MapReduce-Based Computation of Area Skyline Query for Selecting Good Locations in a Map

Chen LI†, Annisa ††, Asif ZAMAN†††, and Yasuhiko MORIMOTO†

† Graduate School of Engineering, Hiroshima University
1-7-1 Kagamiyama, Higashi-Hiroshima City, Hiroshima 739-8521, JAPAN
†† Bogor Agricultural University, Indonesia
††† University of Rajshahi, Bangladesh
E-mail: †{D165000,morimo}@hiroshima-u.ac.jp, ††annisa@myfamily.web.id, †††asifzaman3180@yahoo.com

Abstract Selection of good locations in a map is an indispensable function in many applications. To select specific locations, we have to specify detailed selection criteria. However, it is not easy especially for users of mobile devices. Therefore, we used an idea of skyline queries, which are known to be easy and effective to retrieve interesting data from a database. In our previous work, we have proposed area skyline query that selects good locations in a map. However, the query is not fast enough for handling "big data." We simplify and revise the algorithm of the query in this paper by using MapReduce framework so that we can use it for big data. Experiments' results demonstrate that the performance and scalability are superior to previous area skyline algorithm and can handle big data.

Key words area skyline; grid structure; MapReduce;

1. Introduction

Many people are using GPS equipped mobile devices. For utilizing the GPS function, there are many map applications for mobile devices. Selection of good locations is an indispensable function for such map applications. In general, we have to specify detailed selection criteria to select specific locations. However, it is not easy especially for users of mobile devices. Therefore, we used an idea of skyline queries, which are widely used in information retrieval.

Skyline Query

In general, a good location is close to train stations, shopping malls, and sightseeing spots, etc. Furthermore, it should be far away from the facilities such as factories, noise sources, etc. Skyline query [5] is known to be effective to retrieve interesting data from a database. Specifically, given a set of points $p_1, p_2, ..., p_n$, the skyline query returns a small number of points $\mathbf{P}$, each point $p_i \in \mathbf{P}$ is not dominated by other points in the database.

A skyline example is demonstrated in Table 1 and Figure 1. Table 1 shows the reference price of some hotels and the distance between hotels and somewhere where customers prefer to visit such as sightseeing spots or coast, etc. According to these two numerical attributes of the hotels, we can retrieve the skyline objects which could help customers to make a relatively wise choice. In general, customers would like to choose the hotel with lower price and smaller distance.

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

Table 1 A Hotel Example

In Figure 1, $\{h_1, h_3, h_4\}$ are the skyline objects. The price and distance are smaller than other hotels. In other words, skyline objects $\{h_1, h_3, h_4\}$ are not dominated each other. In addition, h_2 and h_5 are dominated by h_3.

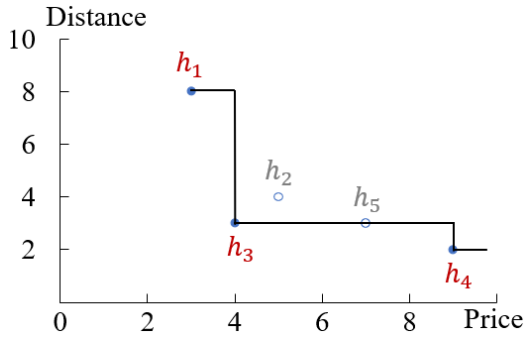

Figure 1 Conventional Skyline Example

A number of algorithms which have been recorded in [5], [7], [11], [15], [21] are effective to compute skyline objects adequately. However, most of the existing skyline algorithms have not been devised for spatial data and thus do not consider spatial relationships between objects. So it is difficult

for skyline query to calculate two-dimensional data such as locations.

Spatial Skyline Query

Figure 2 is an example of a map. In the example, star marks, triangle marks, and square marks are places of facilities. We annotate "+" mark for the preferable facilities such as sightseeing spots for a tourist and "−" mark for the unpreferable facilities such as noisy factories. Star marks, denoted as $F1^+ = \{f1_1^+, f1_2^+, f1_3^+\}$, and triangle marks, denoted as $F2^+ = \{f2_1^+, f2_2^+, f2_3^+\}$, are preferable facilities and square marks, denoted as $F3^- = \{f3_1^-, f3_2^-, f3_3^-\}$, are unpreferable facilities.

