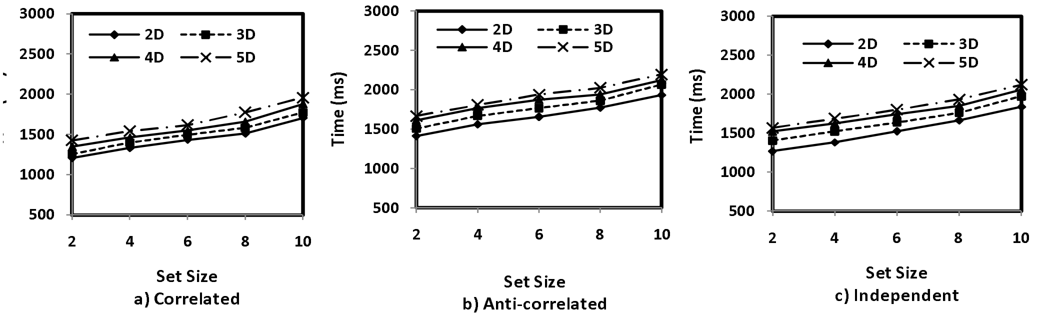


Figure 3.19: Time varying set size

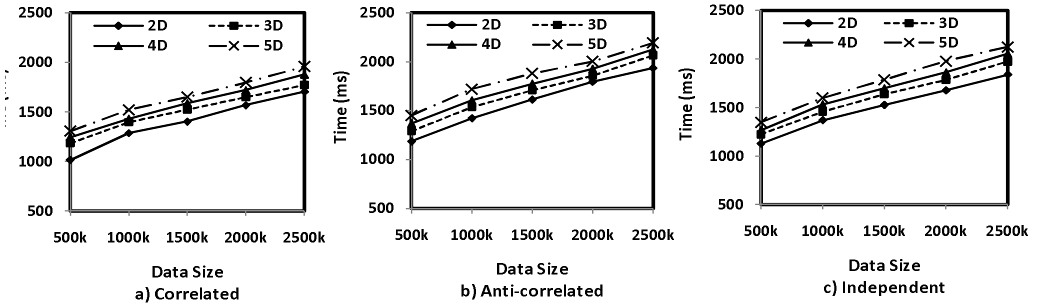


Figure 3.20: Time varying data size

1500k, 2000k, and 2500k each database has at least 20k, 30k, 40k, and 50k data respectively. In this experiment, we set *s* to 10. Figure 3.20 shows the results. In this experiment, it is observed that response time increases with the increase of data set size. It is also observed that response time gradually increases if the dimension increases.

Next, we conduct the experiment to examine the effect of the number of *DB*s in the computation process. In this experiment, we distribute 2500k data to m = 10, 20,

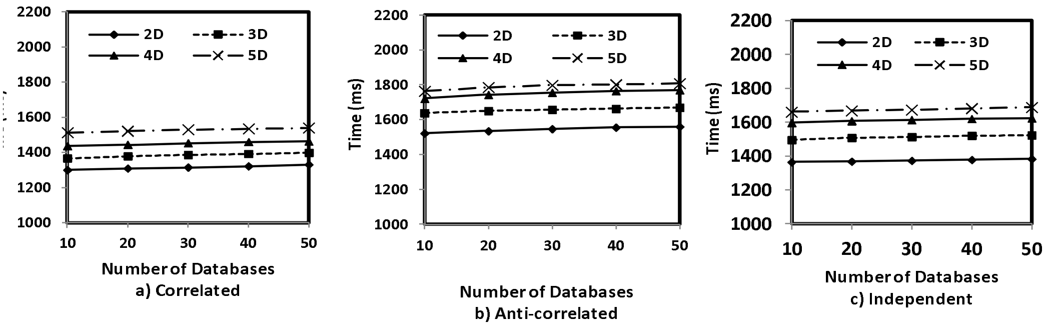


Figure 3.21: Time varying number of databases

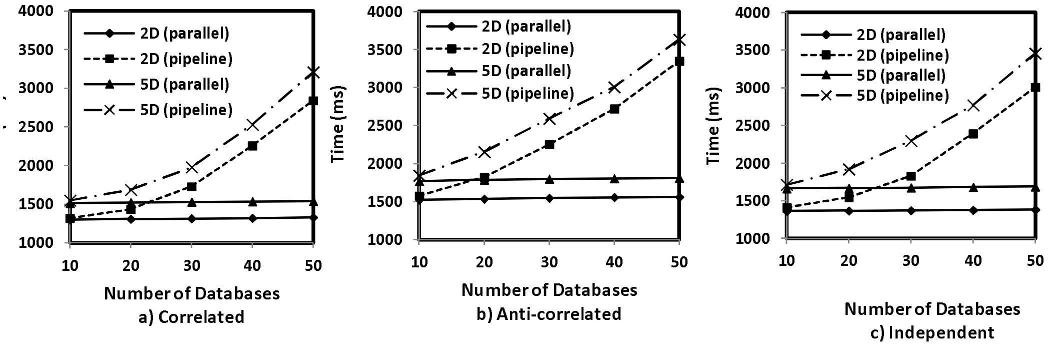


Figure 3.22: Comparison between parallel and pipeline computation

30, 40, 50 databases. Here, we fix the number of clusters to five. For 10, 20, 30, 40, and 50 databases, each cluster contains two , four, six, eight, and ten databases respectively. In this experiment, we set s to 4 and examine 2D, 3D, 4D, and 5D cases. Figure 3.21 shows the result. We find that the computation time is almost independent of the number of databases.

Finally, we conduct an experiment to examine the comparative performance of our method and pipeline computation method, a method where later agents need to wait for the completion of tasks by earlier agents that dramatically reduces the computation performance if there are many databases involved in the computation process. Same as the previous experiment, we distribute 2500k data to m = 10, 20, 30, 40, 50 databases. In this experiment, we set s to 4 and examine 2D and 5D cases. From the result of Figure 3.22, it is found that when the number of databases are relatively small the computation time is almost same in our method and pipeline computation method. However, as the number of databases increases, the pipeline execution method becomes slower while our proposed parallel computation method shows almost similar performance. The waiting delay of the later agents for the completion of tasks by the earlier agents is one reason of slowing down the computation. Another reason is that, in pipeline execution approach many databases become idle due to the lack

3.8 Conclusion

In privacy aware environment in which we are only allowed to disclose aggregated values of objects, skyline sets queries can be a promising alternative for analyzing and making important decisions. With the rapid growth of network infrastructure, distributed databases are becoming popular. In privacy aware environment, each owner of the distributed databases does not want to disclose the attribute's values of her/his databases to others. Therefore, we proposed an agent-based algorithm for computing skyline sets queries in a parallel manner from distributed databases in this chapter.

The proposed algorithm can efficiently calculate skyline sets from the distributed databases. Experimental results demonstrate that the proposed algorithm for skyline sets queries is scalable enough to handle large and high dimensional datasets. The performance of our proposed approach is almost independent of the number of databases involve in skyline sets queries. We have also proposed a privacy protection mechanism in which we detect compromisable sets and perturb such sets so that individual's records values cannot be identified. We additionally provided a solution of skyline sets queries from databases with missing values.

In this work, we assume that all the attributes of the databases are numerical. In future, we hope to develop parallel algorithms for skyline sets queries from databases with categorical attributes and from spatio-temporal databases. Moreover, we hope to compute subspace skyline sets queries.

Chapter 4

Selecting Spatial Objects

Recently, GPS devices and location based services become popular and we have databases containing spatial information. Considering this fact, in this chapter, we provide several approaches of selecting spatial objects by using skyline queries. Here, we provide three different approaches to compute spatial objects. First two methods of this chapter utilize the influences of the surrounding environments while selecting spatial objects. Third method computes spatial objects for a group of users considering both spatial and non-spatial features of the objects as well as the locations of the users.

This chapter is organized as follows. In Section 4.1, at first we present importance of considering surrounding environments during skyline computation. In this section, we also introduce the importance of skyline queries for groups. In Section 4.2, we provide a review of works related to our work in this chapter. Section 4.3 details the computation process of skyline considering the best value of each surrounding facility. In Section 4.4, we present the skyline query mechanism based on objects count. Section 4.5 describes skyline query mechanism for selecting spatial objects for a group of users. We conclude this chapter in Section 4.6.

4.1 Introduction

Conventional skyline queries select objects based on non-spatial attributes such as price and rating. With rapid growth of location-based services and geographic information systems, recently skyline queries for spatial databases [26, 27, 28, 29, 29, 39, 40, 41, 49, 77] have become an important research topic in many fields of computer science. Most of the current spatial skyline queries algorithms consider the distances from a user to the target spatial objects. There are many options for computing distances: the Euclidean distance, the distance in a road network, and a sophisticated adaptive distance.

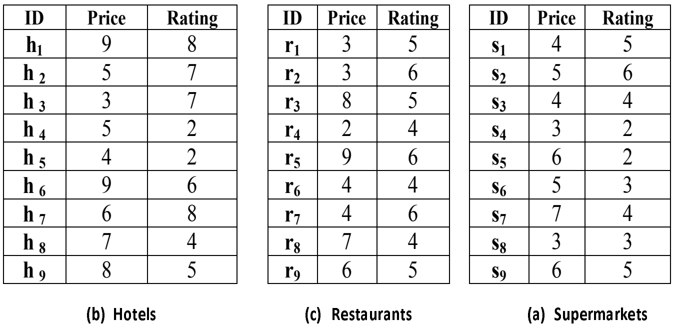
In general, surrounding environments of objects can play a vital role while selecting an object. Considering this fact, in this chapter, at first, we propose two methods to select spatial objects considering surrounding facilities. Both of our approaches utilize the concepts of skyline queries. Our first approach considers the best value in each attribute of each surrounding facility while our second method considers the objects count of each type of facility in the surrounding environment. Our first approach is well suited for hotel recommendation systems while our second approach is applicable for real estate recommendation.

In this chapter, we also consider the selection of spatial objects for a group of users. We consider this problem because there are situations where a group of users at different locations may want to choose a particular object that can fulfil the group's needs. For example, assume that members of a multidisciplinary task force team located at different offices want to put together in a restaurant to hold a lunch-on meeting. Conventional spatial skyline query cannot take into account the group's convenience.

4.2 Related Works

Spatial query processing was first studied for ranking neighboring objects. Several works [73, 74, 75] considered spatial query mechanism for ranking neighboring objects using the distance to a single query point. Papadias et al. [76] considered ranking of objects using aggregate distance of multiple query points.

Sharifzadeh et al. [26] first addressed the problem of spatial skyline queries. They proposed two algorithms, B^2S^2 and VS^2, for static query points and one algorithm, VCS^2, for the query points whose locations change over time. VCS^2 exploits the pattern of change in query points to avoid unnecessary re-computation of the skyline. The main limitation of VS^2 algorithm is that it can not deliver correct results in every situation. To overcome the limitation of VS^2 algorithm, Son et al. [27] presented a simple and efficient algorithm that can compute the correct results. Guo et al. [28] introduced the framework for direction-based spatial skyline computation that can retrieve nearest objects around the user from different directions. They also developed an algorithm to support continuous queries. However, their algorithm for direction-based spatial



ID	Price	Rating
h_1	9	8
h_2	5	7
h_3	3	7
h_4	5	2
h_5	4	2
h_6	9	6
h_7	6	8
h_8	7	4
h_9	8	5

(b) Hotels

ID	Price	Rating
r_1	3	5
r_2	3	6
r_3	8	5
r_4	2	4
r_5	9	6
r_6	4	4
r_7	4	6
r_8	7	4
r_9	6	5

(c) Restaurants

ID	Price	Rating
s_1	4	5
s_2	5	6
s_3	4	4
s_4	3	2
s_5	6	2
s_6	5	3
s_7	7	4
s_8	3	3
s_9	6	5

(a) Supermarkets

Figure 4.1: Non-spatial databases

skyline can not handle more than one query point. Kodama et al. [49] proposed efficient algorithms to compute spatial objects based on a single query point and some non-spatial attributes of the objects.

There are some considerations about spatial skyline computation in road networks. Deng et al. [39] first proposed multi-source skyline query processing in road network and proposed three different spatial skyline query processing algorithms for the computation of skyline points in road networks. In [40], Safar et al. considered nearest neighbour based approach for calculating skylines over road networks. They claimed that their approach performs better than the approach presented in [39]. Zhang et al. [77] proposed two distance-based skyline query techniques those can efficiently compute skyline queries over road networks. Huang al. [41] proposed a query processing method to produce spatial skylines for location-based services. They focus on location-dependent spatial queries (LDSQ) and consider a continually changing user location (query point). In their approach, it is not easy to decide how often the skyline result needs to be updated.

4.3 Spatial Objects Selection Considering Best Values of Surrounding Objects

4.3.1 Problem Formulation

Let us consider the spatial information of three different facilities as shown in Table 4.1. Also, consider the non-spatial attributes of the three facilities as shown in Figure 4.1. Figure 4.2 shows the location of these spatial objects in a map. In Figure 4.2, *h*,

Table 4.1: Spatial Database 1

ID	Longitude	Latitude	Type
h_1	5	1	Hotel
h_2	6	10	Hotel
h_3	10	2	Hotel
h_4	6	7	Hotel
h_5	1	2	Hotel
h_6	2	6	Hotel
h_7	9	6	Hotel
h_8	1	3	Hotel
h_9	6	1	Hotel
r_1	2	5	Restaurant
r_2	11	7	Restaurant
r_3	7	1	Restaurant
r_4	1	6	Restaurant
r_5	7	7	Restaurant
r_6	5	7	Restaurant
r_7	3	3	Restaurant
r_8	11	11	Restaurant
r_9	11	3	Restaurant
s_1	10	9	Supermarket
s_2	1	7	Supermarket
s_3	2	1	Supermarket
s_4	6	3	Supermarket
s_5	2	7	Supermarket
s_6	5	2	Supermarket
s_7	5	6	Supermarket
s_8	5	11	Supermarket
s_9	7	10	Supermarket

r, s represent hotel, restaurant, and supermarket, respectively. For simplicity, consider that smaller values in each non-spatial attribute is better and 1 unit Euclidean distance equivalent to 100 meters.

We used a grid-based data structure to keep the spatial information. Figure 4.3 shows the distribution of objects among nine grids $G_{11}, G_{13}, \cdots, G_{33}$. Each grid size is 1 square kilo meter.

4.3.2 Query Processing

For each grid of Figure 4.3, we pre-compute best value for each attribute as shown in Figure 4.4 of each facility and keep this information in the memory. In the table, $R - Price$ and $R - Rating$ represents best price and best rating of restaurants, respectively. Similarly, $S - Price$ and $S - Rating$ represents best price and best ratings of supermarkets, respectively. Note that there is no object in G_{31}. Therefore, we exclude records of the empty grids from the table.

