A Spatial Skyline Query for a Group of Users

Mohammad Shamsul Arefin†, Geng Ma‡, and Yasuhiko Morimoto‡
†Department of CSE, Chittagong University of Engineering and Technology (CUET), Bangladesh
‡Graduate School of Engineering, Hiroshima University, Japan
Email: sarefincse@gmail.com, [M115245, morimo]@hiroshima-u.ac.jp

***Abstract*— A skyline query finds objects that are not dominated by another object from a given set of objects. Skyline queries can filter unnecessary information efficiently and provide us important clues for various decision making tasks. Now a days, GPS devices and location based services are very popular and they can easily connect users and make groups. Conventional skyline queries are not sufficient to obtain valuable knowledge to fulfil the needs of such groups. Considering this fact, in this paper, we proposed a spatial skyline query for groups of users located at different positions. Our proposed skyline query algorithm selects a set of spatial objects to fulfil the groups' needs. For example, if a group wants to find a restaurant to hold a meeting, our method can select a convenient place for all users of the group. We performed several extensive experiments to show the effectiveness of our approach.**

***Index Terms*— Spatial skyline, Skyline for a group, Voronoi diagram.**

I. Introduction

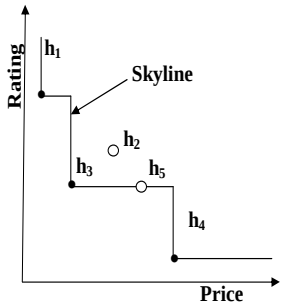
Given a k-dimensional database DB, a skyline query retrieves a set of skyline objects, each of which is not dominated by another object. An object p is said to dominate another object q if p is not worse than q in any of the k dimensions and p is better than q in at least one of the k dimensions. Figure 1 shows a typical example of skyline. The table in Figure 1 is a list of five hotels, each of which contains two numerical attributes "Price" and "Rating". 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. In database literature, there are many recent studies for efficient computation of skyline queries from databases [1]–[9]. All of these works just consider non-spatial information like price and rating.

Recently, GPS devices and location based services become popular. As a result, we have large databases containing spatial information. Therefore, we often have to select spatial objects from a spatial database. Conventional skyline queries are not sufficient to handle spatial objects. To solve the problem, spatial skyline queries have been proposed [14]–[21]. Most of those spatial skyline queries select a set of objects based on proximity from a given query point.

Different from other works, we consider a spatial skyline query for a group of users located at different positions. This is 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.

ID	Price	Rating
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. Skyline example

The problem of spatial skyline queries 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 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.

Social network services can connect users of different positions and make groups easily. Therefore, we often have to solve this spatial problem. Some of the existing spatial skyline queries consider the same spatial problem. However, most of those works only consider spatial information such as locations of the users and objects and do not take into account non-spatial features of objects, such as price and rating. Since both spatial and non-spatial features of objects are very important for efficient knowledge discovery tasks, we consider a method that can select objects based on both spatial and non-spatial features.

A. Motivating Example

Assume there is a database of restaurants as in Table I. The database has two non-spatial attributes: "Rating"

doi:10.4304/jsw.9.11.2938-2947

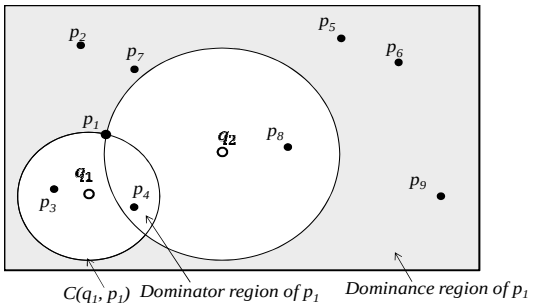


Figure 2. Spatial skyline example

TABLE I.
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

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 II.

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 III. 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. $Sum\text{-}Distance$ attribute in Table III 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 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 I and spatial information of Table III and obtain the information of Table IV. After computing Table IV, 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 III for each of different query, which are time-consuming and not affordable.

TABLE II.
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)

TABLE III.
SPATIAL ATTRIBUTES OF RESTAURANTS

ID	r-u_1	r-u_2	r-u_3	r-u_4	$Sum\text{-}Distance$
$\mathbf{r_1}$	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

In this paper, we consider an efficient method for computing such a spatial skyline query without constructing all the information of Table IV 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 throughout the paper.