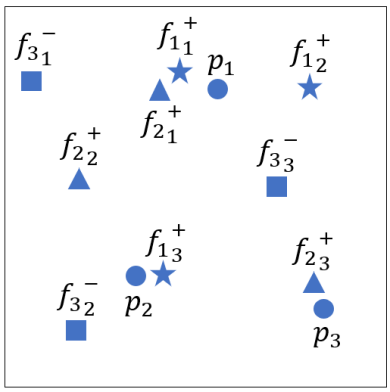

Figure 2 Some Facilities in a Map

Table 2 shows the closest distances between the candidate points and facilities. For example, $f1_1^+, f2_1^+, f3_3^-$ are the closest facilities to the candidate point p_1, which the distance are 3, 5 and 10 respectively. We multiply -1 for each distance value of the unpreferable facilities, so that we can say that the smaller value is better. In the table, point p_1 dominates p_2. p_3 is not dominated by p_1 and p_2, since p_3 is closer to $F2^+$ than p_1, p_2 and farther to $F3^-$ than p_2. Note that the spatial skyline problem can be addressed by conventional skyline algorithms after calculating Table 2.

Point	$F1^+$	$F2^+$	$F3^+$
p_1	3	5	-10
p_2	4	9	-7
p_3	8	1	-8

Table 2 Distance Table

In some practical situations, the point set **P** does not exist. It means we have to find a location where should be closer to all preferable facilities and be far away to all unpreferable facilities. In our previous work [2], we have proposed area skyline query that selects good locations in a map to solve such problems.

Area Skyline Query

Assume a traveler would like to choose a destination which is closer to good touristic places and is far from unsafe places. In the situation, we can define *dominance relationship* between two places say p_1 and p_2 as follows:

Definition 1.1. *(Area Dominance) We divide a map into $n \times n$ small square grids $G = \{g_{1,1}, ..., g_{n,n}\}$. A grid g_i is said to dominate another grid g_j in a map if distance to the nearest touristic place from g_i is smaller than that of g_j and distance to the nearest unsafe place from g_i is larger than that of g_j.*

Definition 1.2. *(Area Skyline) Area Skyline is a set of grids, each of which is not dominated by another grid in a map.*

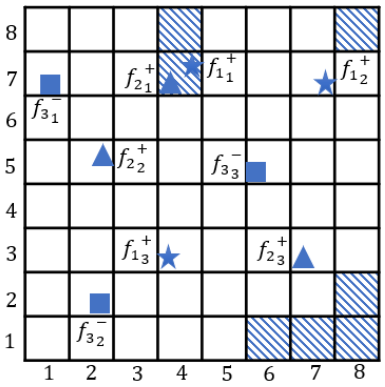

Figure 3 Area Skyline Example

Figure 3 is an example of a map. In the map, shaded grids are in the area skyline computed by the algorithm proposed in [2], which we call it grid-based area skyline (GASKY). Each shaded grid is not dominated by other grids. On the other hand, the unshaded grids are the dominated grids, and we can eliminate such grids from candidates.

By using area skyline queries, we can easily select good areas from a map. However, the complexities of those area skyline queries are much higher than conventional skyline queries. The response time of area skyline query [2] increases linearly with the number of facilities are growing. In this paper, we simplify and revise the algorithm by using MapReduce framework to reduce the response time of area skyline query.

The rest of this paper is structured as follows. Section II describes our MapReduce-based algorithm. Section III gives an overview about related works on skyline queries, spatial skyline queries, and parallel skyline query algorithms. The experiments are conducted in section IV. Finally, some conclusions and directions for future work are reported in section V.

2. Area Skyline on MapReduce

In this section, we propose a novel algorithm which is based on MapReduce framework, called "MRGASKY", to reduce the response time of area skyline query.

Division of Grids

We are focused on using a grid structure to divide a map

into $n \times n$ grids on average for simplicity. In order to facilitate the distinction between different grids, we give each grid an unique ID from bottom-left $g_{1,1}$ to top-right $g_{n,n}$ (see Figure 3). We assume that the facilities inside the grids can be represented by the grids as the grids are small enough.

MRGASKY Algorithm

Our proposed algorithm consists of following two steps:

Step1. Map function computes the distance to the closest facilities in the same row:

Step1.1 The Map function reads girds in the i-th row from left and right respectively to compute the distance. We assume that the value of a grid is infinite unless the first facility is encountered. And the distance of the grid is considered as 0 when the facility is encountered. Then the next grid is based on the previous grid plus one until the next facility is encountered. For example, $f1_1^+$ in 7-th row of Figure 3 is illustrated in Figure 4.