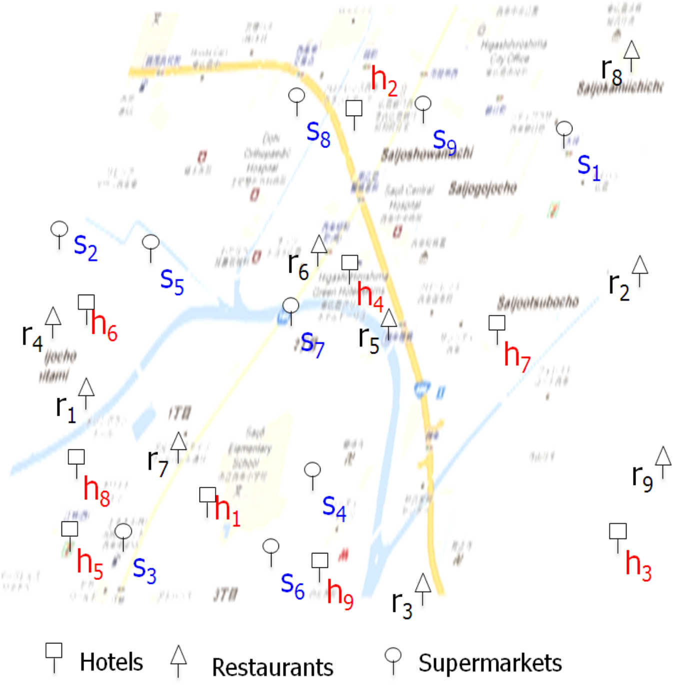


Figure 4.2: Location of spatial objects

Table 4.2: Hotel information of G_{11}

ID	$H-Price$	$H-Rating$	$R-Price$	$R-Rating$	$S-Price$	$S-Rating$
h_8	7	4	3	5	4	4
h_5	4	2	4	6	4	4

Assume that a user want to retrieve hotels in A city. The city A is covered by four grids, G_{11}, G_{12}, G_{21}, and G_{22} as shown in Figure 4.3.

We assume that the user prefers a hotel that has a good restaurant and/or a supermarket within 250 meters. For each grid, we collect hotels and their information of surrounding facilities as follows. In G_{11} of Figure 4.3, we can find there are two hotels h_5 and h_8. Now, consider the collection of surrounding facilities' information for h_8. The grid G_{11} has $R-Price = 4$, $R-Rating = 6$, $S-Price = 4$, $S-Rating = 4$. Since r_7 and s_3, which relate to the best values for restaurants and supermarkets of this grid, are within 1km from h_8, we set their values as the best values for h_8.

Since the grids G_{12}, G_{22}, and G_{21} are adjacent for h_8, we have to examine those adjacent grids. From Figure 4.4, we can see that best values in this grid for $R-Price$, $R-Rating$, $S-Price$, and $S-Rating$ are $8(r_3)$, $5(r_3)$, $2(s_4)$, and $3(s_4)$, respectively.

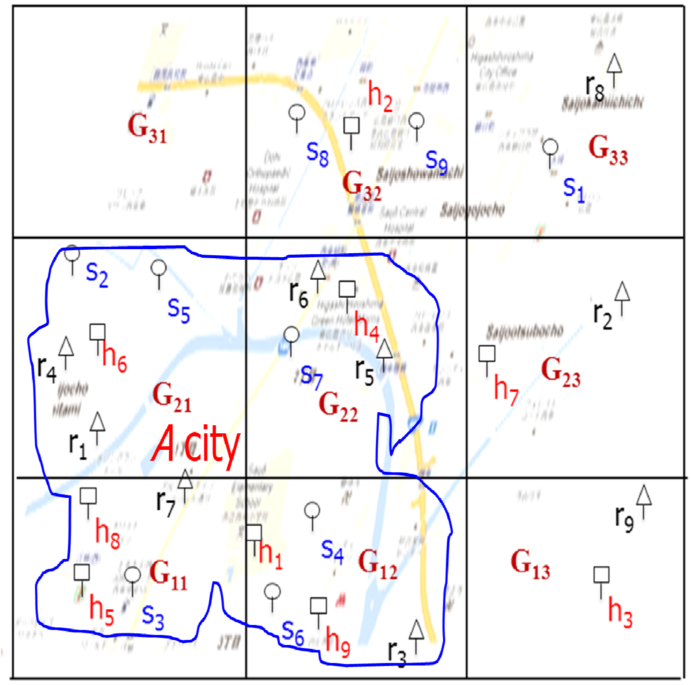
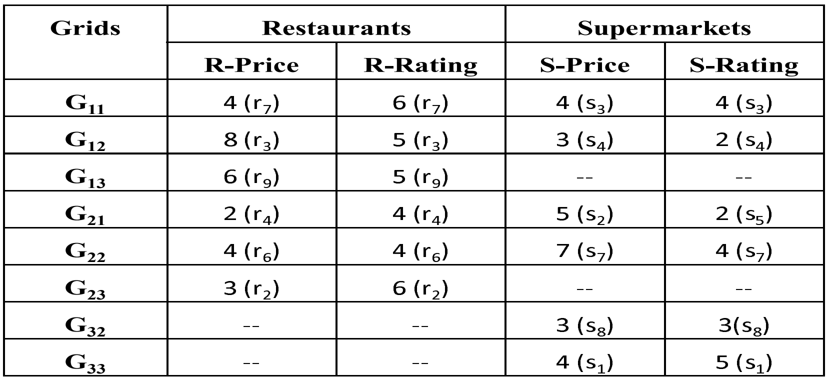


Figure 4.3: Grid wise distribution of spatial objects

Table 4.3: Hotel information of G_{12}

ID	$H-Price$	$H-Rating$	$R-Price$	$R-Rating$	$S-Price$	$S-Rating$
h_1	9	8	8	5	3	2
h_9	8	5	8	5	3	2

From these values, we find that $R-Price$, $R-Rating$ are worst than the corresponding current values for h_8. So, we do not need any further consideration of $R-Price$, $R-Rating$ of this grid. If we look at $S-Price$, and $S-Rating$, we can see that they are better than the current values of $S-Price$, and $S-Rating$ for h_8. We also find that both $S-Price$, and $S-Rating$ are related to s_4. Therefore, we need to compute the distance of s_4 from h_8. As the distance is greater than the user specified distance of 1 kilo meter, we cannot update the values in $S-Price$, and $S-Rating$ of h_8 with the values of s_4. As there is another supermarket s_6 in this grid which has $Price$, and $Rating$ 5 and 3, respectively. However, the distance of s_6 from h_8 is also more than 1 kilo meter. As a result the values for h_8 will as it is after visiting adjacent grid G_{12}. When we visit G_{22} we can see that none of the values for this grid is better than the corresponding values for h_8. Hence, there is no change in the values of h_8. After



Grids	Restaurants		Supermarkets	
	R-Price	R-Rating	S-Price	S-Rating
G_{11}	4 (r_7)	6 (r_7)	4 (s_3)	4 (s_3)
G_{12}	8 (r_3)	5 (r_3)	3 (s_4)	2 (s_4)
G_{13}	6 (r_9)	5 (r_9)	--	--
G_{21}	2 (r_4)	4 (r_4)	5 (s_2)	2 (s_5)
G_{22}	4 (r_6)	4 (r_6)	7 (s_7)	4 (s_7)
G_{23}	3 (r_2)	6 (r_2)	--	--
G_{32}	--	--	3 (s_8)	3(s_8)
G_{33}	--	--	4 (s_1)	5 (s_1)

Figure 4.4: Best values of grids

Table 4.4: Hotel information of G_{22}

ID	$H-Price$	$H-Rating$	$R-Price$	$R-Rating$	$S-Price$	$S-Rating$
h_4	5	2	4	4	7	4

visiting G_{21}, we can see that this grid has values $2(r_4)$, $4(r_4)$, $3(s_2)$, and $2(s_5)$, for $R-Price$, $R-Rating$, $S-Price$, and $S-Rating$, respectively. However, r_4, s_2, and s_5 are not within the user specified distance of 1 kilo meter. Hence, we cannot update the values of h_8 with any of these values. Now, we need further checking of other objects in this grid and find that r_1 is within the user specified distance that has values $Price$ = 3 and $Rating$ = 5 those are better than $R-Price$, and $R-Rating$ of h_8. Hence, the value in h_8 will be updated as $R-Price$ = 3, and $R-Rating$ = 5. As there is no more grid to check we can stop the computation for h_8 after traversing G_{21}. We can continue same procedure for hotels h_5, h_1, h_9, h_4 and h_6 as all these hotels are within city A.

Finally, we get the hotels' information in each of the four grids considering the surrounding objects as shown in Table 4.2, Table 4.3, Table 4.4, and Table 4.5.

Finally, making the union of the information of Table 4.2, Table 4.3, Table 4.4, and Table 4.5, we obtain Table 4.6. **Algorithm 4.1** shows the computation process of the surrounding environment information.

After obtaining Table 4.6, we use Sort Filter Skyline (SFS) [5] algorithm to obtain

Table 4.5: Hotel information of G_{21}

ID	$H-Price$	$H-Rating$	$R-Price$	$R-Rating$	$S-Price$	$S-Rating$
h_6	9	6	2	4	5	2

Table 4.6: Hotel information of City A

ID	$H-Price$	$H-Rating$	$R-Price$	$R-Rating$	$S-Price$	$S-Rating$
h_1	9	8	8	5	3	2
h_4	5	2	4	4	7	4
h_5	4	2	4	6	4	4
h_6	9	6	2	4	5	2
h_8	7	4	3	5	4	4
h_9	8	5	8	5	3	2

Table 4.7: Parameters and Values

Parameters	**Values**	**Default Value**
Raw data size of each facility	10k, 20k, 30k, 40k, 50k	20k
Types of surrounding facilities	1, 2, 3, 4, 5	2
Number of grids	50, 100, 150, 200, 250	100
Number of dimension of each object in each facility	2D, 3D, 4D, 5D	2D
Considerable distance of each surrounding facility from main facility in meters	200, 400, 600, 800, 1000	400

h_4, h_5, h_6, h_8, and h_9 as final skyline objects. Note that if we use conventional skyline query, only hotel h_5 is in the skyline. However, hotels h_4, h_6, h_8, and h_9 will not be in the skyline, although they have good surrounding facilities. However, in our approach all such types of hotels will be retrieved as skyline objects.

4.3.3 Experiments

We have simulated the proposed skyline queries algorithm in a Mac PC having Intel core i5 processor, 2.3 GHz CPU, and 4 GB main memory. We evaluated our skyline queries algorithm on synthetic datasets. As benchmark databases, we use the databases containing synthetic data with "anti-correlated" distribution. The parameters and values those have been used in our experiments are given Table 4.7.

Algorithm 4.1 Computation

Consideration: Consider that *cur* be the best value in the computation so far, *best* is the best value in an attribute of a facility in a grid G, *usd* be the user specified distance, *grd* is the distance between an object o_i of the main facility and an adjacent grid.

```
begin
for each grid $G$ adjacent for $o_i$ do
    for each surrounding facility $S_l$ do
        for each attribute $m_j$ of $S_l$ do
            find the best value best in $m_j$
            if ($cur \leq best$ ) then
                proceed to the next grid.
            else
                compute Euclidean distance $D$ between object $o_i$ and the objects that
                owns best
                if ($D \leq usd$) then
                    replace the value of cur with the best
                else if (($D > usd$) and ($grd < usd$)) then
                    compute the best value within usd
                    if (newly computed value $< cur$) then
                        replace the value cur with newly computed value
                    end if
                end if
            end if
        end for
    end for
end for
end
```

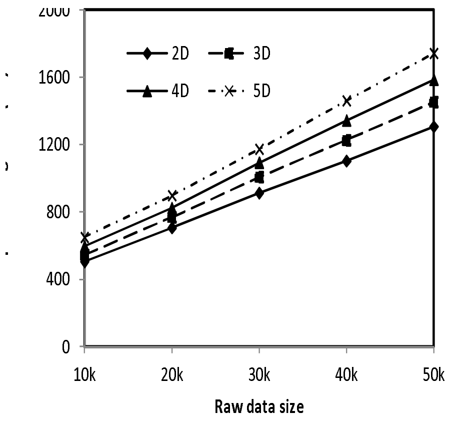


Figure 4.5: Preprocessing time

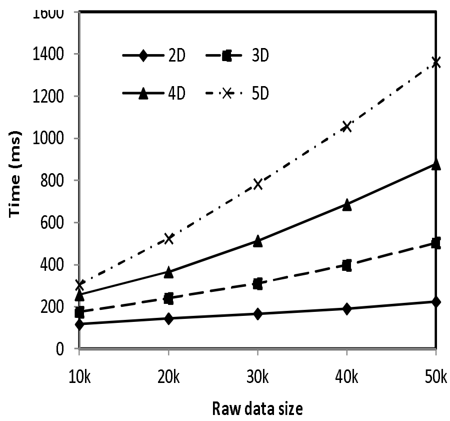


Figure 4.6: Time vs data size

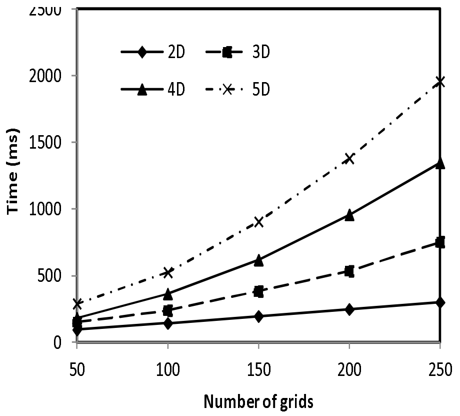


Figure 4.7: Time vs grids

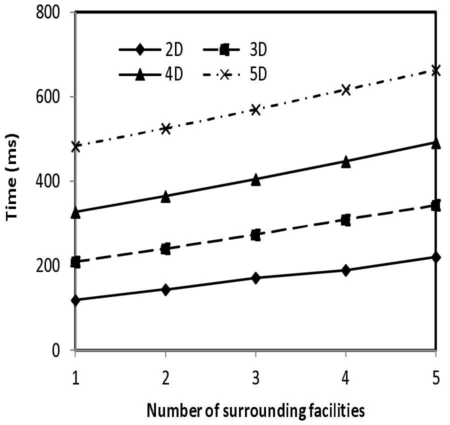


Figure 4.8: Time vs facilities

We first evaluate the cost of calculating best object of each facility in each dimension in each grid considering uniform distribution of data among 1600 grids and number of facilities seven. Figure 4.5 shows the results of 2D, 3D, 4D, and 5D cases. From the result, we observe that there is an increase in preprocessing time with the increase of data size. Also the preprocessing time increases with the increase in data dimensionality. As such computation is performed in off line, this will not effect the performance of our system.

In the next experiment, we evaluate the retrieval time of skyline results with varying data size. Figure 4.6 shows the results. In this experiment, it is observed that response time increases with the increase of data size. It is also observed that response time gradually increases if the dimension increases.

Next, we conduct the experiment to examine the effect of the number of grids in the

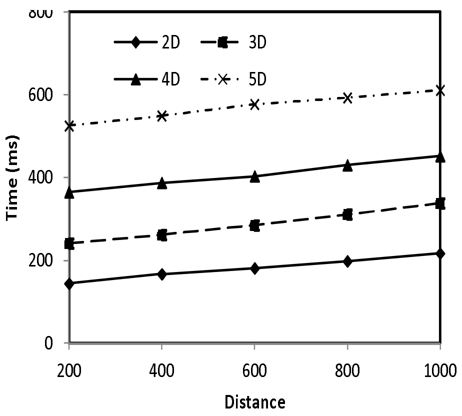


Figure 4.9: Time vs distance

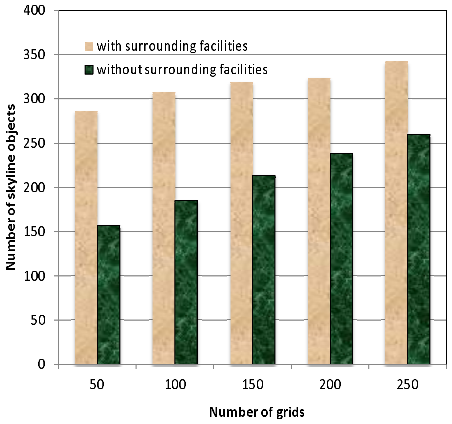


Figure 4.10: Comparative results

computation process. The result is shown in Figure 4.7. We find that the computation time increases with the increase of the number of grids. Also, there is an increase in computation time with increase in in data dimensionality.