The proposed method can be summarized as follows:

- First, we compute skyline objects based on "spatial sub-space" of the data points. In this step, we do not compute all values in Table III but compute only necessary distances to find dominated objects on the spatial sub-space.
- Next, we compute dominated objects on "non-spatial sub-space" of the data points.
- Then, we integrate those information to compute final skyline result.

We intensively evaluate our framework using both synthetic and real data and validate the effectiveness of our method.

The remainder of this paper is organized as follows. In Section II, we provide a brief survey of related works. In Section III, we describe some preliminary concepts related to our work. Section IV briefly explains over all procedure of the proposed skyline computation method. In Section V, we report our evaluation results, and finally this paper is concluded in Section VI.

II. RELATED WORKS

A. Skyline Computation

Skyline queries were originally considered for maximal vectors computation [1]. Borzsonyi et al. [2] first introduced skyline queries in database applications and proposed Block Nested Loop (BNL), Divide-and-Conquer, and B-tree based algorithms. Later, a number of different algorithms such as progressive skyline computation algorithm [3], nearest neighbor algorithm [4], branch and bound skyline (BBS) algorithm [5], and sort-filter-skyline (SFS) algorithm [6] were proposed for efficient skyline computation.

Due to the increase in data dimensionality, there have been many research efforts to address the dimensionality

TABLE IV.
NON-SPATIAL AND SPATIAL ATTRIBUTES OF RESTAURANTS

ID	Rating	Price	r-u_1	r-u_2	r-u_3	r-u_4	Sum-$Distance$	
$\mathbf{r_1}$	3	2	3.81	2.97	5.12	5.57	17.47	dominated by r_2, r_7
$\mathbf{r_2}$	2	2	2.55	2.69	1	2.33	8.57	not dominated
$\mathbf{r_3}$	3	4	2.92	2.01	2.24	3.77	10.4	dominated by r_7
$\mathbf{r_4}$	3	2	4.53	5.8	4.12	2.8	17.25	dominated by r_7
$\mathbf{r_5}$	2	3	1.58	2.97	2.24	1.02	7.81	not dominated
$\mathbf{r_6}$	3	3	2.55	1.56	3.61	4.32	12.04	dominated by r_7
$\mathbf{r_7}$	3	1	0.71	0.89	1.41	1.48	4.49	not dominated
$\mathbf{r_8}$	3	2	4.30	5.52	5.39	4.08	19.29	dominated by r_2, r_7, r_9
$\mathbf{r_9}$	2	2	2.54	3.8	2.24	0.8	9.38	not dominated
$\mathbf{r_{10}}$	1	1	5.15	5.18	3.61	4.08	18.38	not dominated

problem of skyline queries such as skyline frequency [7], k-dominant skylines [8], and k-representative skylines [9].

All these efforts, however, do not consider spatial relationships between data points.

B. Spatial Skyline Query

Spatial query processing was first studied for ranking neighboring objects. Several works [10]–[12] considered spatial query mechanism for ranking neighboring objects using the distance to a single query point. Papadias et al. [13] considered ranking of objects using aggregate distance of multiple query points.

Sharifzadeh et al. [14] 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. [15] presented a simple and efficient algorithm that can compute the correct results. Guo et al. [16] 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 skyline can not handle more than one query point. Kodama et al. [17] 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. [18] 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 [19], 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 [18]. Huang et al. [20] proposed two distance-based skyline query techniques those can efficiently compute skyline queries over road networks. Zheng et al. [21] proposed a query processing method to produce spatial skylines for location-based services. They focus on location-dependent

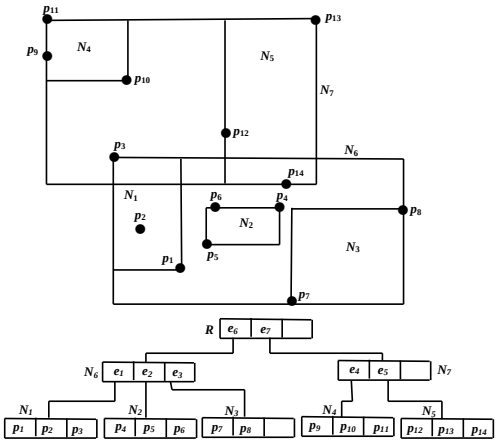


Figure 3. Example of an R-tree

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.