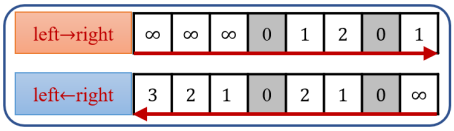

Figure 4 Example of Step 1.1 Process

Step1.2 We select a minimum value of calculated two distances as the final value of the distance, as illustrated in Figure 5.

Figure 5 Example of Step 1.2 Process

The algorithm of Map function showed in Figure 6. The input data stored in Hadoop distributed file system (HDFS) are formed as a binary image. Specifically, the binary image of size $n \times n$ maintained in an array $m_{i,j}(1 \leqq i, j \leqq n)$. $m_{i,j} = 1$ represents that the grid $g_{i,j}$ is a facility and $m_{i,j} = 0$ when there is not any facility inside $g_{i,j}$. The output of Map function are the key-value pairs. The keys are the column IDs and row IDs of $g_{i,j}$, and the values are the distances calculated in step 1.

Algorithm of Map Function

Input: input data in HDFS
Output: (key, values) = (column ID and Row ID of grids, distance)

1. for each line in HDFS do
2. calculate Euclidean distance from left to right: $dist_{left \to right}$
3. calculate Euclidean distance from right to left: $dist_{left \leftarrow right}$
4. for each grid in the same row do
5. $dist_{min} = min(dist_{left \to right}, dist_{left \leftarrow right})$

Figure 6 Algorithm of Step 1 Process

Figure 7 shows the results of type $F1^+$ of step 1 process. The distance of every grid in the same row is computed in this step.

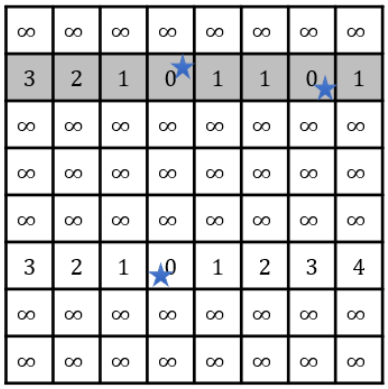

Figure 7 Example of Step 1 Process

Step2. Again, we compute Euclidean Distance to the nearest facility along with column-wise in Reduce function:

Step2.1 The Reduce function reads grids in the same column based on the results of step 1(see Figure 7). We map every i-th column to a two-dimensional coordinate where x-axis represents the grids which are sorted from bottom to top, and y-axis represents the distance of step 1. The points are saved in a stack. It should be clear that each column has its own corresponding stack. We bisect the adjacent two points p_i, p_j, and name the intersection of the perpendicular bisector line $\overline{p_i p_j}$ and x-axis as x_{ij}. x_{ij} can be computed by formula (1):

$$x_{ij} = \frac{(y_j^2 - y_i^2) + (x_j^2 - x_i^2)}{2(x_j - x_i)} \tag{1}$$

where (x_i, y_i) and (x_j, y_j) are the coordinates of point p_i and p_j. If $x_{ij} > x_{jk}$ $(i < j < k)$, the point p_j then be deleted from stack, otherwise point p_j is stored into the stack. Figure 8 and 9 showed the two cases whether delete p_j or not.

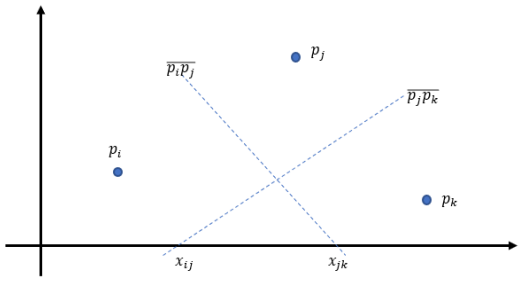

Figure 8 Example of deleting point p_j

Figure 10 showed an example of 5-th column in Figure 7 of step 2.1 process. x_{12} is larger than x_{23}, so we delete point p_2 from the stack. And then we compare x_{13} and x_{34}. Finally, all the points except p_3 and p_7 are deleted from the stack.