Next experiment shows the effect of the number of surrounding facilities. Figure 4.8 shows the result. We find that the computation time increases with the increase in the number of surrounding facilities. This is because with the increase of surrounding facilities we need to consider more objects.

Our next experiment showed the effect of query time with varying distance between requested and surrounding facilities. The result is shown in Figure 4.9. From the figure, we find that the query time increases with the increase of distance between requested facility and surrounding facilities. This is because with the increase in distance, we need to consider more objects in skyline computation..

Our final experiment result is shown in Figure 4.10. It shows the comparative analysis of the retrieval of skyline points with and without considering surrounding facilities. Here, we can see that if we consider surrounding facilities more skyline objects are retrieved. Thus a user has more option in his decision making.

4.4 Spatial Objects Selection Based on Surrounding Objects Count

The second method of this chapter selects spatial objects, such as houses, based on the objects count in the surrounding areas. In our life, selecting a good house is very

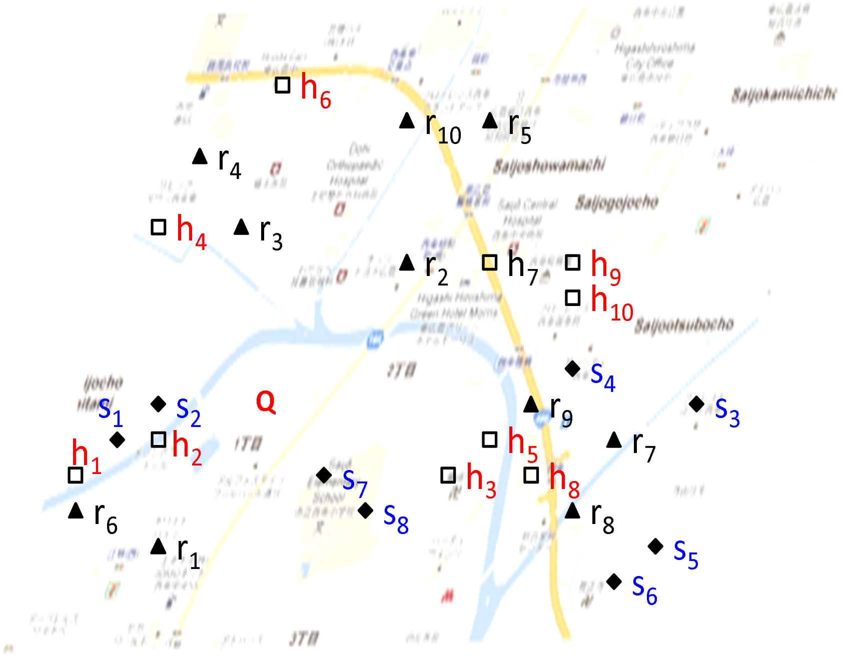


Figure 4.11: Three different facilities in a location

important for us. If we use the conventional skyline query for selecting houses, we can filter many dominated houses based on features of the houses, such as price, age, and so on. However, the location is a very important factor for selecting a house. For example, a house is convenient if there are many supermarkets within a walking distance. Motivated with such influences of surrounding facilities while selecting a house, in this section we provide a framework for selecting good houses based on the features of the houses as well as objects count in the surrounding areas.

In our method, a user specifies a list of surrounding facilities within a specified distance with favourable conditions for the objects. Similarly, a user can specify unfavourable conditions for the objects of the facilities if necessary. A user may specify the conditions of favourable and unfavourable.

For each specified surrounding facility, we count the number of objects those satisfy user defined distance and conditions. We, then, add a new attribute for each chosen surrounding facility that contains the number of objects those satisfies the conditions. Then, we compute the skyline result from the extended database.

For simplicity, in this section, we consider larger values are preferable and 1 unit distance Euclidean distance equals to 200 meters.

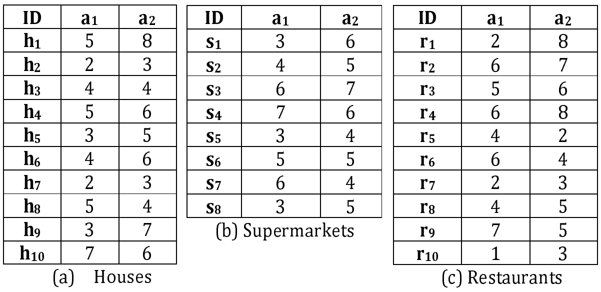
Table 4.8: Spatial Database 2

ID	Longitude	Latitude	Type
h_1	2	5	House
h_2	4	6	House
h_3	11	5	House
h_4	4	12	House
h_5	12	6	House
h_6	5	14	House
h_7	12	11	House
h_8	13	5	House
h_9	14	11	House
h_{10}	14	10	House
s_1	3	6	Supermarket
s_2	4	7	Supermarket
s_3	17	7	Supermarket
s_4	14	8	Supermarket
s_5	16	3	Supermarket
s_6	15	2	Supermarket
s_7	8	5	Supermarket
s_8	9	4	Supermarket
r_1	4	3	Restaurant
r_2	10	11	Restaurant
r_3	6	12	Restaurant
r_4	5	14	Restaurant
r_5	12	15	Restaurant
r_6	2	4	Restaurant
r_7	15	6	Restaurant
r_8	14	4	Restaurant
r_9	13	7	Restaurant
r_{10}	10	15	Restaurant

4.4.1 Problem Formulation

Let us consider the spatial information of three different facilities of Figure 4.11 as shown in Table 4.8 and non-spatial features information of these facilities as shown in Figure 4.12. We used a variant of *R*-tree index structure called *aR*-tree [57] to keep both spatial and non-spatial information of each facility.

Our method is based on four computation steps. First, a user specifies a place Q and distance (ε_1). Based on this information, at first we select spatial objects of the target facility like houses within the specified distance from Q. Second, the user specifies preferable surrounding facilities and conditions those influence the quality of the selected objects. We, then, count the number of such objects for each selected house. Third, we combine the count of surrounding facilities and the non-spatial information. . Finally, we perform skyline queries to select spatial objects from the combined table.



ID	a1	a2
h1	5	8
h2	2	3
h3	4	4
h4	5	6
h5	3	5
h6	4	6
h7	2	3
h8	5	4
h9	3	7
h10	7	6

(a) Houses

ID	a1	a2
s1	3	6
s2	4	5
s3	6	7
s4	7	6
s5	3	4
s6	5	5
s7	6	4
s8	3	5

(b) Supermarkets

ID	a1	a2
r1	2	8
r2	6	7
r3	5	6
r4	6	8
r5	4	2
r6	6	4
r7	2	3
r8	4	5
r9	7	5
r10	1	3

(c) Restaurants

Figure 4.12: Databases showing non-spatial features of three facilities

4.4.2 *aR*-Tree Indexing

The aggregation *R*-tree (*aR*-tree [57]) is an *R*-tree each node of which corresponds to minimum bounding rectangle (MBR) that contains objects in a plane. Figure 4.13 depicts the MBRs and corresponding *aR*-tree for the houses $h_1, \cdots, h_{10}$. A leaf node in the *aR*-tree contains objects and their corresponding information. An internal node contains the minimum value in each attribute of its descendent objects and total number of descendent objects. For example, in Figure 4.13, the left most leaf node contains the information of h_1. The parent node e_4, which is MBR e_4, contains two objects h_1 and h_2. The minimum values of e_4 in attributes a_1 and a_2 are 2 and 3, respectively. Therefore, the node has an entry $(e_4, 2, 3, 2)$. Similarly, the root node has an entry $(e_1, 2, 3, 10)$.

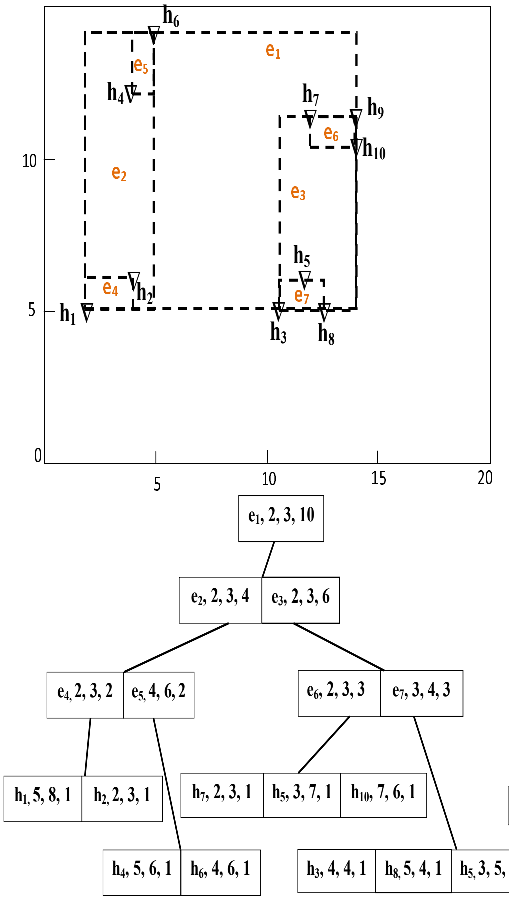
We can construct the *aR*-trees for restaurants and supermarkets as shown in Figure 4.14 and 4.15, respectively.

4.4.3 Computing Candidate Objects of Target Facility

We first select spatial objects of the target facility that are within the specified distance (ε_1) from a given query point. We call such spatial objects as "candidate objects". We are considering that the houses those are within 1200 meters from Q are "candidate objects". We can select candidate objects efficiently by using the *aR*-tree.

If a point p is given, we find a top most MBR that contains p and an internal node e that corresponds to the MBR. Let $mindist(p, e)$ and $maxdist(p, e)$ denote the minimum and maximum possible distance between p and any point in e.

In order to find objects that are within a user specified distance (ε_1) from a query point p, we first check $mindist(p, e_{root})$. If we find the $mindist(p, e_{root})$ is less than or



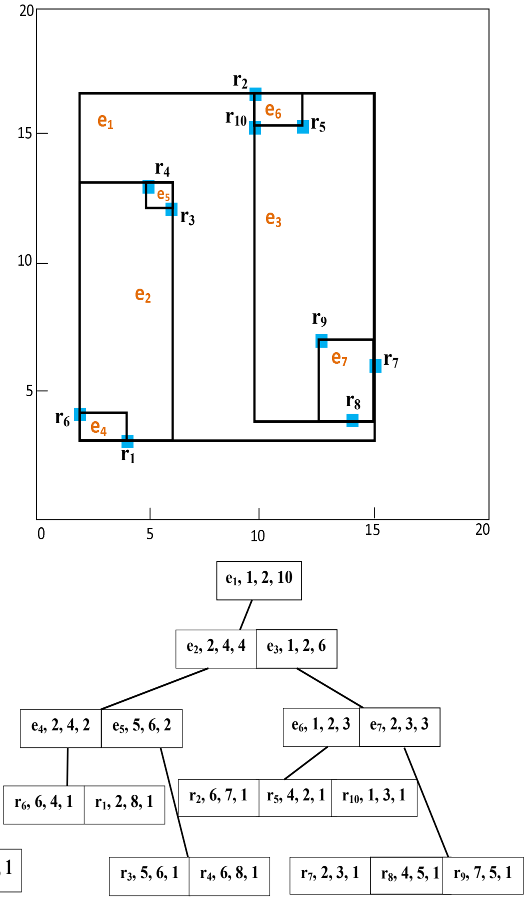


Figure 4.13: *aR*-tree for houses

Figure 4.14: *aR*-tree for restaurants

equal to ε_1, we recursively continue the searching the child nodes. An MBR having $mindist(p,e)$ larger than ε_1 will be pruned and will not be considered for the further processing. When we reach at a leaf node, we select spatial objects based on the distance from p.

Figure **??** shows the exploration of the nodes of the *aR*-tree when we search houses that are within 1200 meters from Q. From the *aR*-tree in Figure **??**, shaded rectangles satisfy the condition. In this step, we find h_1, h_2, h_3, h_4, h_5 as the candidate houses.

4.4.4 Calculating Surrounding Facility Count

Let us consider that a user specify S favourable surrounding facilities for selecting spatial objects. For selecting such spatial objects, we count the number of objects of the favourable facilities such as restaurants and supermarkets from each candidate

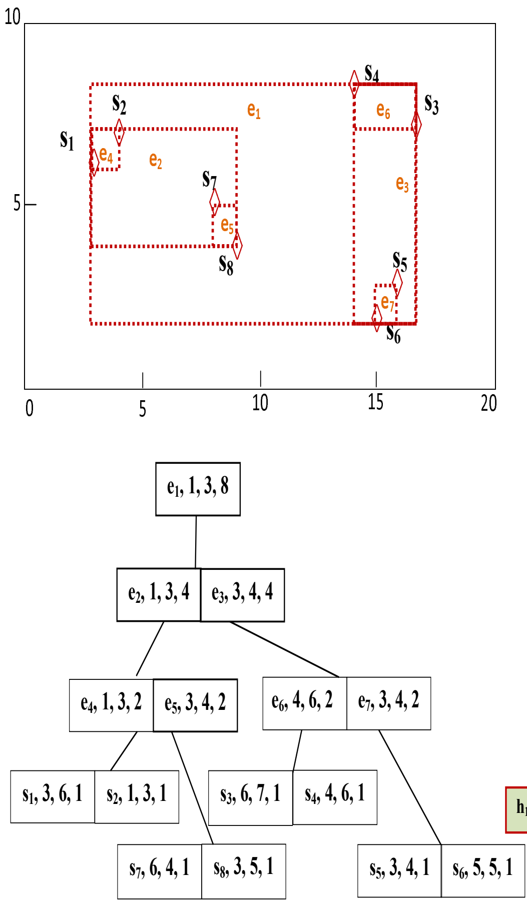


Figure 4.15: *aR*-tree for supermarkets

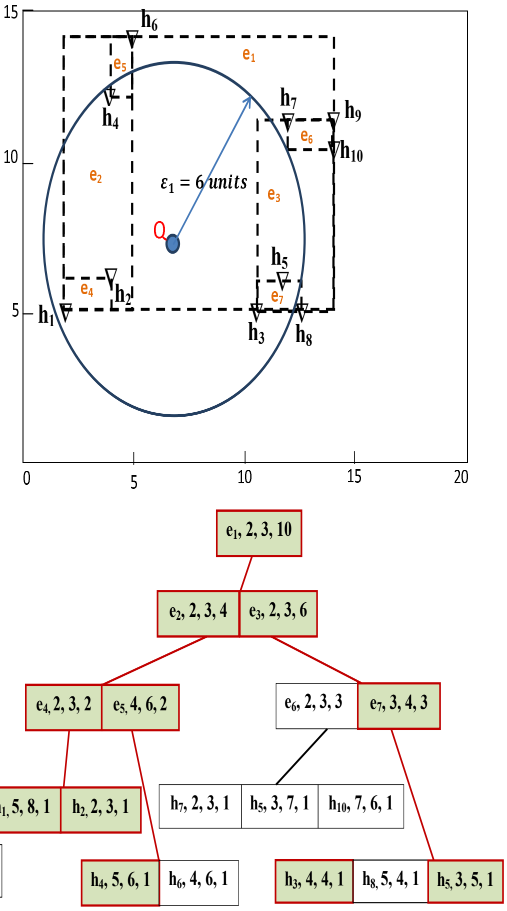


Figure 4.16: Candidate objects selection

object within the the user specified distance ε_2.