None of the above works considered the computation of spatial skyline objects for a group of users based on both spatial and non-spatial information. In this paper, we consider the issue and propose an efficient method for computing such spatial skyline objects.

III. PRELIMINARIES

A. Skyline Queries

Let p and q be objects in a database 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.

B. Spatial Skyline Queries

Assume that there are two point sets. One is a set of data points, say P, and the other is a set of query points, say Q. We also assume that each point in P and Q has spatial attributes, which are 2-dimensional coordinate attributes. Let us also consider that the distance function $d(p, q)$ returns the Euclidean distance between a pair of points p and q.

Definition 1: We say that p_1 "spatially dominates" 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 $q \in Q$.

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.

C. R-Tree

R-tree is the most prominent index structure widely used for spatial query processing. Figure 3 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 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.

D. 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. In this paper, 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 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, 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 [22] 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 $O(n \log n)$.

E. VoR-Tree

A VoR-tree [23] 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.

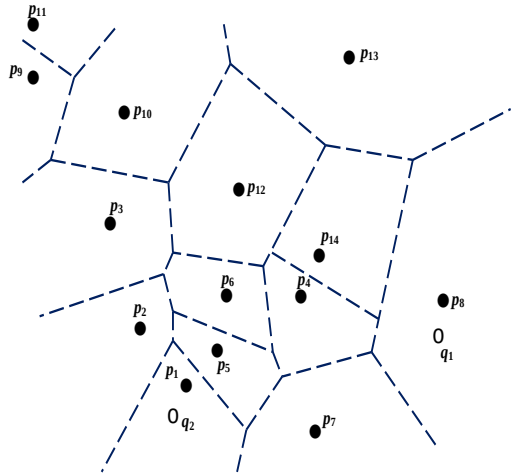


Figure 4. Example of a Voronoi diagram

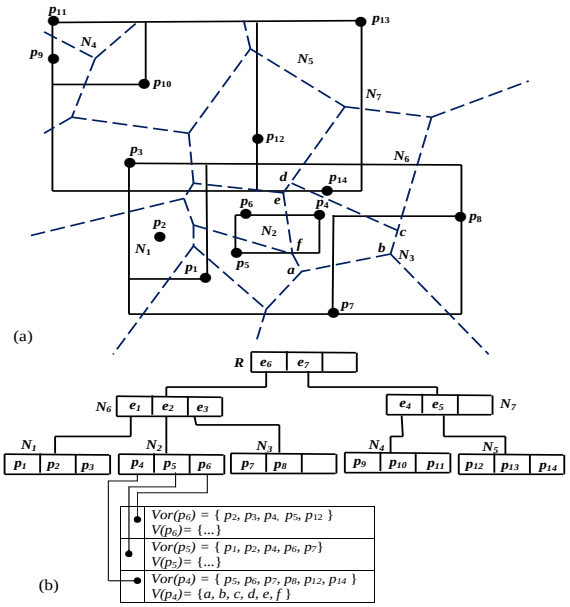


Figure 5. (a) Voronoi diagram (b) VoR-tree (adapted from [22])

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 5(b) shows an example of VoR-tree for the data points of Figure 3. Each rectangular in Figure 5 is a node of the VoR-tree. In Figure 5, 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 5 (b), we can see that data point p_5, p_6, p_7, p_8, p_{12}, and p_{14} are Voronoi neighbors 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 of our query. We give detail explanation of how VoR-tree improves our query performance in subsection IV-A.

IV. Query Processing

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

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

In the first phase, we compute skyline results in the spatial sub-space like ($r-u_1$, $r-u_2$, $r-u_3$, $r-u_4$, $Sum\text{-}Distance$) of Table IV. We utilize the concept of $Sum\text{-}Distance$ 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 IV. 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.