Step2.2 For every i-th column, we calculate the Euclidean Distance of deleted points (e.g., $p_1, p_2, p_4, p_5, p_6, p_8$ in Figure 11). Specifically, we determine proximate intervals [14] of left points (e.g., p_3 and p_7) of step2.1. We bisect p_3 and p_7, the intersection of $\overline{p_3 p_7}$ and the x-axis is $x_{37} = 5$. It means the

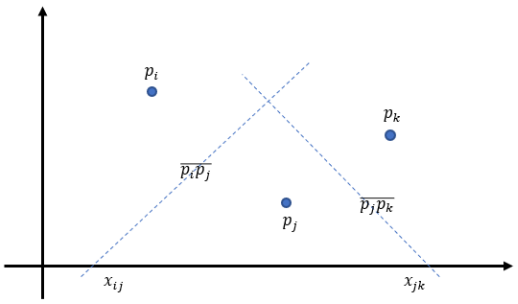

Figure 9 Example of maintaining point p_j

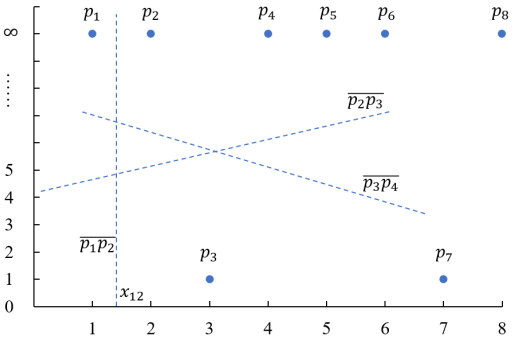

Figure 10 Example of Step 2.1 Process

point p_3 dominate points p_1, p_2, p_4 and p_5. Point p_7 dominate points p_6, p_8. Figure 11 showed the two intervals of 5-th column of Figure 7. Then the Euclidean Distance of every grid can be calculated in the obvious way.

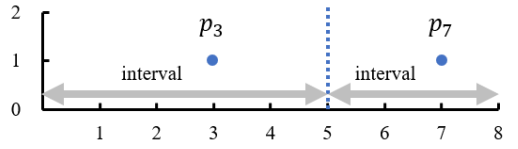

Figure 11 Example of Step 2.2 Process

The algorithm of Reduce function showed in Figure 12. The input data of this phase are the key-value pairs which the keys are column IDs and the values, are sorted distances computed in Map function. The output data are area skyline objects such as the shaded grids showed in Figure 3.

Figure 13 shows the results of type $F1^+$ of step 2 process. Intuitively, the grids with the same color are closest to the corresponding facility of $F1^+$ type. And the distances of white girds to $f1_1^+$ and $f1_3^+$ are same.

Facility type $F2^+$ and $F3^-$ can be calculated by the same process in MapReduce framework. Figure 14 demonstrate MapReduce data flow of our proposed algorithm of $F1^+$ and $F2^+$. The input data stored in HDFS are formed as facility type, row ID, the value of binary image in the same row. Then the data are sent to Map function by key-value pairs. The outputs of Map function produced new key-value pairs. The intermediate key-value pairs are then grouped and sorted with the same intermediate key. After sorted & shuffle phase, the values of the same column are grouped

Algorithm of Reduce Function

Input: the results of sorted and shuffle phase
Output: area skyline objects

```
1.    for each grid in same column do
2.        if x_ij > x_jk do
3.            delete the point p_j from stack
4.    for all the maintained points do
5.        calculate the proximate interval of each point on x-axis
6.        calculate the Euclidean distance of the points in the interval
7.
8.    //remove dominated grids
9.    for the record r_i of the Euclidean distance for all types of each grid do
10.       if r_i < r_j do
11.           remove dominated grid from the record
12.   return skyline objects
```

Figure 12 Algorithm of Step 2 Process

together. Then the grouped data are sent to Reduce function. Reduce function calculate the Euclidean Distance to the closest facility in column wise.

$\sqrt{10}$	$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{2}$	1	$\sqrt{2}$
3	2	1	0	1	1	0	1
$\sqrt{10}$	$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{2}$	1	$\sqrt{2}$
$\sqrt{13}$	$\sqrt{8}$	$\sqrt{5}$	2	$\sqrt{5}$	$\sqrt{5}$	2	$\sqrt{5}$
$\sqrt{10}$	$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$	3	$\sqrt{10}$
3	2	1	0	1	2	3	4
$\sqrt{10}$	$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$	$\sqrt{10}$	$\sqrt{17}$
$\sqrt{13}$	$\sqrt{8}$	$\sqrt{5}$	2	$\sqrt{5}$	$\sqrt{8}$	$\sqrt{13}$	$\sqrt{20}$

Figure 13 Example of Step 2 Process

3. Experiments

In this section, we conducted on comparing the performance of GASKY and MRGASKY algorithms. Experiments of GASKY is conducted on Linux operating system with Intel Core i7 3.40GHz processor with 4GB of RAM. And using this PC as one of four compute nodes. The other three nodes conduct on Linux operating system with Intel Core 2 3.16GHz and 2.13GHz processors, 4GB RAM. MRGASKY algorithm is implemented on Hadoop 2.5.2. Also, we used synthetic datasets to evaluate our algorithm. Each experiment is repeated ten times, and we evaluated average processing time as the performance indicator. Since the step of removing dominated areas for both GASKY and MRGASKY algorithms is same, and the performance of this step is not different from other conventional skyline algorithms, we excluded it from the processing time calculation.