We use the *aR*-tree to compute the surrounding facilities count for each candidate object. For an object p in the candidate, we compute the surrounding facility count by traversing nodes from the root of the *aR*-tree. In a node e of the tree, (1) if $mindist(p,e) > \varepsilon_2$, we prune the subtree of the node. (2) If $maxdist(p,e) < \varepsilon_2$ and value in each attribute satisfies favourable condition, we increment the surrounding facility count by the node's count without traversing its subtree. (3) Otherwise, we recursively traverse each child of e.

Figure 4.17 and Figure 4.19 illustrate the computation process of "surrounding supermarkets count" for h_3. In this example, we assume that $\varepsilon_2 = 4$ and values in each attribute not less than 3 is favourable. The search procedure starts from the root e_1 of the *aR*-tree as shown in Figure 4.17. Since $mindist(h_3, e_1) = 3$ and $maxdist(h_3, e_1) =$

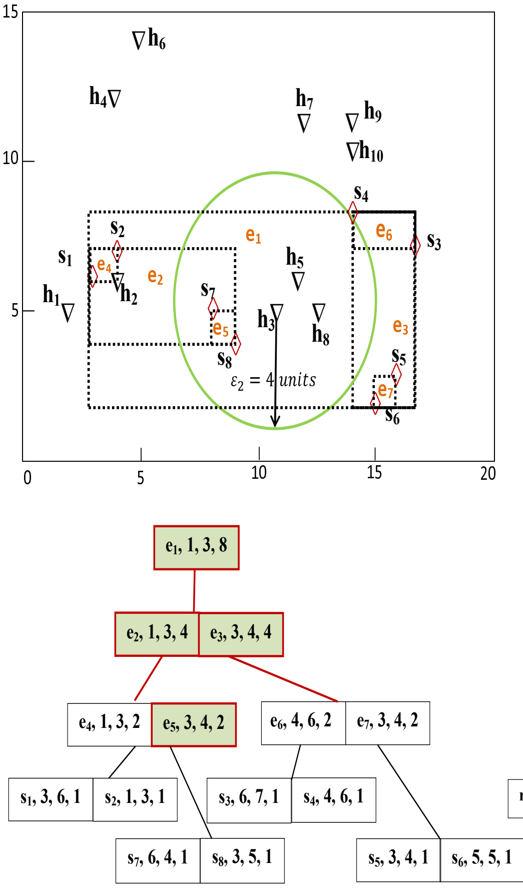


Figure 4.17: *aR*-tree search for supermarkets count

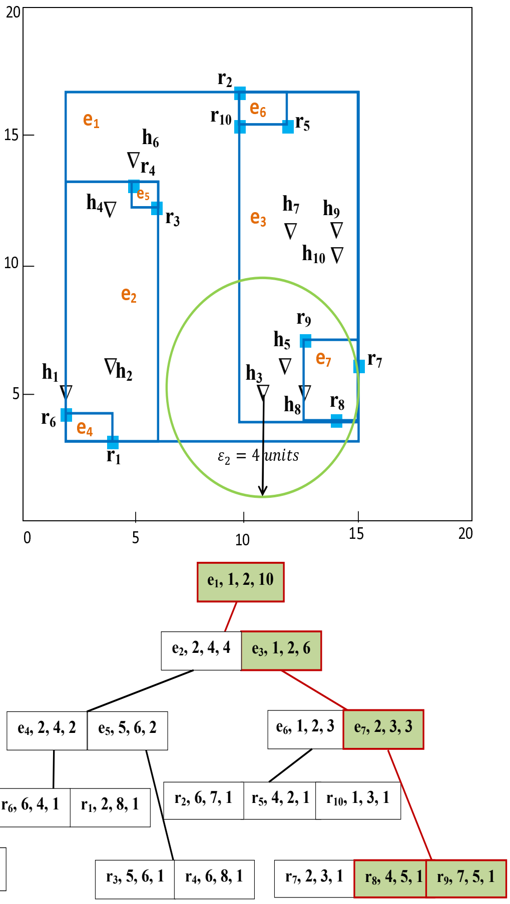


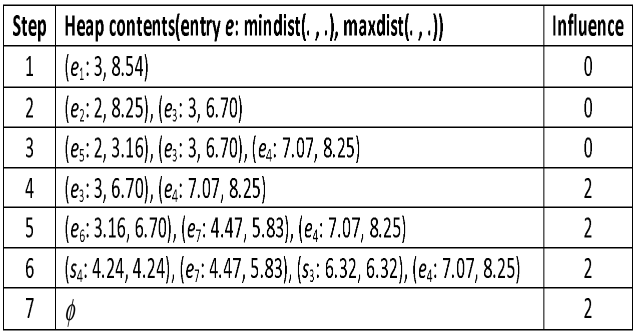
Figure 4.18: *aR*-tree search for restaurants count

8.54, we examine the children, e_2 and e_3. In e_2, we recursively examine the children and can find that e_5 satisfies the condition (2) i.e. $maxdist(h_3, e_5) = 3.16 < \varepsilon_2 = 4$ and there no object with value less than 3 in any of their attributes.

Therefore, we increment the count by 2. Note that we can skip the children, which are s_7 and s_8, of e_5. We continue the process from the next node similarly.

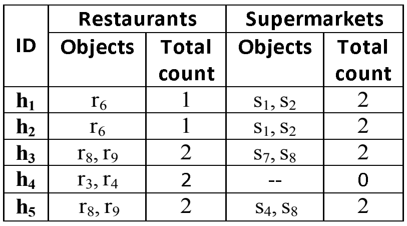
Figure 4.18 shows the tree structure for the search procedure for the restaurants count.

After this process, we get the counts of restaurants and supermarkets for each candidate object of target facility as shown in third and fifth columns of Figure 4.20.



Step	Heap contents(entry *e*: mindist(. , .), maxdist(. , .))	Influence
1	(e_1: 3, 8.54)	0
2	(e_2: 2, 8.25), (e_3: 3, 6.70)	0
3	(e_5: 2, 3.16), (e_3: 3, 6.70), (e_4: 7.07, 8.25)	0
4	(e_3: 3, 6.70), (e_4: 7.07, 8.25)	2
5	(e_6: 3.16, 6.70), (e_7: 4.47, 5.83), (e_4: 7.07, 8.25)	2
6	(s_4: 4.24, 4.24), (e_7: 4.47, 5.83), (s_3: 6.32, 6.32), (e_4: 7.07, 8.25)	2
7	ϕ	2

Figure 4.19: Computation process of the supermarket counts for h_3



ID	Restaurants		Supermarkets	
	Objects	Total count	Objects	Total count
h₁	r_6	1	s_1, s_2	2
h₂	r_6	1	s_1, s_2	2
h₃	r_8, r_9	2	s_7, s_8	2
h₄	r_3, r_4	2	--	0
h₅	r_8, r_9	2	s_4, s_8	2

Figure 4.20: Surrounding facilities satisfying both the distance and the condition

4.4.5 Combining Information and Generation of Final Result

After computing the count information, we extend the table of candidates by adding the count information. Table 4.9 is the example of the extended table of houses. In the table, there are five candidates of the target facility (house). First two numerical attributes represent their non-spatial attributes, while last two are for the count of restaurants and supermarkets. After obtaining such a table, we use Sort Filter Skyline (SFS) [5] algorithm to obtain h_1, h_3, h_4 and h_5 as final skyline objects.

Table 4.9: Database containing non-spatial information of target facility and surrounding information

SID	Price	Age	Restaurants-count	Supermarkets-count
h₁	5	8	1	2
h₂	2	3	1	2
h₃	4	4	2	2
h₄	5	6	2	0
h₅	3	5	2	2

Table 4.10: Parameters and Values

Parameters	**Values**	**Default Value**
Raw data size of each facility	20k, 40k, 60k, 80k, 100k	40k
Types of surrounding facilities	1, 2, 3, 4, 5	2
Number of dimension of each object in each facility	2D, 3D, 4D, 5D	2D
Considerable distance from query point to the target facility in meters	500, 1000, 1500, 2000, 2500	1000

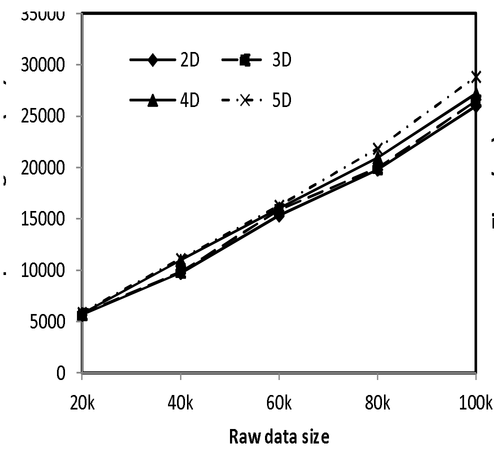


Figure 4.21: Cost for *aR*-tree indexing

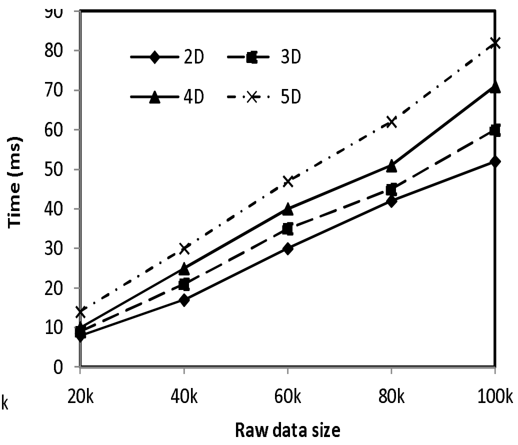


Figure 4.22: time vs data size

4.4.6 Experiments

The environmental setting for the experiment of this method is same as our previous method of this chapter. However, the parameters and values are different as shown in Table 4.10.

We first evaluate the cost of building the *aR*-tree index structure for for each facility. Figure 4.21 shows the results. Here, we consider 2D, 3D, 4D and 5D cases for each facility type and varied the data size for each facility from 20k to 100k. From the result, we observe that there is an increase in time in building the index structure with the increase of data size. Also the time increases with the increase in data dimensionality. As such index is built in off line, this will not effect the performance of our system.

In the next experiment, we evaluate the retrieval time of the results with varying

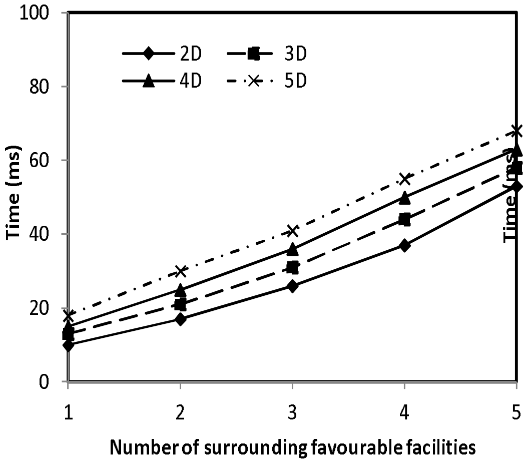


Figure 4.23: time vs facilities

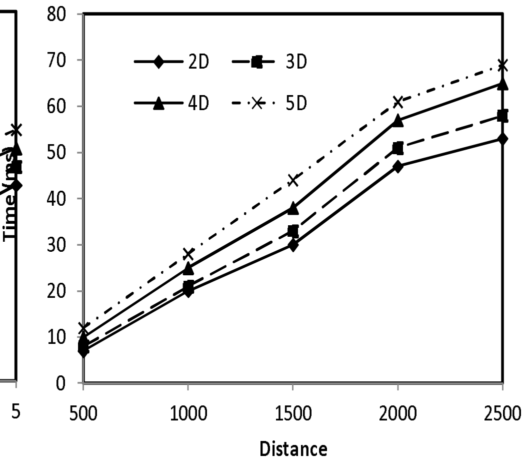


Figure 4.24: time vs distance

data size. Figure 4.22 shows the results. In this experiment, it is observed that response time increases with the increase of data size. It is also observed that response time gradually increases if the dimension increases.

Next, experiment shows the effect of the number of requested surrounding facilities. Figure 4.23 shows the result. We find that the computation time increases with the increase in the number of surrounding facilities as we need to consider more objects when there are more surrounding facilities.

Our next experiment shows the effect of query time with varying distance between query point and requested target facility. The result is shown in Figure 4.24. From the figure, we find that the query time increases with the increase of distance between query point and requested target facility. This is because with the increase in distance, we need to consider more objects in computation.

Our final experiment result is shown in Figure 4.25. It shows the comparative analysis of the retrieval of points with and without considering surrounding facilities. Here, we can see that if we consider surrounding facilities more objects are retrieved. Thus a user has more option in his decision making.

4.5 Spatial Objects Selection for a Group of Users

In this section, we consider the problem of selecting spatial objects for a group of users located at different positions by utilizing skyline queries since there are situations where a group of users at different locations may want to choose a particular object that

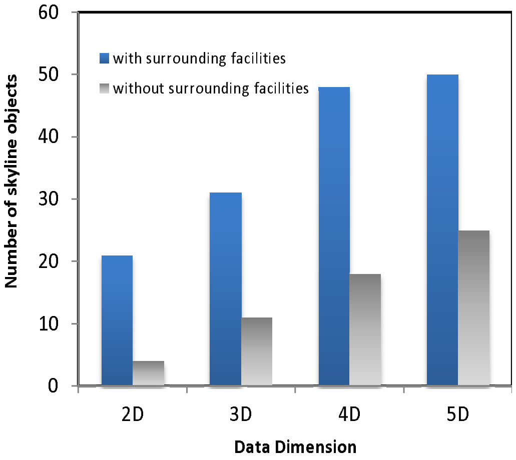


Figure 4.25: Comparative results

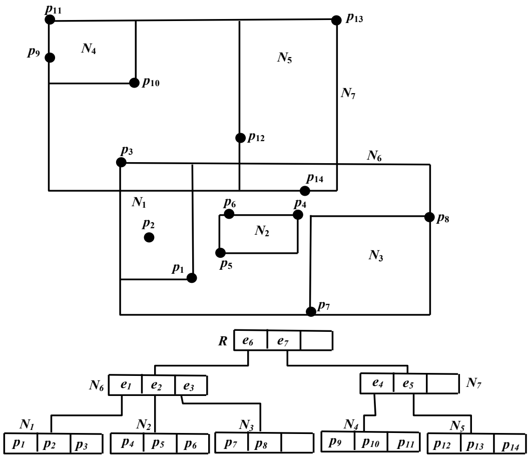


Figure 4.26: Example of an *R*-tree

can fulfil the group's needs. For example, assume that members of a multidisciplinary task force team located at different offices want to put together in a restaurant to hold a lunch-on meeting.

4.5.1 Preliminary Concepts

4.5.1.1 *R*-Tree

R-tree [59] is the most prominent index structure widely used for spatial query processing. Figure 4.26 shows an *R*-tree containing $P = \{p_1, \cdots, p_{14}\}$. We set the capacity of each node to three. The leaf nodes N_1, ..., N_5 store the coordinates of the grouped points

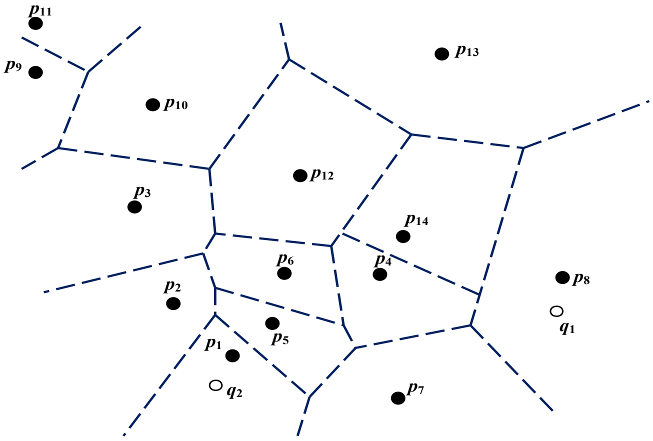


Figure 4.27: Example of a Voronoi diagram

together with optional pointers to their corresponding records. Each intermediate node contains the Minimum Bounding Rectangle (MBR) of the sub-tree of the nodes. For example, node e_1 corresponds to MBR N_1, which covers the points, p_1, p_2, and p_3. Similarly, node e_6 and node e_7 correspond to MBR N_6 and MBR N_7, respectively.