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

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 II, we can find that the centroid of $r\text{-}u_1$, $r\text{-}u_2$, $r\text{-}u_3$, and $r\text{-}u_4$ is (5.13, 5. 28). From Table I, we can find that r_j is nearest to (5.13, 5. 28). So, we

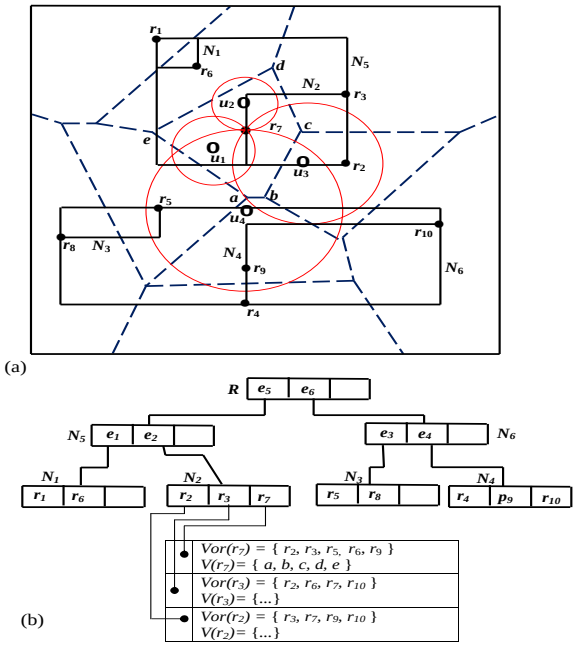


Figure 6. (a) Location of users and restaurants (b) VoR-tree

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".

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

TABLE V.
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 VI.
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	$\oslash$	$\{r_7, r_5, r_2, r_9\}$

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 6. In the Figure 6(a), white dots are locations of four users and black dots are locations of restaurants. We first pick up r_7 and compute $C(u_i, r_7)$ 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 V. In the process, we keep the heap data structure like Table VI.

Looking at the information of Table V, 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, 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 VI. 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.

TABLE VII.
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 VIII.
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

B. Non-spatial Processing

In non-spatial processing, at first, we collect all dominated data points at spatial sub-space. Table VII shows such data points with non-spatial information. From Table VII, 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 subspace, we check their dominance against the skyline objects r_2, r_5, r_7, and r_9 of spatial sub-space. Table VIII shows non-spatial information of these skyline objects in spatial sub-space. Note that objects of Table VIII are in the final skyline as well.

If we check the objects of Table VII against the objects of Table VIII, 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 VIII and there is no other non-dominated object in Table VII. 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 1 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 1** 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).

C. Correctness of Algorithm

The correctness of **Algorithm 1** follows some basic properties of geometry and skyline query. From **Algorithm 1**, 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

Algorithm 1 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, $S \subseteq R$

1: **begin**
2: set $D, (D \subseteq R)$ = the set of dominated objects in spatial sub-space
3: select a data point r_j that is closest to the centroid of the query points $U = \{u_1, u_2, \cdots, u_i\}$
4: compute the search region of r_j
5: obtain the data points set, say T within the "search region", $T \subseteq R$
6: compute the "Sum Distance" $dist_k$ of each data point r_k, $r_k \in T$
7: select the data point r_k that has minimum "Sum Distance"
8: add (r_k, $dist_k$) to the heap H
9: 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"
10: remove (r_k, $dist_k$) from H and add r_k to S
11: **repeat**
12: choose the top element, say r_l from H
13: 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"
14: pop $(r_l, dist_l)$ from H
15: **if** r_l is not dominated by some other objects in S in spatial sub-space **then**
16: add r_l to S
17: **else**
18: add r_l to D
19: **end if**
20: **until** H becomes empty
21: **for** each data point $r_m \in D$ **do**
22: **if** r_m is dominated by some other objects of S in non-spatial sub-space **then**
23: $r_m \notin S$
24: **else if** r_m is dominated by some other objects of D in non-spatial sub-space **then**
25: $r_m \notin S$
26: **else**
27: add r_m to S
28: **end if**
29: **end for**
30: return S as the spatial skyline result
31: **end**

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 1** 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.

TABLE IX.
DATASETS FOR EXPERIMENTS

Datasets	Total Objects	Density
r	50,747	–
$\mathbf{s_1}$	80,000	0.08
$\mathbf{s_2}$	50,000	0.05
$\mathbf{s_3}$	20,000	0.02

Line 21-29 of **Algorithm 1** shows the computation of skyline objects in non-spatial sub-space. The correctness of **Algorithm 1** 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.