Effect on Grid Number

In these experiments, we used two sets of synthetic

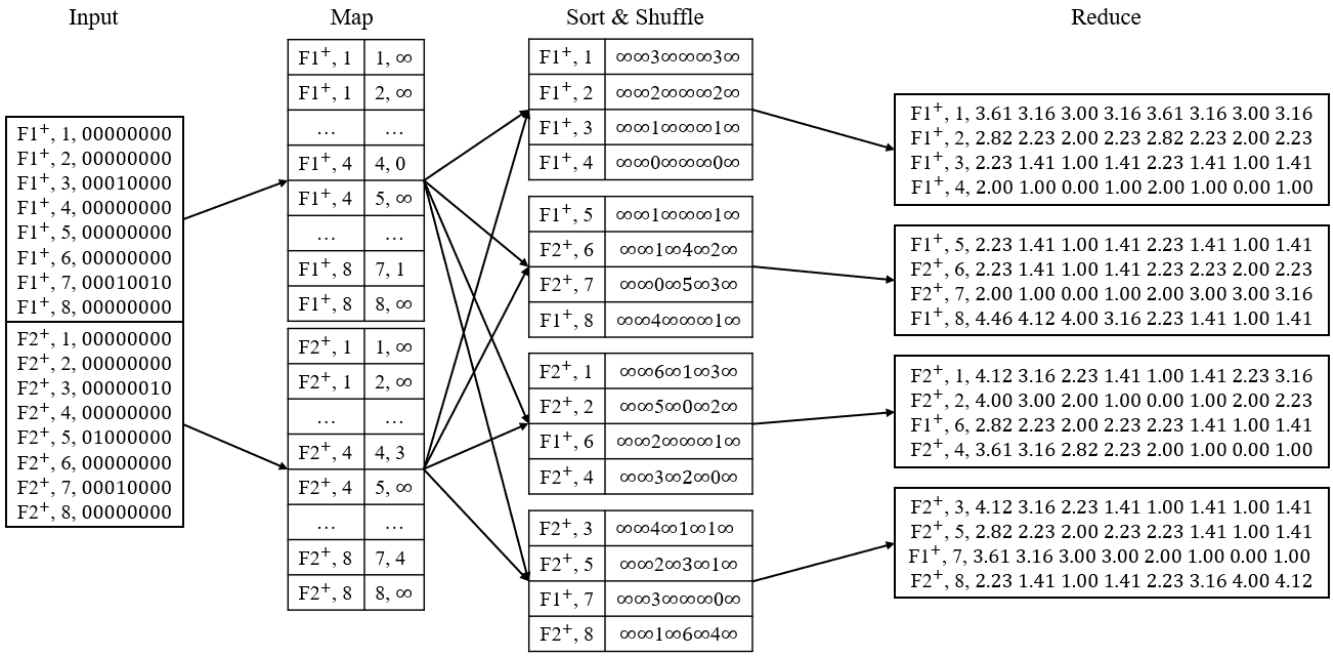

Figure 14 MapReduce data flow of MRGASKY algorithm

datasets, said DS_A1 and DS_A2. DS_A1 consists of 32 objects, and four types of facilities which two types are preferable facilities and the other 2 are unpreferable facilities. We varied number of grids with 32×32, 64×64, 128×128, 512×512 and 1024×1024. For DS_A2, we fixed the number of facility types as 2 and fixed the number of objects as 2000. We varied the number of grids as 100×100, 500×500 and 1000×1000. We compared the effect on grid number in Figure 15 and Figure 16.

In Figure 15, we can observe that the processing time of GASKY increases faster than MRGASKY when the number of grids is larger than 256×256. The reason is GASKY taking more time in $min - max$ computation when the number of grids increased.

In Figure 16, when we raised the number of objects to 2000, we can observe that the processing time of GASKY increases faster than MRGASKY. The reason is that GASKY spent more time on building Voronoi Diagram and $min-max$ computation. Thus MRGASKY has better scalability than GASKY since grid number increasing.