4.5.1.2 Voronoi Diagram

Let P is the set of n distinct data points on the plane. The Voronoi diagram of P is the subdivision of the plane into n cells. Each cell contains only one point of P, which is called the Voronoi point of the cell. We denote $V(p_j)$ as a cell of a Voronoi point p_j, $p_j \in P$, and $VN(p_j)$ as a set of cells that are adjacent to $V(p_j)$.

Assume that P contains fourteen data points $\{p_1, p_2, \cdots, p_{14}\}$ and two query points q_1 and q_2. Figure 4.27 shows the Voronoi diagram of the points in P. We can say that a query point is nearest to a data point if the query point is within Voronoi cell of the data point. As for example, from the Voronoi diagram of Figure 4.27, we can find that the nearest Voronoi point of the query point q_1 is p_8, since q_1 is within the Voronoi cell of p_8. Similarly, the nearest Voronoi point of query point q_2 is p_1.

Voronoi diagram provides an efficient data structure to compute the nearest Voronoi point for a given query point q. We use Fortune's algorithm [78] to construct Voronoi diagram for a set of points. Fortune's algorithm is a sweep line algorithm for generating a Voronoi diagram from a set of points in a plane. Though the worst time complexity for constructing Voronoi diagram for a set of n points using Fortune's algorithm is $O(n^2)$, the expected time complexity is $O(n \log n)$.

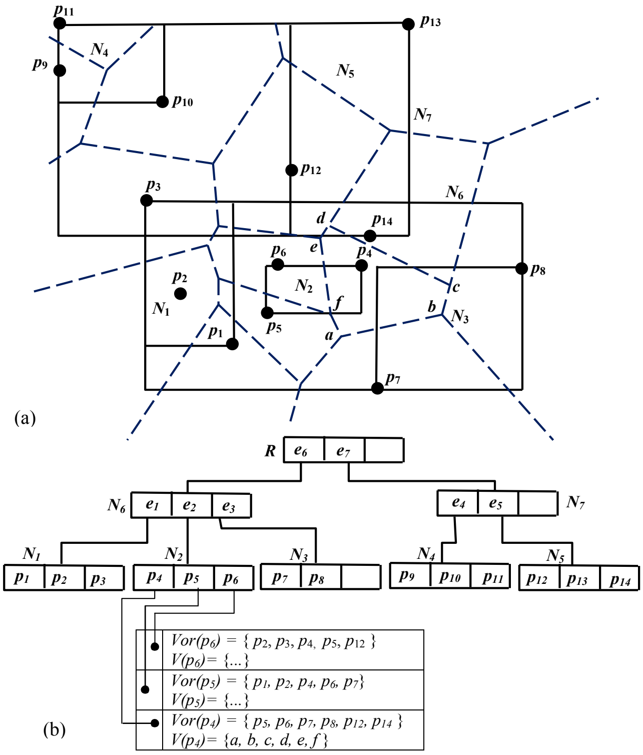


Figure 4.28: (a) Voronoi diagram (b) *VoR*-tree (adapted from [58])

4.5.1.3 *VoR*-Tree

A *VoR*-tree [58] is a variation of *R*-tree that index the data points using the concepts of Voronoi diagram and *R*-tree. Each leaf node stores a subset of data points. Each leaf node also includes the data records containing extra information about the corresponding points. In the record of a data point p_j in a *VoR*-tree, we store the pointer to the location of Voronoi neighbors $VN(p_j)$ and the vertices of $V(p_j)$, i.e., vertices of the Voronoi cell of p_j. Here, a vertex represents a common endpoint of two edges of a Voronoi cell.

For constructing *VoR*-tree, at first, we index the data points using an *R*-tree. Then, we use the Voronoi diagram of the data points to find the Voronoi neighbors and vertices of a Voronoi cell for each data point p_j. Next, we store both information as a record associated with each data point p_j. Each Voronoi neighbor of p_j in this record is a pointer to the disk block storing the information of that Voronoi neighbor. A disk block also known as a sector is a sequence of bytes for storing and retrieving data.

Figure 4.28(b) shows an example of *VoR*-tree for the data points of Figure 4.26.

Each rectangular in Figure 4.28 is a node of the *VoR*-tree. In Figure 4.28, rectangular N_2 contains three points, i.e., p_4, p_5, and p_6. N_2 and two other rectangular boxes N_1 and N_3 are contained by the parent, which is the rectangular N_6. For simplicity, we show only the contents of the records of the data points of node N_2. From Figure 4.28 (b), we can see that data point p_5, p_6, p_7, p_8, p_{12}, and p_{14} are Voronoi neigbors of p_4 and its Voronoi cell has vertices a, b, c, d, e, and f.

Since the expected time complexity for constructing a Voronoi diagram using Fortune's algorithm is $O(n\log n)$, we can expect to construct the *VoR*-tree with a time-complexity very close to $O(n\log n)$. Since the locations of spatial objects, such as restaurants, are static, we can construct *VoR*-tree before processing the groups' skyline query.

VoR-tree provides us an efficient way to search non-dominated objects in spatial sub-space, since we can find the nearest spatial object in *VoR*-tree from a given query point in $O(\log n)$ time. Using *VoR*-tree, we can significantly reduce the search space that dramatically improves the performance while computing skyline objects at spatial sub-space.

4.5.2 Problem Formulation

Assume that there is a database of restaurants as in Table 4.11. The database has two non-spatial attributes: "Rating" and "Price", in addition to the "Location" attribute. We assume that lower value is better in each of the non-spatial attributes. We also assume there are four users u_1, u_2, u_3, and u_4, whose current locations are at (4.5, 5.5), (5, 6.8), (6, 5), and (5, 3.8), respectively, as in Table 4.12.

To select a good restaurant for the four users, at first, we calculate the Euclidean distance of each restaurant from each of the four users (query points) and construct the table as shown in Table 4.13. In the table, the attribute r-u_1 represents Euclidean distances of the restaurants from user u_1. Similarly, r-u_2, r-u_3, and r-u_4 are the Euclidean distances of restaurants from u_2, u_3, and u_4, respectively. *Sum-Distance* attribute in Table 4.13 contains the sum of Euclidean distances of each data point (restaurant) from the users u_1, u_2, u_3, and u_4.

Note that a restaurant that is the closest from one user can be an attractive candidate. In addition, a restaurant whose sum of Euclidean distances from the four users is

Table 4.11: Restaurant database

ID	Location	Rating	Price
$\mathbf{r_1}$	(3, 9)	3	2
$\mathbf{r_2}$	(7, 5)	2	2
$\mathbf{r_3}$	(7, 7)	3	4
$\mathbf{r_4}$	(5, 1)	3	2
$\mathbf{r_5}$	(4, 4)	2	3
$\mathbf{r_6}$	(4, 8)	3	3
$\mathbf{r_7}$	(5, 6)	3	1
$\mathbf{r_8}$	(1, 3)	3	2
$\mathbf{r_9}$	(5, 3)	2	2
$\mathbf{r_{10}}$	(9, 3)	1	1

Table 4.12: Users' location database

ID	Location
$\mathbf{u_1}$	(4.5, 5.5)
$\mathbf{u_2}$	(5, 6.8)
$\mathbf{u_3}$	(6, 5)
$\mathbf{u_4}$	(5, 3.8)

smallest must be an attractive candidate. Therefore, we use those five spatial attributes for the four users problem.

Next, we join the non-spatial attributes of Table 4.11 and spatial information of Table 4.13 and obtain the information of Table 4.14. After computing Table 4.14, we can get the skyline for the four users by using conventional skyline query, which are r_2, r_5, r_7, r_9, and r_{10}. However, we have to compute spatial features like Table 4.13 for each of different query, which are time-consuming and not affordable.

Considering this fact, we consider an efficient method for computing such a spatial skyline query without constructing all the information of Table 4.14 for a group of users of different locations. Instead, we only compute necessary spatial information for each of different query (group) efficiently. For simplicity, we consider the above examples as running examples in the following subsections.

Table 4.13: Spatial attributes of restaurants

ID	r-u_1	r-u_2	r-u_3	r-u_4	*Sum-Distance*
$\mathbf{r_1}$	3.81	2.97	5.12	5.57	17.47
$\mathbf{r_2}$	2.55	2.69	1	2.33	8.57
$\mathbf{r_3}$	2.92	2.01	2.24	3.77	10.94
$\mathbf{r_4}$	4.53	5.8	4.12	2.8	17.25
$\mathbf{r_5}$	1.58	2.97	2.24	1.02	7.81
$\mathbf{r_6}$	2.55	1.56	3.61	4.32	12.04
$\mathbf{r_7}$	0.71	0.89	1.41	1.48	4.49
$\mathbf{r_8}$	4.30	5.52	5.39	4.08	19.29
$\mathbf{r_9}$	2.54	3.8	2.24	0.8	9.38
$\mathbf{r_{10}}$	5.15	5.18	3.61	4.08	18.38

4.5.3 Query Processing

It is possible to calculate skyline query after constructing a table like Table 4.14 by conventional skyline queries. However, the number of data points such as restaurants is too large that the construction of a table like Table 4.14 and computation of skyline result from such a table using any conventional skyline query algorithm are not affordable.

Considering this fact, we compute the skyline results in two phases.

In the first phase, we compute skyline results in the spatial sub-space like ($r-u_1$, $r-u_2$, $r-u_3$, $r-u_4$, *Sum-Distance*) of Table 4.14. We utilize the concept of *Sum-Distance* for spatial processing which can easily eliminate a large number of objects during the computation of skyline objects in the spatial sub-space.

Based on the skyline result of the spatial sub-space, the second phase efficiently computes whether some other objects can be in the skyline in the non-spatial sub-space like (*Rating*, *Price*) of Table 4.14. In this phase, we check the dominance of non-skyline objects of spatial sub-space against the skyline objects of spatial sub-space. Such an approach can easily eliminate many objects from domination check.

4.5.3.1 Spatial Processing

We say that an object is "spatially dominated" if the object is dominated in the spatial sub-space. For example, we can say that a restaurant in Table 4.13 is "spatially

Table 4.14: Non-spatial and spatial attributes of restaurants

ID	Rating	Price	r-u_1	r-u_2	r-u_3	r-u_4	*Sum-Distance*	
r$_1$	3	2	3.81	2.97	5.12	5.57	17.47	dominated by r_2, r_7
r$_2$	2	2	2.55	2.69	1	2.33	8.57	not dominated
r$_3$	3	4	2.92	2.01	2.24	3.77	10.4	dominated by r_7
r$_4$	3	2	4.53	5.8	4.12	2.8	17.25	dominated by r_7
r$_5$	2	3	1.58	2.97	2.24	1.02	7.81	not dominated
r$_6$	3	3	2.55	1.56	3.61	4.32	12.04	dominated by r_7
r$_7$	3	1	0.71	0.89	1.41	1.48	4.49	not dominated
r$_8$	3	2	4.30	5.52	5.39	4.08	19.29	dominated by r_2, r_7, r_9
r$_9$	2	2	2.54	3.8	2.24	0.8	9.38	not dominated
r$_{10}$	1	1	5.15	5.18	3.61	4.08	18.38	not dominated

dominated", if the restaurant is dominated in its sub-space $\{r$-u_1, r-u_2, r-u_3, r-u_4, *Sum-Distance*$\}$.

For selecting non-dominated objects in spatial sub-space, at first, we select the Voronoi point (restaurant) that is nearest to the centroid of the query points (user locations). For example, if we consider the users (query points) of Table 4.12, we can find that the centroid of r-u_1, r-u_2, r-u_3, and r-u_4 is (5.13, 5. 28). From Table 4.11, we can find that r_j is nearest to (5.13, 5. 28). So, we select r_7. Next, for each of the user, we draw a circle. The radius of each circle is the Euclidean distance from the user and r_j. Let $C(u_i, r_j)$ be a circle whose center is the position of user u_i. The radius of $C(u_i, r_j)$ is the Euclidean distance from u_i to data point r_j. We denote this distance by $D(u_i, r_j)$. We call the region within the union of the circles of r_j as the "search region" of r_j.

We, then, search for the data points within the "search region". To obtain the data points within the "search region", we just consider the Voronoi cells those are either completely inside the "search region" or those have some intersections with any of the circles. If a Voronoi cell is completely inside the search region, we can say that corresponding data point is within the "search region". If a Voronoi cell intersects with any of the circles, we need to check the distance of the corresponding data point from the center of the circles. If we find that the Euclidean distance is less than or equal to the radius of any of the circles, we can decide that the data point is inside the "search region". Otherwise, it is outside the "search region".

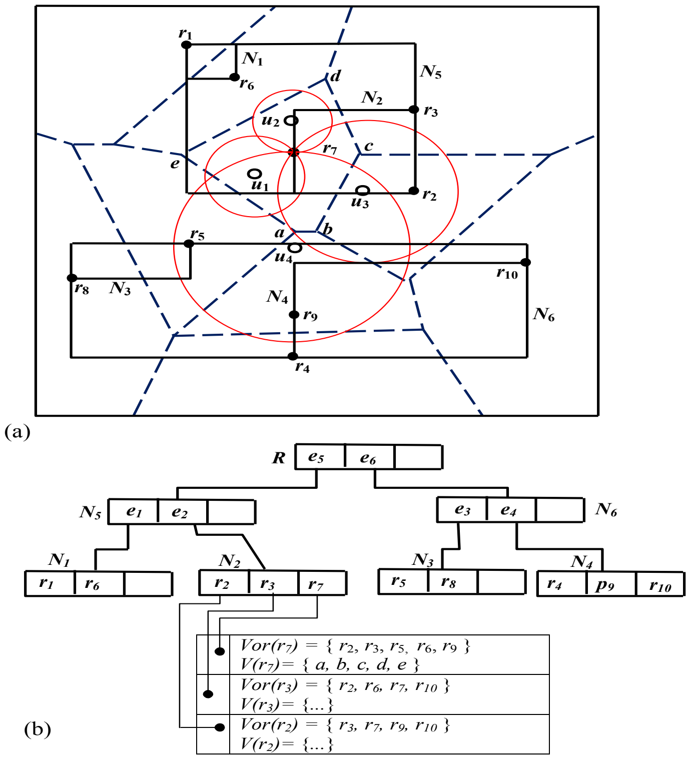


Figure 4.29: (a) Location of users and restaurants (b) *VoR*-tree

Later, we compute the sum of Euclidean distances of a data point (restaurant) from the query points (users). We call this distance "Sum Distance".

We can efficiently compute the set of objects those are not spatially dominated using "search region", "Sum Distance" and *VoR*-tree that incrementally returns the skyline points as explain below.

First, we compute the sum of Euclidean distances for each data point within the "search region". Then, we pick the data point, say r_k that has minimum "Sum Distance" and add r_k along with its "Sum Distance" to a heap. Next, we examine the Voronoi neighbours of r_k, $VN(r_k)$ and add the Voronoi neighbors within the search region in the heap in increasing order of their "Sum Distance". When a data point r_k is explored, we pop it from the heap and add it to the skyline list if it is not dominated in spatial sub-space by some other objects already in the skyline. We continue the process until the heap becomes empty.