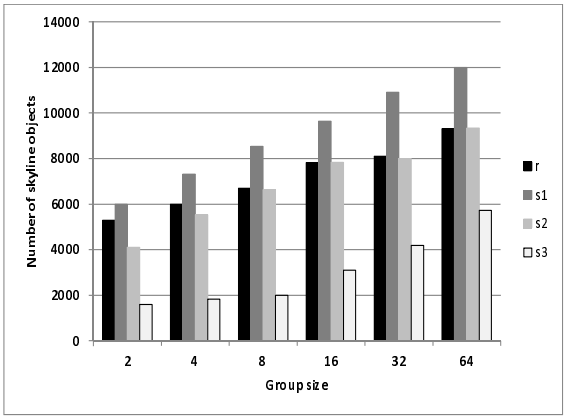


Figure 7. Number of skyline objects

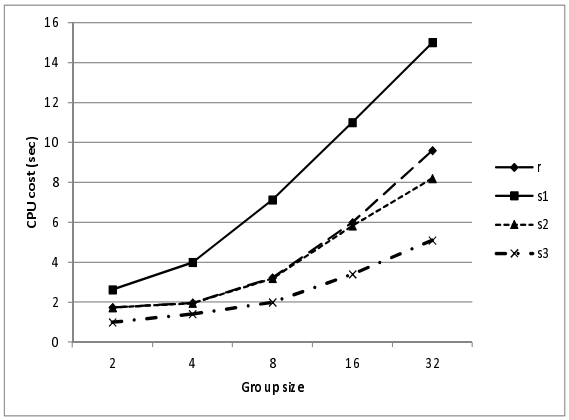


Figure 8. Running time varying group size

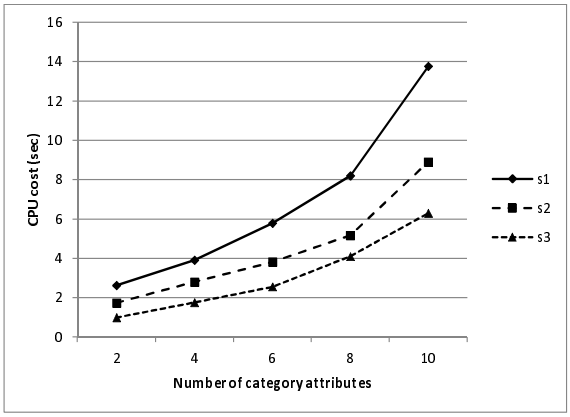


Figure 9. Running time varying number of category attributes

V. 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. Our developed system is able to handle large volume of data containing both spatial and non-spatial information.

A. 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 [24]. 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] $\times$ [0, 1000] space. There are three synthetic datasets $s1$, $s2$, and $s3$ with different densities normalized in [0,1000] $\times$ [0,1000] space as in Table IX. In Table IX, r stands for real dataset of TIGER 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.

B. Experimental Results

The first experiment studies the numbers of skyline objects under different densities and different group size. Figure 7 shows the total numbers of skyline objects from datasets r, s_1, s_2, and s_3. From Figure 7, 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 8, 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 9 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 spatial and non-spatial attributes. From the result of Figure 10, we can see that our algorithm (VR) significantly outperforms BBS algorithm.

Next experimental results are shown in Figure 11. It shows the relative dominance check between our algorithm and BBS algorithm. From Figure 11, we can see that our algorithm constantly performs less number of dominance check compared with BBS algorithm.

Figure 12 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 12, we can see that in case of fixed number of users and more category attributes, the performance of our algorithm is still better