Effect on Facility Types

For experiment DS_B in Figure 17, we fixed the number of objects and grids as 10000 and 128×128. Besides, we varied the number of types to 2, 4, 6 and 8 respectively. And the number of preferable facility types is set to be same as the number of unpreferable facility types. Varying with the number of types, the processing time of GASKY increases linearly. The curve of MRGASKY is under GASKY and tends to stable. This result illustrated that performance of MRGASKY is also better than GASKY varying with facility types.

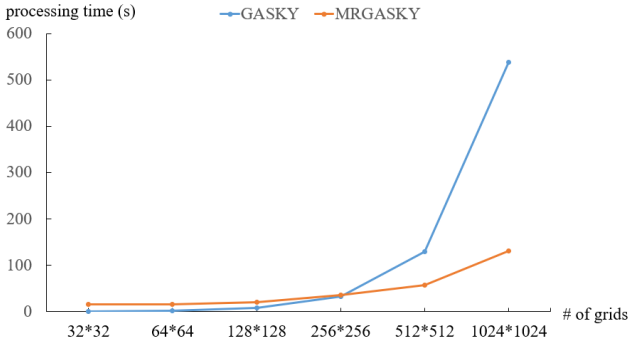

Figure 15 Processing time of DS_A1

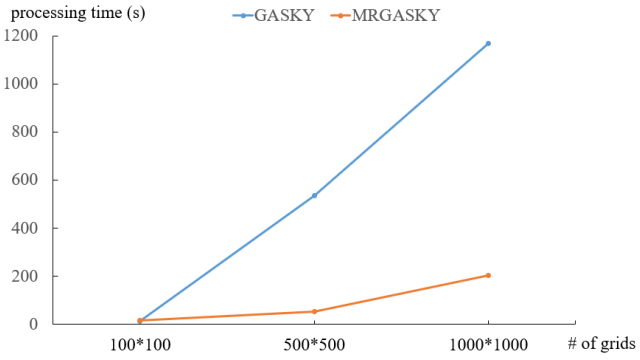

Figure 16 Processing time of DS_A2

Effect on Object Number

For the effect on object number, the experiment DS_C in Figure 18, we set the number of types as 2 and the grid size as 128×128. We raised the number of objects to 4000, 8000, 12000 and 16000. The results demonstrated that the processing time of the proposed algorithm is much smaller

than previous works and and maintained stability enough to handling ”big data”.

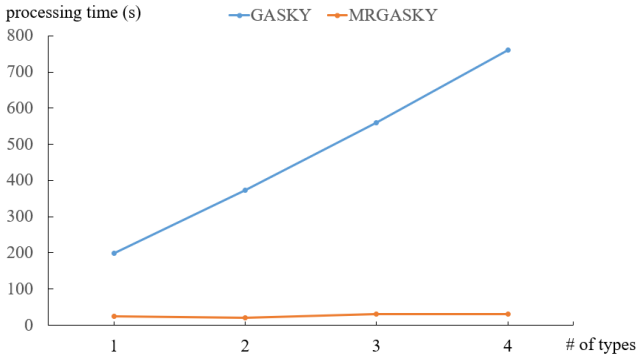

Figure 17 Processing time of DS_B

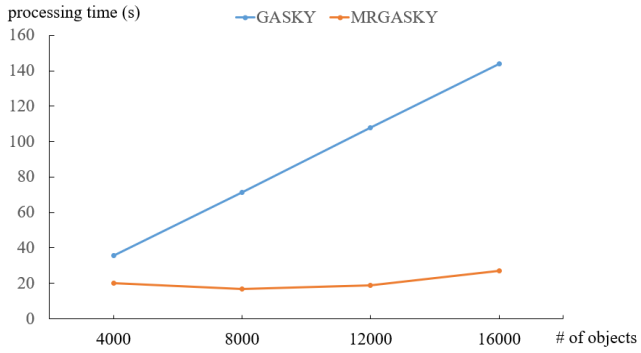

Figure 18 Processing time of DS_C

4. Related Works

Skyline Query

Skyline queries are focused on filtering out a set of points from a large set of data points. The points which are not dominated by other points are interesting. Skyline operator is firstly proposed by Brozsonyi et al. [5]. They presented two algorithms, say *Block-Nested-Loops*(BNL) and *Divide-and-Conquer* (D&C). Furthermore, they also discussed B-trees and R-trees on evaluating skyline queries. Chomicki et al. proposed a skyline algorithm, *Sort-Filter-Skyline* (SFS), based on presorting BNL [7]. Tan at al. presented two progressive algorithms, *Bitmap* and *Index*, to improve the performance of skyline computing [18]. A widely used effective algorithm nowadays, *Brach and Bound Skyline* (BBS), which is based on nearest neighbor search was developed by Papadias et al. [21].