Now, consider the computation process of skyline objects in spatial sub-space from the example as shown in Figure 4.29. In the Figure 4.29(a), white dots are locations of four users and black dots are locations of restaurants. We first pick up r_7 and compute

Table 4.15: Spatial information of the data points within search region

ID	r-u_1	r-u_2	r-u_3	r-u_4	*Sum-Distance*
$\mathbf{r_2}$	2.55	2.69	1	2.33	8.57
$\mathbf{r_5}$	1.58	2.97	2.24	1.02	7.81
$\mathbf{r_7}$	0.71	0.89	1.41	1.48	4.49
$\mathbf{r_9}$	2.54	3.8	2.24	0.8	9.38

Table 4.16: Heap for traversing *VoR*-tree

Step	**Heap content**	**Skyline** S
1	(r_7, 4.49)	$\oslash$
2	(r_7, 4.49), (r_5, 7.81), (r_2, 8.57), (r_9, 9.38)	$\oslash$
3	(r_5, 7.81), (r_2, 8.57), (r_9, 9.38)	$\{r_7\}$
4	(r_2, 8.57), (r_9, 9.38)	$\{r_7, r_5\}$
5	(r_9, 9.38)	$\{r_7, r_5, r_2\}$
6	$\oslash$	$\{r_7, r_5, r_2, r_9\}$

$C(u_i, r_7)$ for each user u_i ($i = 1, ..., 4$) to get the “search region”. We, then, find that restaurants r_2, r_5, r_7, and r_9 are within the “search region” of r_7. Next, we compute the “Sum Distance” for each of these restaurants and construct the table as shown in Table 4.15. In the process, we keep the heap data structure like Table 4.16.

Looking at the information of Table 4.15, we can find that returant r_7 has minimum “Sum Distance”. So, we add $(r_7, dist(r_7, U))$ to the heap and marks r_7 as “checked”. Next, we collect the Voronoi neighbors of r_7 and find that its Voronoi neighbors r_2, r_5, and r_9 are inside the “search region” (union of $C(u_i, r_7)$ for user u_i ($i = 1, ..., 4$)). Then,

Table 4.17: Non-spatial information of dominated objects in spatial sub-space

ID	**Rating**	**Price**
$\mathbf{r_1}$	3	2
$\mathbf{r_3}$	3	4
$\mathbf{r_4}$	3	2
$\mathbf{r_6}$	3	3
$\mathbf{r_8}$	3	2
$\mathbf{r_{10}}$	1	1

Table 4.18: Non-spatial information of the skyline objects in spatial sub-space

ID	Rating	Price
$\mathbf{r_2}$	2	2
$\mathbf{r_5}$	2	3
$\mathbf{r_7}$	3	1
$\mathbf{r_9}$	2	2

we add $(r_2, dist(r_2, U))$, $(r_5, dist(r_5, U))$ and $(r_9, dist(r_9, U))$, to the heap in ascending order of their "Sum Distance".

After the steps, restaurant r_7 is added to the skyline list S as shown in step-3 of Table 4.16. Next, we pick the top element r_5 from the heap and find that its Voronoi neighbours are r_1, r_6, r_7, r_8 and r_9. Among them r_1, r_6 and r_8 are outside the search region and r_7 and r_9 are already checked. Therefore, no new entry is added in the heap by r_5. After that, we examine the spatial dominance of r_5 against r_7. Since r_5 is not spatially dominated by r_7, we add r_5 in S as in step-4. Similarly, we continue the process and add r_2 and r_9 to the skyline. After the process of r_9, the heap becomes empty. Finally, we get $S = \{r_2, r_5, r_7, r_9\}$ as skyline result based on spatial sub-space.

Since the "search region" is relatively very small compared with the whole space, such computation is very much efficient with respect to space and time.

4.5.3.2 Non-spatial Processing

In non-spatial processing, at first, we collect all dominated data points at spatial sub-space. Table 4.17 shows such data points with non-spatial information. From Table 4.17, we can see that data points r_1, r_3, r_4, r_6, r_8, and r_{10} are spatially dominated. So, we need to check their dominance in the non-spatial sub-space.

To obtain non-dominated objects at non-spatial sub-space, we check their dominance against the skyline objects r_2, r_5, r_7, and r_9 of spatial sub-space. Table 4.18 shows non-spatial information of these skyline objects in spatial sub-space. Note that objects of Table 4.18 are in the final skyline as well.

If we check the objects of Table 4.17 against the objects of Table 4.18, we can find that r_7 also dominates r_1, r_3, r_4, r_6, r_8 in non-spatial sub-space. So, they are not in the skyline. However, object r_{10} is not dominated in its non-spatial sub-subspace by any object of Table 4.18 and there is no other non-dominated object in Table 4.17. So, r_{10} is also in the skyline. Finally, we find r_2, r_5, r_7, r_9 and r_{10} as final skyline result.

Algorithm 4.2 shows the proposed computation procedure of the spatial skyline queries. It first computes "spatially dominated" objects based on spatial sub-space (line 3-20). Then, **Algorithm 4.2** computes whether there are skyline objects among the "spatially dominated" objects by examining non-spatial sub-space (line 21-29). Finally, the algorithm returns the spatial skyline objects (line 30).

4.5.4 Correctness of Algorithm

The correctness of **Algorithm 4.2** follows some basic properties of geometry and skyline query. From **Algorithm 4.2**, we can see that for a set of query points Q, it first adds the data point r_j with minimum "Sum Distance" to the skyline S. All the Voronoi neighbors of r_j are then checked and added to the heap in increasing order of their their "Sum Distance" if they are within the "search region".

The traversal started from the data point with minimum "Sum Distance" towards the Voronoi neighbors in increasing order of "Sum Distance" and we can find that the data point r_j with minimum "Sum Distance" is in the skyline S. The reason is that "Sum Distance" is considered as an attribute in the spatial sub-space. During the consideration of Voronoi neighbors of a data point, we just consider the Voronoi neighbors within the "search region". We can easily ignore the Voronoi neighbors of a data point those are outside the "search region". This is because, the Euclidean distances between a Voronoi neighbor that is outside the "search region" and query points must be larger than the Eucledian distances between r_j and query points. Hence, any Voronoi neighbor that is outside the "search region" will never be in the skyline in the spatial sub-space. However, the Voronoi neighbors those are within the "search region" can be in the skyline of spatial sub-space. So, **Algorithm 4.2** further checks such Voronoi neighbors against the data points in S to determine whether they are in the skyline of the spatial sub-space or not.

Line 21-29 of **Algorithm 4.2** shows the computation of skyline objects in non-spatial sub-space. The correctness of **Algorithm 4.2** for computing skyline objects in non-spatial sub-space comes from the basic idea of skyline. If an object is in the skyline of $d - i$ (i = 1 to d -1) dimensions, it will also be in the skyline of d dimensions.

Algorithm 4.2 Computation

Input: Set of query points $U = \{u_1, u_2, \cdots, u_i\}$ and data points $R = \{r_1, r_2, \cdots, r_j\}$

Output: Spatial skyline objects Set S, $S \subseteq R$

```
begin
set $D, (D \subseteq R)$ = the set of dominated objects in spatial sub-space
select a data point $r_j$ that is closest to the centroid of the query points $U = \{u_1, u_2, \cdots, u_i\}$
compute the search region of $r_j$
obtain the data points set, say $T$ within the “search region“, $T \subseteq R$
compute the “Sum Distance” $dist_k$ of each data point $r_k$, $r_k \in T$
select the data point $r_k$ that has minimum “Sum Distance”
add ($r_k$, $dist_k$ ) to the heap $H$
select the Voronoi neighbors of $r_k$ those are within the “search region” and add them to $H$ in increasing order of their “Sum Distance”
remove ($r_k$, $dist_k$ ) from $H$ and add $r_k$ to $S$
repeat
    choose the top element, say $r_l$ from $H$
    select the Voronoi neighbors of $r_l$ those are within the “search region” and add them to $H$ in increasing order of their “Sum Distance”
    pop $(r_l, dist_l)$ from $H$
    if $r_l$ is not dominated by some other objects in $S$ in spatial sub-space then
        add $r_l$ to $S$
    else
        add $r_l$ to $D$
    end if
until $H$ becomes empty
for each data point $r_m \in D$ do
    if $r_m$ is dominated by some other objects of $S$ in non-spatial sub-space then
        $r_m \notin S$
    else if $r_m$ is dominated by some other objects of $D$ in non-spatial sub-space then
        $r_m \notin S$
    else
        add $r_m$ to $S$
    end if
end for
return $S$ as the spatial skyline result
end
```

Table 4.19: Datasets for experiments

Datasets	Total Objects	Density
r	50,747	–
s_1	80,000	0.08
s_2	50,000	0.05
s_3	20,000	0.02

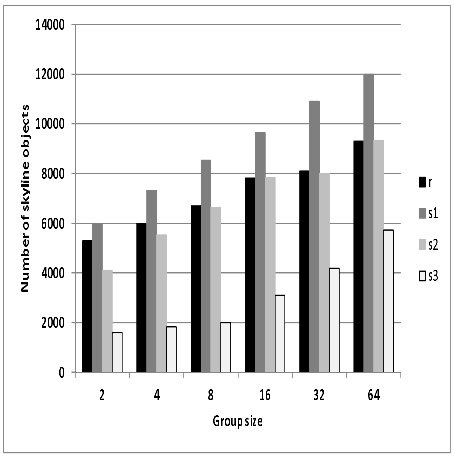


Figure 4.30: No. of skyline objects

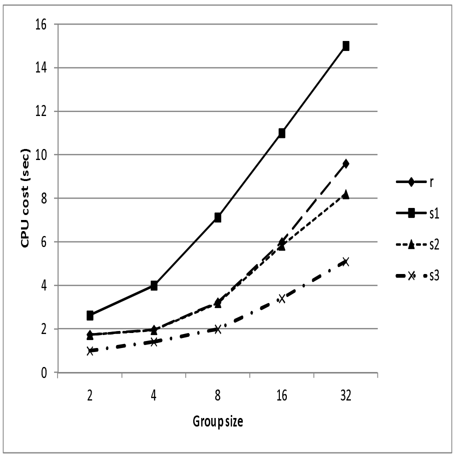


Figure 4.31: time vs group size

4.5.5 Performance Evaluation

To evaluate the efficiency and effectiveness of the proposed skyline queries algorithm, we conducted extensive experiments. We implemented all algorithms using Microsoft Visual C++ V6.0, and conducted the experiments on a PC with Intel core i5 processor, 2.3 GHz CPU, 4G main memory and 200G hard disk, running Microsoft Windows 7 Professional Edition.

4.5.5.1 Experimental Setup

We implemented the experiments by deploying both real and synthetic datasets. The real datasets came from line segment data of Long Beach from the TIGER database [79]. We made this point set by extracting the midpoint for each road line segment. The set consists of 50,747 points normalized in [0,1000] × [0, 1000] space. There are three synthetic datasets $s1$, $s2$, and $s3$ with different densities normalized in [0,1000] × [0,1000] space as in Table 4.19. In Table 4.19, r stands for real dataset of TIGER

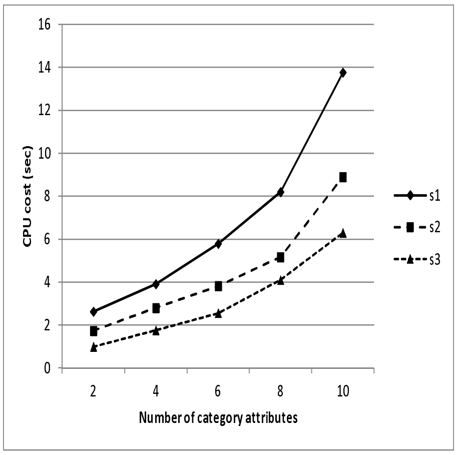


Figure 4.32: time vs category attributes

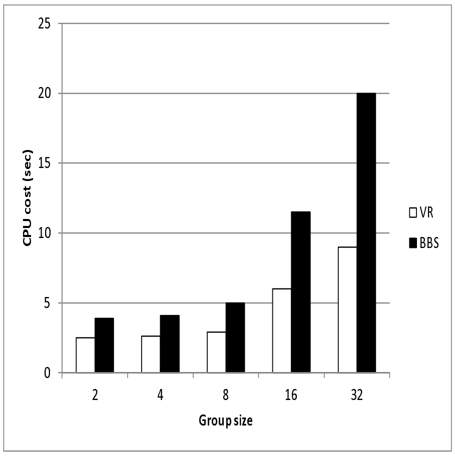


Figure 4.33: Comparative computation time

database and density means how many points fall into one square unit in average. The points in each synthetic dataset are distributed randomly. We indexed all datasets by using a *VoR*-tree. By default, we consider a location attribute and two category attributes for each data set.

4.5.5.2 Experimental Results

The first experiment studies the numbers of skyline objects under different densities and different group size. Figure 4.30 shows the total numbers of skyline objects from datasets r, s_1, s_2, and s_3. From Figure 4.30, we can see that total number of skyline objects increases with the increase in density and group size.

The second experiment explores the performance of the algorithm under different group size and different densities. From Figure 4.31, we can observe that the running time increases with the increase in group size. Also, it is observed that running time increases if the density of data points increases.

Next experiment shows the effect of the increase in the number of category attributes while keeping the group size to 32. In this experiment, we considered three synthetic datasets. Figure 4.32 shows result. From the result, we can see that there is an increase in computation time with the increase in the number of category attributes.

In the fourth experiment, we compared our algorithm with BBS approach using the dataset r. Although there are some other spatial skyline query algorithms, we considered BBS algorithm for comparison due to its effectiveness in handling both

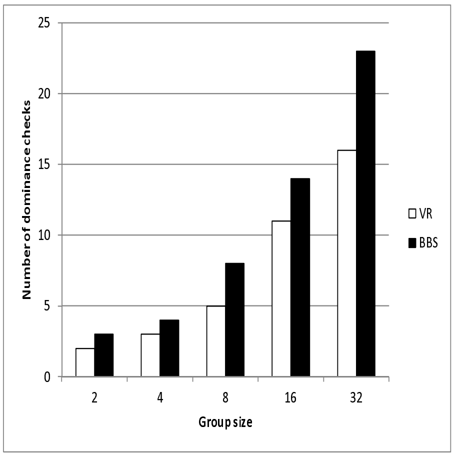


Figure 4.34: Comparative dominance check

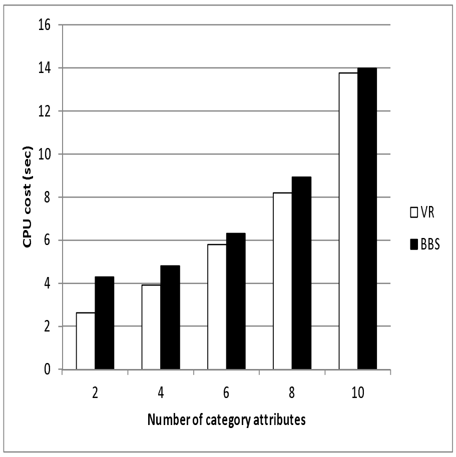


Figure 4.35: Comparative performance on category attributes

spatial and non-spatial attributes. From the result of Figure 4.33, we can see that our algorithm (VR) significantly outperforms BBS algorithm.

Next experimental results are shown in Figure 4.34. It shows the relative dominance check between our algorithm and BBS algorithm. From Figure 4.34, we can see that our algorithm constantly performs less number of dominance check compared with BBS algorithm.

Figure 4.35 shows the results of our sixth experiment. It shows the effectiveness of our algorithm while there is an increase in the number of category attributes. In this experiment, we considered the synthetic dataset s_1 and group size 2. From the result of Figure 4.35, we can see that in case of fixed number of users and more category attributes, the performance of our algorithm is still better than BBS algorithm.