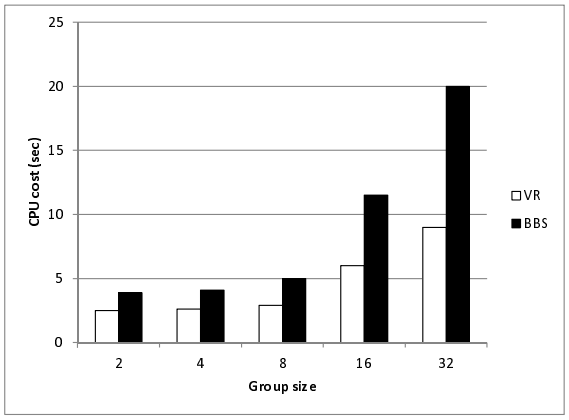


Figure 10. Comparative performance in running time varying group size

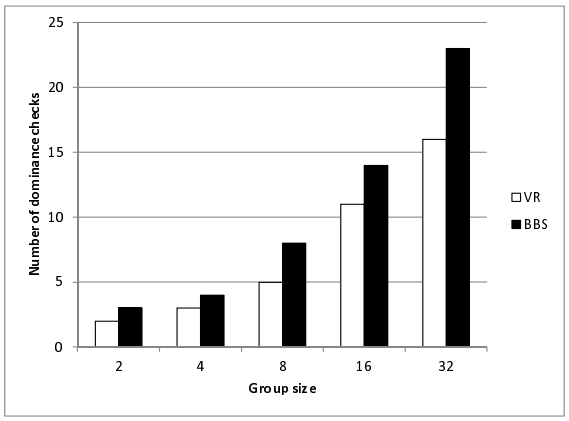


Figure 11. Comparative performance in dominance check varying group size

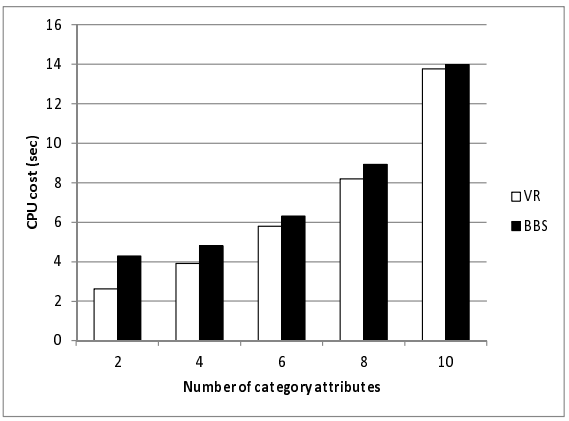


Figure 12. Comparative performance in running time varying number of category attributes

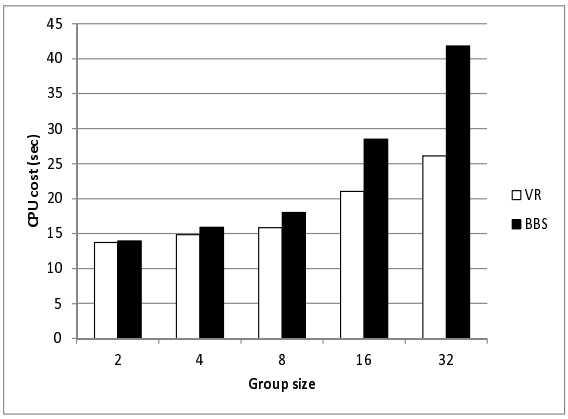


Figure 13. Comparison in running time varying group size for large number of category attributes

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 13, we can find that our algorithm becomes comparatively better than BBS algorithm with an increase in group size.

VI. Conclusion

In this paper, we proposed a framework for computing skyline of spatial objects for a group of users located at different locations. In the proposed framework, different from existing works, we took into account not only spatial features, but also non-spatial features of the objects.

Recently, many social network services create groups considering users located in different places. Spatial skyline queries for a group can be able to play an important role in such environments.

In our computation 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 the proposed algorithm is scalable enough to handle large and high dimensional datasets.

In this paper, 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.

References

[1] H. T. Kung, F. Luccio, and F. Preparata, "On finding the maxima of a set of vectors", *Journal of the Association for Computing Machinery*, vol. 22 no. 4, pp. 469-476, 1975.

[2] S. Borzonyi, D. Kossmann, and K. Stocker, "The skyline operator", *In Proc. of the 17th International Conference on Data Engineering*, pp. 421-430, 2001.

[3] K. L. Tan, P. K. Eng, and B. C. Ooi, "Efficient progressive skyline computation", *In Proc. of the 27th International Conference on Very Large Data Bases*, pp. 301-310, 2001.

[4] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries", *In Proc. of the 28th International Conference on Very Large Data Bases*, pp. 275-286, 2002.