Spatial Skyline Query

Spatial Skyline Queries (SSQ) was firstly proposed by Sharifzadeh et al. [16]. Two algorithms are focused on static query points, say B^2S^2 and VS^2, and one algorithm is for dynamic query points, say VCS^2. Lots of literature proposed that the distance between objects and surrounding facilities should be considered as a parameter for selecting spatial objects [4], [8], [10], [12]. Based on previous work, You et al. considered that sometimes the retrieved data points should be far away from unpreferable facilities. So they proposed the farthest spatial skyline queries which is an efficient progressive algorithm, say *Branch-and-Bound Farthest Spatial Skyline* (BBFS), for exploiting spatial locality [22]. In [13], Lin et al. considered that the retrieved data objects not only should be close to desirable facilities but also should be far from undesirable facilities by using EFFN algorithm. Different from previous work, Annisa et al. proposed a method, *Unfixed-Shape Areas Skyline* (UASKY), which divided the target area into several disjoint subareas by using Voronoi Diagram. The subareas may be further divided to sub-subareas by other facility types. For each sub-subareas, they calculated the maximum and minimum distance to closest facility for each F^+ and F^- in [1]. In [2], Annisa et al. proposed a superior algorithm to UASKY, *Grid-based Area Skyline* (GASKY), which divided the target area into several grids on average. Then they used Voronoi Diagram to divide the grids, and calculated the maximum and minimum distance for selecting good locations.

MapReduce Based Query Processing

Distributed skyline computation for big data has received more attention in recent years. K. Hose et al. provided a good survey of skyline processing in highly distributed environments [9]. Different from traditional more tightly-coupled parallel platforms, Hadoop MapReduce is becoming more attractive because of their simplicity. MapReduce is a programming model and an associated implementation for processing a massive volume of data. It is widely used for computing skyline in recent years [6], [17], [24].

Zhang et al. designed three MapReduce based BNL (MR-BNL), MapReduce based SFS (MR-SFS) and MapReduce based Bitmap (MR-Bitmap) algorithms, for processing skyline queries [24]. The results outperformed conventional skyline algorithms. Chen et al. applied an angular data partition in the MapReduce-based solution for skyline query evaluation [6]. In [17], Siddique et al. proposed k-dominant skyline query computation in MapReduce Environment. In [20], Wu et al. proposed a novel distributed skyline query processing algorithm (DSL) that discovers skyline points progressively based on the grid structure. Zhang et al. adapted skyline computation to the MapReduce framework, and they recursively divided the dimensions of data into some parts on a grid-based partitioning scheme [23]. However, few papers focus on parallel spatial skyline queries , which motivated us to propose an efficient algorithm for selecting good locations by using MapReduce.

5. Conclusions

An area which is close to preferable facilities and far from undesirable facilities are essential for various applications.

GASKY can help to find such areas which are not dominated by another area. Our proposed MRGASKY is using MapReduce to implement GASKY which has better performance and scalability when increase grid number, facility types and object number.

In addition to motivating example, some concrete examples which could utilize this method are:

- In the business field: suppose a real estate developer would like to find a region to build a community. For attracting customers, the community should be in an area that is close to train stations, convenient stores, school, and should be far from factories and open landfill. The proposed method can help the real estate developer find the potential areas in a map, thus reducing cost of surveying the whole areas.
- In the travel planning: when tourists travel to some unfamiliar places or some countries, they would like to enjoy as many local conditions and customs as possible in the shortest possible time. So our method can recommend some areas which are close to all those preferable local conditions and customs.

This paper addresses a method to improve the performance of area skyline query using grid data structure. The experiments are conducted to demonstrate the effectiveness and efficiency of the algorithms.

In future, we will consider selecting $k-$dominant areas and utilizing some non-spatial properties such as price and population density, as additional parameters in our proposed algorithm.

Acknowledgments

This work is supported by KAKENHI (16K00155,17H01823) Japan. C. Li is supported by KUMAHIRA Scholarship, Japan. A. Zaman was supported by Japanese Government MEXT Scholarship. Annisa was supported by Indonesian Government DG-RSTHE Scholarship.