The final experiment shows the effectiveness of our algorithm in case of large number of category attributes while there is an increase in group size. In this experiment, we considered ten category attributes. From the result of Figure 4.36, we can find that our algorithm becomes comparatively better than BBS algorithm with an increase in group size.

4.6 Conclusion

In this chapter, we have three different approaches for selecting spatial objects. First two approaches of skyline queries based on surrounding environments. Third approach

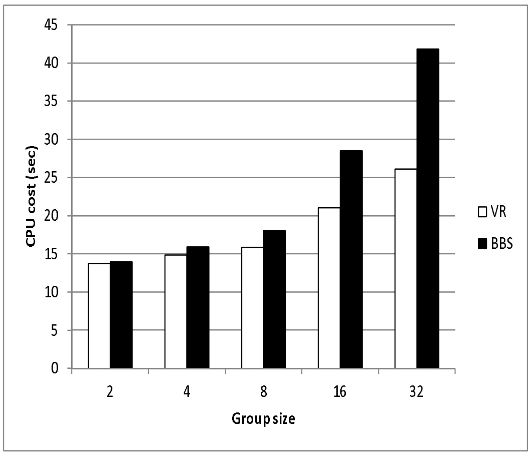


Figure 4.36: Comparative running vs group size for large number of category attributes

considered spatial objects selection for a group of users. Our first approach considered the best values in each attribute of each surrounding facility. This approach is applicable for hotel recommendation. In this approach, we used a grid-based approach for materialization so that we can compute the result quickly and efficiently. Second approach considered objects count of the facilities in the surrounding environments. This approach is suitable for real estate recommendations. For efficient computation, we have utilized *aR*-tree indexing mechanism. Our third approach combined spatial and non-spatial features while selecting spatial objects as both of these features are important to describe a spatial object. In this framework, we utilized *VoR*-tree and "Sum Distance" to calculate spatial skyline objects for a group of users of different locations efficiently. Experimental results demonstrate that effectiveness of all three approaches

In our first two approaches, we did not consider any weighting mechanism for the surrounding facilities. However, we have noticed that we should weight each surrounding facility based on distance, quality, and user's preferences in order to improve the result. To use a proper weighting is one of an important open problem of the works presented in this chapter.

In our third method, we have considered static query points, which mean all query points do not move. However, in general, query points are not static. Therefore, we have to develop an efficient algorithm that can handle the change in the locations of query points in our future works.

Chapter 5

Conclusions

The main focus of this dissertation is on information filtering of very large databases by using skyline queries. In particular, this dissertation covered three different types of information filtering mechanisms using skyline queries. These are skyline sets queries in distributed databases, skyline queries by utilizing surrounding environments, and spatial skyline queries for groups of users. For efficient computation of skylines in each of the above situations, we proposed novel methods, efficient algorithms, and processing techniques. We showed the efficiency of our approaches through extensive experiments.

The remaining of this chapter is organized as follows. In Section 5.1, we provide the application areas of our developed algorithms. Then, in Section 5.2, we present the contributions of this dissertation. Finally, we present some future directions derived from this work in Section 5.3.

5.1 Application of the Proposed Skyline Queries

With the increase of data volume in different applications, methods for ranking the usefulness of query results are highly desirable, in particular in case of large amount data that often generates long result against any query. Such an approach could bridge the gap between the two alternative paradigms of skyline queries and rank-aware query processing.

In this dissertation, at first, we proposed skyline sets queries from distributed databases in Chapter 3. This type of information filtering mechanism deals with data that are combination of raw data and is applicable to different application areas. For example, in the field of stock investment, it is very likely that an investor will not invest all his money in just one stock, but many stocks distributed at different geographic locations, which allows the investor to obtain a higher return and/or a lower risk. In such a situation, our proposed skyline sets queries can efficiently help the investors. Moreover, consider a global web information application distributed among several servers around the world that help different companies to perform their online business. Each server locally stores the data of the company at its location. In general, the data belongs to a company is very useful for the company and leaks of valuable data might be a big problem for the company. However, the company wants to continue its business while preserving the privacy of data. In such a situation, our proposed skyline set queries can be an effective solution for the company as our developed skyline sets query mechanism does not disclose individual record's values. In addition, our skyline sets queries are useful for data with outliers and in frequently update situations. As for example, in environment sensitive situations where data are collected using many sensors, its is important to take decisions by combining the data of several sensors to minimize the effect of outliers in sensor data. We can apply our proposed skyline sets queries in such environment sensitive situations. Also, in case of applications like hotel recommendation systems where data are frequently updated, we can apply the proposed system as it will provide the user some alternatives.

Next part of this dissertation considers spatial objects selection using skyline queries considering surrounding environments. Using this type of skyline queries, we can recommend good hotels and/or houses based on not only their features but also their surrounding environments. Such type of skyline queries are mainly applicable in two different application areas: hotel recommendations and real estate recommendations. In case of booking a hotel, a user may want to stay in a hotel that has bars and restaurants in its surrounding areas, although the features of the hotel is not good. In such a situation, our first approach presented in Chapter 4 is helpful. In our life, selecting a good house is very important for us. The location is a very important factor for us during the selection of a house. For example, a house is convenient if there are many supermarkets within a walking distance. Our second approach presented in Chapter 4 can recommend users such houses to the users so that the users can make decision during buying their houses.

The last part of this dissertation also presented at Chapter 4 that considered the group needs. Consider that the people of a multidisciplinary task force located at different places are looking for a suitable place such as restaurants to hold a lunch-on meeting. Social network services can connect such users and make such groups. Selection of such a restaurant not only depends on its location but also its non-spatial features, such as price and rating. In such a situation, our spatial skyline queries presented in Chapter 4 is highly applicable. Is is also applicable to many other applications such as disaster management and attack planning during war. As for example, during a disaster, there are many locations under critical situations. We can apply our approach to find the locations of such critical places those are easily reachable by the people of the rescue team located at different places. We can also use our method to plan an attack based on the solders locations and importance of the opposition's camp.

5.2 Contributions

Due to the rapid increase in data volume in present days, efficient filtering of information is very important. Skyline queri mechanism is an efficient tool in this regard. In this dissertation, we studied information filtering by using skyline queries in three different domains: (i) privacy preserving information retrieval from distributed databases, (ii) skyline queries for selecting spatial objects by utilizing surrounding environments and (iii) spatial skyline queries for spatial object selection for a group of users. The main contributions of them are stated below.

5.2.1 Privacy Preserving Information Retrieval

In Chapter 3, we provided an agent-based parallel computation framework for secure retrieval of information from distributed databases by using skyline sets queries. The approach is well applicable for large-scale distributed databases. Moreover, this approach can preserve privacy under statistical compromisable situations. As a result the proposed mechanism can retrieve information without the discloser of individual's information. We performed several experiments to show the effectiveness of our algorithms. Experimental evaluation demonstrates that the proposed framework is meaningful and scalable enough to handle large and high dimensional datasets.

5.2.2 Spatial Objects Selection by Utilizing Surrounding Environments

Conventional skyline queries select objects based on non-spatial attributes such as price and rating. However, surrounding environments of objects can play a vital role while selecting an object. Considering this fact, in Chapter 4, we proposed two methods to compute skyline objects considering surrounding facilities. Our first approach in Chapter 4 considered the best value in each attribute of each surrounding facility while our second method considered the objects count of each type of facility in the surrounding environment. Besides theoretical guarantees, our comprehensive performance study indicate that the techniques are very effective and efficient.

5.2.3 Spatial Objects Selection for a Group of Users

Recently, GPS devices and location based services become popular and we have databases containing spatial information. Considering this fact, in Chapter 4, we also presented an efficient method for selecting spatial objects for a group of users considering both spatial and non-spatial features of the objects in Chapter 4. Due to the current availability of social network services, nowadays we can connect users and make such groups. If a group wants to find a restaurant to hold a meeting, we have to select a convenient place for all users. In such cases, proposed skyline query algorithm selects a set of spatial objects, which are not dominated by other spatial objects, for the group. We performed several extensive experiments to show the effectiveness of our approach.

5.3 Future Research Direction

In the following, we outline some possibilities for extending the work presented in this dissertation.

5.3.1 On Sets Skyline Query

The work on skyline sets queries in this dissertation assumes that all the attributes of the databases are numerical having static dataset. Development of efficient skyline sets query mechanism for frequently update data can be a promising future research. Moreover, development of algorithms for skyline sets queries from databases with categorical attributes can be another future research. Another interesting future research is the efficient computation of sub-space skyline sets and *k*-dominant skyline sets.

5.3.2 On Spatial Objects Selection by Utilizing Surrounding Environments

Our skyline queries for selecting spatial objects by utilizing surrounding facilities considered uniform weights of all facilities in the surrounding areas. However, we have noticed that we should weight each surrounding facility based on distance, quality, and user's preferences in order to improve the result. To use a proper weighting is one of an important open problem of the works presented in this dissertation.

5.3.3 On Selecting Spatial Objects for a Group of Users

In this dissertation, we used numerical attributes to describe the features of spatial objects. However, features of the objects can be nominal and skyline computation of spatial objects based on users locations and nominal features of the objects can be one future research. In this dissertation, we just considered static query points, which mean all query points do not move. However, in general, query points are not static. Therefore, we want to develop an efficient algorithm that can handle the change in the locations of query points in our future works.

Bibliography

[1] S. Borzonyi, D. Kossmann, and K. Stocker, "The skyline operator", *In Proc. of ICDE*, 2001, pp. 421-430. 1, 2, 9, 11, 12, 14, 16, 48, 59, 62, 68, 90

[2] J. Pei, W. Jin, M. Ester, and Y. Tao, "Catching the best views of skyline: a semantic approach based on decisive subspaces", *In Proc. of VLDB Conference*, 2005, pp. 253-264. 2, 10, 48

[3] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang, "Efficient computation of the skyline cube", *In Proc. of VLDB Conference*, 2005, pp. 241-252. 2, 10, 48

[4] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An Optimal and progressive algorithm for skyline queries", *In Proc. of ACM SIGMOD Conference*, 2003, pp. 467-478. 2, 10, 22, 40, 54, 62, 68

[5] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting", *In Proc. of ICDE*, 2003, pp. 717-719. 2, 16, 62, 68, 100, 111

[6] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets", *In Proc. of VLDB Conference*, 2005, pp. 229-240. 2, 17

[7] K. L. Tan, P. K. Eng, and B. C. Ooi, "Efficient progressive skyline computation", *In Proc. of VLDB Conference*, 2001, pp. 301-310. 2, 18, 20, 33, 62, 68

[8] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries", Ȋn Proc. of VLDB Conference, 2002, pp. 275-286. 2, 21, 62, 68

[9] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems", *In ACM Transactions on Database Systems*, 2005, vol. 30(1), pp. 41-82. 2, 22, 40, 54

[10] P. Wu, C. Zhang, Y. Feng, B. Zhao, D. Agrawal, and A. Abbadi, “Parallelizing skyline queries for scalable distribution”, *In Proc. of International Conference on Extending Database Technology (EDBT)*, 2006, pp. 112-130. 2, 24, 25, 68

[11] S. Wang, B. Ooi, A. Tung, L. Xu, “Efficient skyline query processing on peer-to-peer networks”, *In Proc. of International Conference on Data Engineering (ICDE)*, 2007, pp. 1126-1135. 2, 26, 27, 62, 69, 70

[12] S. Wang, Q. H. Vu, B. C. Ooi, A. K. Tung, and L. Xu, “Skyframe: a framework for skyline query processing in peer-to-peer systems”, *VLDB Journal*, 2009, vol. 18(1),pp. 345-362. 2, 26, 27

[13] L. Chen, B. Cui, H. Lu, L. Xu, and Q. Xu, “iSky: efficient and progressive skyline computing in a structured P2P network”, *In Proc. of the International Conference on Distributed Computing Systems (ICDCS)*, 2008, pp. 160-167. 2, 27, 28

[14] H. Li, Q. Tan, and W. Lee, “Efficient progressive processing of skyline queries in peer-to-peer systems”, *In Proc. of the International Conference on Scalable Information Systems (Infoscale)*, 2006, p. 26. 2, 29, 62, 69

[15] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi, “Skyline queries against mobile lightweight devices in manets” *In Proc. of International Conference on Data Engineering (ICDE)*, 2006, p. 66. 2, 27, 28, 30, 35, 39, 62, 69

[16] K. Hose, C. Lemke, and K. Sattler, “Processing relaxed skylines in PDMS using distributed data summaries”, *In Proc. of International Conference on Information and Knowledge Management (CIKM)*, 2006, pp. 425-434. 2, 30, 62, 69

[17] K. Hose, C. Lemke, K. Sattler, and D. Zinn, “A relaxed but not necessarily constrained way from the top to the sky”, *In Proc. of International Conference on Cooperative Information Systems (CoopIS)*, 2007, pp. 339-407. 2, 30

[18] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis, “SKYPEER: efficient subspace skyline computation over distributed data”, *In Proc. of International Conference on Data Engineering (ICDE)*, 2007, pp. 416-425. 2, 32, 34, 62, 69

[19] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis, “Efficient routing of subspace skyline queries over highly distributed data”, *IEEE Trans. Knowl. Data Eng. (TKDE)*, 2010, vol. 22, no. 12, pp. 1694-1708. 2, 32, 33

[20] K. Fotiadou, and E. Pitoura, “BITPEER: continuous subspace skyline computation with distributed bitmap indexes”, *In Proc. of International Workshop on Data Management in Peer-to-Peer Systems (DaMaP)*, 2008, pp. 35-42. 2, 33, 62, 70

[21] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou, “Parallel distributed processing of constrained skyline queries by filtering”, *In Proc. of International Conference on Data Engineering (ICDE)*, 2008, pp. 546-555. 2, 34, 35, 37

[22] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Norvag, “AGiDS: a grid-based strategy for distributed skyline query processing”, *In Proc. of International Conference on Data Management in Grid and Peer-to-Peer Systems (Globe)*, 2009, pp. 12-23. 2, 35, 36

[23] L. Zhu, Y. Tao, and S. Zhou, “ Distributed skyline retrieval with low bandwidth consumption”, *IEEE Trans. Knowl. Data Eng. (TKDE)*, 2009, vol. 21, no. 3, pp. 384-400. 2, 36, 37

[24] J. B. Rocha-Junior, A. Vlachou,C. Doulkeridis, and K. Norvag, “Efficient execution plans for distributed skyline query processing”, *In Proc. of International Conference on Extending Database Technology (EDBT)*, 2011, pp. 271-282. 2, 37, 38, 62, 70

[25] K. Hose, and A. Vlachou, “A survey of skyline processing in highly distributed environments”, *The VLDB Journal*, 2012, vol. 21, pp. 359-384. 11, 24

[26] M. Sharifzadeh, and C.Shahabi, “The spatial skyline queries”, *In Proc. of the 32nd International Conference on Very Large Data Bases*, 2006, pp. 751-762. 3, 4, 10, 39, 40, 41, 43, 94, 95

[27] W. Son, M. Lee, H. Ahn, and S. Hwang, “Spatial skyline queries: an efficient geometric algorithm”, *In Proc. of 11th International Symposium on Spatial and Temporal Databases*, 2009, pp 247-264. 4, 44, 94, 95

[28] X. Guo, Y. Ishikawa, and Y. Gao, “Direction-based spatial skylines”, *In Proc. of the ACM SIGMOD International Conference on Management of Data*, 2010, pp. 73-80. 4, 45, 46, 47, 94, 95

[29] S. H. Yoon, and C. Shahabi C., “Distributed spatial skyline query processing in wireless sensor networks”, *In IPSN*, 2009. 4, 47, 94