[5] D.Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries", *In Proc. of ACM SIGMOD International Conference on Management of Data*, pp. 467-478, 2003.

[6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting", *In Proc. of the 19th International Conference on Data Engineering*, pp. 717-719, 2003.

[7] C. Y. Chan, H. Jagadish, K. Tan, and A. K. Tung, Z. Zhang, "On high dimensional skylines", *In Proc. of LNCS, Springer, Heidelberg*, pp. 478-495, 2006.
[8] C. Y. Chan, H. Jagadish, K. L. Tan, and A. K. Tung, Z. Zhang, "Finding k-dominant skylines in high dimensional space", *In Proc. of ACM SIGMOD International Conference on Management of Data*, pp. 444-457, 2006.
[9] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting stars: The k most representative skyline operator", *In Proc. of the 23rd International Conference on Data Engineering*, pp. 86-95, 2007.
[10] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries", *In Proc. of the ACM SIGMOD International Conference on Management of Data* , pp. 71-79, 1995.
[11] 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* , pp. 78-86, 1997.
[12] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?", *In LNCS, Springer, Heidelberg*, vol. 1540, pp. 217-235, 1999.
[13] 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, pp. 529-576, 2005.
[14] M. Sharifzadeh, and C.Shahabi, "The spatial skyline queries", *In Proc. of the 32nd International Conference on Very Large Data Bases*, pp. 751-762, 2006.
[15] 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*, pp. 247-264, 2009.
[16] X. Guo, Y. Ishikawa, and Y. Gao, "Direction-based spatial skylines", *In Proc. of the ACM SIGMOD International Conference on Management of Data*, pp. 73-80, 2010.
[17] K. Kodama, Y. Iijima, X. Guo, and Y Ishikawa, "Skyline queries based on user locations and preferences for making location-based recommendations" *Proc. of International Workshop on Location Based Social Networks*, pp. 9-16, 2009.
[18] 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*, pp. 796-805, 2007.
[19] M. Safar, D. E. Amin, and D. Taniar, "Optimized skyline queries on road networks using nearest neighbors", *Journal of Personal and Ubiquitous Computing*, vol. 15 no. 8, pp.845-856, 2011.
[20] Y. K. Huang, C. H. Chang, and C. Lee, "Continuous distance-based skyline queries in road networks" *Journal of Information Systems*, vol. 37, pp. 611-633, 2006.
[21] 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.
[22] S. Fortune, "A sweep line algorithm for Voronoi diagrams", *In Proc. of the Second Annual Symposium on Computational geometry*, pp. 313-322, 1986.
[23] M. Sharifzadeh, and C. Shahabi, "VoR-Tree: R-trees with Voronoi diagrams for efficient processing of spatial nearest neighbor queries", *In Proc. of the 36th International Conference on Very Large Data Bases*, pp. 1231-1242, 2010.
[24] Tiger. Available at: http://tiger.census.gov/

Mohammad Shamsul Arefin received his B.Sc. Engineering in Computer Science and Engineering from Khulna University, Khulna, Bangladesh in 2002, and completed his M.Sc. Engineering in Computer Science and Engineering in 2008 from Bangladesh University of Engineering and Technology (BUET), Bangladesh. He received his Doctor of Engineering Degree from Hiroshima University with support of the scholarship of MEXT, Japan in 2013. He is a member of Institution of Engineers Bangladesh (IEB) and currently working as an Associate Professor in the Department of Computer Science and Engineering, Chittagong University of Engineering and Technology, Chittagong, Bangladesh. His research interest includes privacy preserving data mining, cloud privacy, multilingual data management, semantic web, and object oriented system development.

Geng Ma received his M.Sc in Information Engineering from Hiroshima University, Japan in 2013. His current research interest includes spatial databases, preference-based queries, and recommendation systems.

Yasuhiko Morimoto is an Associate Professor at Hiroshima University. He received B.E., M.E., and Ph.D. from Hiroshima University in 1989, 1991, and 2002, respectively. From 1991 to 2002, he had been with IBM Tokyo Research Laboratory where he worked for data mining project and multimedia database project. Since 2002, he has been with Hiroshima University. His current research interests include data mining, machine learning, geographic information system, and privacy preserving information retrieval.