[30] W. Son, S. W. Hwang, and H. K. Ahn, “MSSQ: Manhattan Spatial Skyline Queries”, *In Lecture Notes in Computer Science*, vol. 6849, 2011, pp 313-329. 4, 48

[31] G. W. You, M. W. Lee, H. Im, and S. W. Hwang, “The FarthestSpatialSkyline-Queries”, *In Information Systems*, 2013, vol. 38, pp. 286-301. 4, 48

[32] C. Y. Chan, H. V. Jagadish, K. L. Tan, A. K. H. Tung, and Z. Zhang, “On high dimensional skylines”, *In Proc. of EDBT*, 2006, pp. 478-495. 49

[33] 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 Proc. of ACM SIGMOD Conference*, 2006, pp. 503-514. 49, 50

[34] M. A. Siddique, and Y. Morimoto, “Efficient maintenance of all *k*-dominant skyline query results for frequently updated database”, *IARIA International Journal on Advances in Software*, 2010, vol. 3, no. 3, pp. 424-433. 50

[35] C. Y. Chan, P. K. Eng, and K. L. Tan, “Stratified computation of skylines with partially ordered domains”, *In ACM SIGMOD Conference*, 2005, pp. 203-214. 51

[36] S. Zhang, N. Mamoulis, D. W. Cheung, and B. Kao, “Efficient skyline evaluation over partially ordered domains”, *In Proc. of VLDB Conference*, 2010, pp. 1255-1266. 52

[37] D. Sacharidis, S. Papadopoulos, and D. Papadias, “Topologically sorted skylines for partially ordered domains”, *In Proc. of ICDE* , 2009, pp. 1072-1083. 52

[38] J. Lee, G. W. You, S. W. Hwang, J. Selke, and W. T. Balke, “Optimal Preference Elicitation for Skyline Queries over Categorical Domains”, *In LNCS*, 2008, vol 5181, pp. 610-624. 52

[39] K. Deng, X. Zhou, and H. T. Shen, “Multi-source skyline query processing in road networks”, *In Proc. of 23rd International Conference on Data Engineering*, 2007, pp. 796-805. 52, 94, 96

[40] M. Safar, D. E. Amin, and D. Taniar, “Optimized skyline queries on road networks using nearest neighbors”, *Journal of Personal and Ubiquitous Computing*, 2011, vol. 15 no. 8, pp.845-856. 52, 94, 96

[41] Y. K. Huang, C. H. Chang, and C. Lee, “Continuous distance-based skyline queries in road networks” *Journal of Information Systems*, 2006, vol. 37, pp. 611-633. 52, 94, 96

[42] Z. Qiao, J. Gu, X. Lin, and J.Chen, “Privacy-preserving skyline queries in LBS”, *In Proc. of International Conference on Machine Vision and Human-machine Interface* 2010, pp. 499-504. 2, 52, 70

[43] I. F. Su, Y. C. Chung, and C. Lee, “Top-k combinatorial skyline queries”, *In Proc. of 15th Database Systems for Advanced Applications (DASFAA)*,2010, pp.79-93. 2, 53, 70

[44] M. A. Siddique, and Y. Morimoto, “Algorithm for computing convex skyline objectsets on numerical databases”, *In IEICE Transaction on Information and Systems*, 2010, vol. E93-D, pp. 2709–2716. 2, 53, 61, 65, 67, 71, 72, 73

[45] Y. Morimoto, and M. A. Siddique, “Skyline sets query and its extension to spatio-temporal databases”, *In LNCS*, 2010, vol. 5999, pp. 317-329. 53

[46] E. Dellis, and B. Seeger, “Efficient computation of reverse skyline queries”, *In Proc. of VLDB*, 2007, pp. 291-302. 53, 54

[47] B. Jiang, and Jian Pei, “Online interval skyline queries on time series”, *In Proc. of ICDE*, 2009, pp.1036-1047. 54, 55, 56

[48] J. Pei, B. Jiang, X. Lin, and Y. Yuan, “Probabilistic skylines on uncertain data”, *In Proc. of VLDB Conference*, 2007, pp. 15-26. 57

[49] K. Kodama, Y. Iijima, X. Guo, and Y. Ishikawa, “Skyline queries based on user locations and preferences for making location-based recommendations”, *In Proc. of ACM LBSN*, 2009, pp. 9-16. 58, 94, 96

[50] R. C. Wong, A. W. Fu, J. Pei, Y. S. Ho, T. Wong, and Y. Liu, “Efficient skyline querying with variable user preferences on nominal attributes”, *In Proc. of VLDB*, 2008, pp. 1032-1043. 58

[51] H. Choi, H. Jung, K. Y. Lee, and Y. D. Chung, “Skyline queries on keyword-matched data”, *In Information Sciences*, 2013, vol. 232, pp. 449-463. 58

[52] W. B. Frakes, and R. A. B. Yates, “Information retrieval: data structures & Algorithms”, Prentice-Hall, 1992. 59

[53] B. Ricardo, and R. Berthier, “Modern information retrieval”, Pearson Education Limited, England, 1999. 59

[54] J. Zobel, A. Moffat, K. Ramamohanarao, “Inverted files versus signature files for text indexing”, *ACM Transactions on Database Systems*, 1998, vol. 23, no. 4, pp. 453-490. 59

[55] A. Silberschatz, H. F. Korth, and S. Sudarshan, “Database systems concepts”, Fifth Edition, Tata McGraw Hill, pp. 709 730, 2002.

[56] N. Beckmann, H. P. Kriegel, R. S., and B. Seeger, “The R*-tree: an efficient and robust access method for points and rectangles”, *In Proc. of ACM SIGMOD*, 1990, pp. 322-331. 44

[57] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, “Efficient OLAP operations in spatial data warehouses”, *In Lecture Notes in Computer Science*, 2001, vol. 2121, pp. 443-459. 5, 106, 107

[58] M. Sharifzadeh, and C. Shahabi, “VoR-Tree: R-trees with Voronoi diagrams for efficient processing of spatial nearest neighbor queries”, *In Proc. of VLDB Conference*, 2010, pp. 1231-1242. 5, 116

[59] A. Guttman, “R-trees: a dynamic index structure for spatial searching”, *In Proc. of ACM SIGMOD*, 1984, pp. 47-57. 22, 114

[60] D. H. McLain, “Drawing contours from arbitrary data points”, *In The Computer Journal*, 1974, pp. 318-324. 9

[61] H. T. Kung, F. Luccio, and F. Preparata, “Finding the maxima of a set of vectors”, *In Journal of ACM*, 1975, pp. 469-476. 9

[62] F. P. Preparata, and M. I. Shamos, “Computational geometry: an introduction”, Springer Verlag, 1985. 9

[63] P. Godfrey, R. Shipley, J. Gryz, “Maximal vector computation in large data sets”, *In Proc. of VLDB Conference*, 2005, pp. 229-240. 62, 68

[64] W. T. Balke, U. Guntzer, and J. X. Zheng, “Efficient distributed skylining for web information systems”, *In Proc. of 9th International Conference on Extending Database Technology*, 2004, pp. 256-273. 62, 70

[65] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, “A scalable content-addressable network”, *In Proc. of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2001, pp. 161-172. 25, 27

[66] H. V. Jagadish, B. C. Ooi, and Q. H. Vu, “BATON: a balanced tree structure for peer-to-peer networks”, *In Proc. of VLDB Conference*, 2005, pp. 661-672. 26, 27, 69

[67] Y. Morimoto, T. Fukuda, H. Matsuzawa, K. Yoda, and T. Tokuyama, “Algorithms for mining association rules for binary segmentations of huge categorical databases”, *In Proc. of VLDB Conferences*, 1998, pp. 380-391. 77

[68] S. Park, T. Kim, J. Park, J. Kim, and H. Im, “Parallel skyline computation on multicore architectures”, *In Proce. of the ICDE*, 2009, pp. 760-771. 69

[69] Y. Gao, G. Chen, L. Chen, and C. Chen, “Parallelizing progressive computation for skyline queries in multi-disk environment”, *In Proc. of International Conference of Database and Expert Systems Applications (DEXA)*, 2006, pp. 697-706. 69

[70] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou, “Parallel distributed processing of constrained skyline queries by Filtering”, *In Proc. of ICDE*, 2008, pp. 546-555. 69

[71] A. Vlachou, C. Doulkeridis, and Y. Kotidis, “Angle-based space partitioning for efficient parallel skyline computation”, *In Proc. of ACM SIGMOD Conference*, 2008, pp. 227-238. 69

[72] F. A. Rabhi, and S. Gorlatch, “Patterns and skeletons for parallel and distributed computing”, Springer-Verlag, 2003. 69

[73] N. Roussopoulos, S. Kelley, and F. Vincent, “Nearest neighbor queries”, *In Proc. of ACM SIGMOD*, 1995, pp. 71-79. 95

[74] S. Berchtold, C. Bohm, D. A. Keim, and H. P. Kriegel, “A cost model for nearest neighbor search in high-dimensional data space”, *In Proc. of the 16th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* , 1997, pp. 78-86. 95

[75] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is “nearest neighbor” meaningful?”, *In LNCS, Springer, Heidelberg*, vol. 1540, 1999, pp. 217-235. 95

[76] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, “Aggregate nearest neighbor queries in spatial databases”, *ACM Transactions on Database Systems*, vol. 30 no. 2, 2005, pp. 529-576, 2005. 95

[77] B. Zhang, K. C. K. Lee, and W. C. Lee, “Location-dependent skyline query”, *In Proc. of 9th International Conference on Mobile Data Management*, pp. 3-8, 2008. 94, 96

[78] S. Fortune, “A sweep line algorithm for Voronoi diagrams”, *In Proc. of the Second Annual Symposium on Computational geometry*, pp. 313-322, 1986. 115

[79] Tiger. Available at: http://tiger.census.gov/ 126

List of Referred Publications

Referred International Journal

[J-1] Yasuhiko Morimoto, Mohammad Shamsul Arefin, Mohammad Anisuzzaman Siddique, Agent-Based Anonymous Skyline Set Computation in Cloud Databases, International Journal of Computational Science and Engineering (ISSN 1742-7185, an EI Indexed Journal), vol. 7, no. 1, pp. 73-81, 2012.

[J-2] Mohammad Shamsul Arefin and Yasuhiko Morimoto, Skyline Sets Queries for Incomplete Data, International Journal of Computer Science & Information Technology (ISSN: 0975 - 3826, an Inspec Indexed Journal), vol. 4, no. 5, pp. 67-80, 2012.

[J-3] Mohammad Shamsul Arefin, Xu Jinhao, Chen Zhiming, and Yasuhiko Morimoto, Skyline Query for Selecting Spatial Objects by Utilizing Surrounding Objects, Journal of Computers (ISSN 1796-203X, an EI Indexed Journal), vol. 8, no. 7, pp. 1742-1749, 2013.

[J-4] Mohammad Shamsul Arefin and Yasuhiko Morimoto, Privacy Aware Parallel Computations of Skyline Sets Queries from Distributed Databases, Computing and Informatics, (ISSN:1335-9150, an SCIE Indexed Journal), *to appear*.

Referred International Conference

[C-1] Mohammad Shamsul Arefin and Yasuhiko Morimoto, Privacy Aware Parallel Computation of Skyline Sets Queries from Distributed Databases, Second International Conference on Networking and Computing (ICNC 2011), pp. 186-192, Japan, 2011.

[C-2] Mohammad Shamsul Arefin and Yasuhiko Morimoto, Skyline Sets Queries from Databases with Missing Values, 22nd International Conference on Computer Theory and Applications (ICCTA 2012), pp. 24-29, Egypt, 2012.

[C-3] Xu Jinhao, Mohammad Shamsul Arefin, Chen Zhiming and Yasuhiko Morimoto, Real Estate Recommender: Location Query for Selecting Spatial Objects, Third International Conference on Networking and Computing (ICNC 2012), pp. 278-282, Japan, 2012.

[C-4] Chen Zhiming, Mohammad Shamsul Arefin and Yasuhiko Morimoto, Skyline Queries for Spatial Objects: A Method for Selecting Spatial Objects Based on Surrounding Environments, Third International Conference on Networking and Computing (ICNC 2012), pp. 215-220, Japan, 2012.

[C-5] Ma Geng, Mohammad Shamsul Arefin and Morimoto Yasuhiko, A Spatial Skyline Query for a Group of Users Having Different Positions, The Third International Conference on Networking and Computing (ICNC 2012), pp. 137-142, Japan, 2012.

[C-6] Mohammad Shamsul Arefin and Yasuhiko Morimoto, Skyline Queries for Sets of Spatial Objects by Utilizing Surrounding Environments, International Workshop on Databases in Networked Information Systems (DNIS 2013), LNCS 7813, Springer, pp. 293-309, 2013.

Other Papers (Not in Dissertation)

Referred International Journal

[O-1] Mohammad Shamsul Arefin and Yasuhiko Morimoto, Management of Multilingual Contents in Web Environment, International Journal of Computing Communication and Networking Research (ISSN: 1839-7212), Vol. 1, issue. 1, pp. 18-37, 2012.

[O-2] Mohammad Shamsul Arefin, Yasuhiko Morimoto, Mohammad Amir Sharif, BAENPD: A Bilingual Plagiarism Detector, Journal of Computers (ISSN 1796-203X, an EI Indexed Journal), vol. 8, no. 5, pp. 1145-1156, 2013.

[O-3] Hongbo Chen, Mohammad Shamsul Arefin, Zhiming Chen, Yasuhiko Morimoto, Place Recommendation Based on Users Check-in History for Location-Based Services, International Journal of Networking and Computing (ISSN 2185-2839, a DBLP Indexed Journal), vol. 3, no. 2, pp. 228-243, 2013.

Referred International Conference

[O-4] M. S. Arefin, Y. Morimoto, and A. Yasmin, Multilingual Content Management in Web Environment, International Conference on Information Science and Applications (ICISA 2011), pp. 1-9, South Korea, 2011.

[O-5] Yasuhiko Morimoto, Md. Anisuzzaman Siddique, and Md. Shamsul Arefin, Information Filtering by Using Materialized Skyline View, International Workshop on Databases in Networked Information Systems (DNIS 2011), LNCS 7108, pp. 179-189, 2011.

[O-6] M. S. Arefin, Y. Morimoto, and M. A. Sharif, Bilingual plagiarism detector, 14th International Conference on Computer and Information Technology (ICCIT 2011), pp. 451-456, Bangladesh, 2011.

[O-7] Mohammad Shamsul Arefin, Yasuhiko Morimoto, Chen Zhiming, Developing a Framework for Generating Bilingual English-Bangla Dictionaries from Parallel Text Corpus, International Conference on Electrical, Computer and Telecommunication Engineering (ICECTE 2012), pp. 549-552, Bangladesh, 2012.

[O-8] Chen Hongbo, Chen Zhiming, Mohammad Shamsul Arefin and Yasuhiko Morimoto, Place Recommendation from Check-in Spots on Location-Based Online Social Networks, Third International Conference on Networking and Computing (ICNC 2012), pp. 143-148, Japan, 2012.

[O-9] Mohammad Shamsul Arefin, Mohammad Amir Sharif, Yasuhiko Morimoto, BEBS: A Framework for Bilingual Summary Generation, 2nd International Conference on Informatics, Electronics & Vision (ICIEV 2013), pp. 1-7, Bangladesh, 2013.

[O-10] Zhichao Chang, Mohammad Shamsul Arefin, Yasuhiko Morimoto, Hotel Recommendation Based On Surrounding Environments, International Symposium on Applied Computing and Information Technology (ACIT 2013). (to appaer).