References

[1] Annisa, M.A. Siddique, A. Zaman and Y. Morimoto: A Method for Selecting Desirable Unfixed Shape Areas from Integrated Geographic Information System, *Proc. IIAI 2015*, pp.195–200 (2015).

[2] Annisa, A. Zaman and Y. Morimoto: Area skyline query for selecting good locations in a map. *J. Inf. Process*, 24(6), 946955 (2016).

[3] Annisa, A. Zaman and Y. Morimoto: Reverse Area Skyline in a Map. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 8(2), 333-343 (2017).

[4] M. Arefin, J. Xu, Z. Chen and Y. Morimoto: Skyline Query for Selecting Spatial Objects by Utilizing Surrounding Objects, *Journal of Software*, Vol. 8, No. 7, pp. 1742–1749 (2013).

[5] S. Borzsonyi, D. Kossmann and K. Stocker: The skyline operator, *Proc. 17th International Conference on Data Engineering (ICDE)*, April 2-6, Heidelberg, Germany, pp.421-430 (2001).

[6] L. Chen, K. Hwang and J. Wu: MapReduce skyline query processing with new angular partitioning approach. In*IPDPS Workshops*, pp. 2262–2270 (2012).

[7] J.Chomicki, P. Godfrey, J. Gryz and D. Liang: Skyline with Presorting, *Proc. 19th International Conference on Data Engineering (ICDE)*, March 5-8, Bangalore, India, pp. 717-719 (2003).

[8] X. Guo, Y. Ishikawa and Y. Gao: Direction-based Spatial Skylines, *Proc. 9th ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*, June 6, Indiana, USA, pp.73–80 (2010).

[9] K. Hose and A. Vlachou: A survey of skyline processing in highly distributed environments, *VLDB*, 2012.

[10] K. Kodama, Y. Iijima, X. Guo and Y. Ishikawa: Skyline Queries Based on User Locations and Perferences for Making Location-based Recommendations, *Proc. 19th International Workshop on Location Based Social Networks (LBSN)*, November 3, Washington, USA, pp.9–16 (2009).

[11] D. Kossmann, F. Ramsak and S. Rost: Shooting stars in the key: An online algorithm for skyline queries, *Proc. 28th International Conference on Very Large Data Bases (VLDB)*, August 20-23, Hong Kong, China, pp 275–286 (2002).

[12] Q. Lin, Y. Zhang, W. Zhang and X. Lin: Efficient general spatial skyline computation, *World Wide Web*, Vol. 16, No. 3, pp. 247–270 (2013)

[13] Y.-W. Lin, E.-T. Wang, C.-F. Chiang and A.L.P. Chen: Finding Targets with the Nearst Favor Neighbor and Farthest Disfavor Neighbor by a Skyline Query, *Proc. 29th Annual ACM Symposium on Applied Computing (SAC)*, March 24–28, Gyeongju, Korea, pp.821–826 (2014).

[14] D. Man, K. Uda, Y. Ito and K. Nakano: Accelerating computation of Euclidean distance Map using the GPU with efficient memory access, *The International Journal of Parallel, Emergent and Distributed Systems*, June 12, pp. 1-23 (2012).

[15] D. Papadias, Y. Tao, G. Fu and B. Seeger: Progressive skyline computation in database systems, *ACM Trans. Database Systems*, Vol.30, No.1, pp.41-82 (2005).

[16] M. Sharifzadeh and C. Shahabi: The Spatial Skyline Queries, *Proc. 32nd International Conference on Very Large Data Bases (VLDB)*, September 12–15, Seoul, Korea, pp.751–762 (2006).

[17] M.A. Siddique, H. Tian and Y. Morimoto: k-dominant skyline query computation in MapReduce environment. IEICE Trans. Inf. Syst. 98, 17451361 (2015).

[18] K.-L. Tan, P.-K. Eng and B.C. Ooi: Efficient Progressive Skyline Computation, *Proc. 27th International Conference on Very Largr Data Bases (VLDB)*, September 11–14, Rome, Italy, pp.301–310 (2001).

[19] T. Hayashi, K. Nakano and S. Olariu: Optimal parallel algorithm for finding proximate points, with applications. IEEE Transactions on Parallel and Distributed System 9, pp.1153-1166 (1998).

[20] P. Wu, C. Zhang, Y. Feng, B. Zhao, D. Agrawal and A. Abbadi: Parallelizing skyline queries for scalable distribution, *EDBT*, 2006.

[21] T. Xia, D. Zhang and Y. Tao: ON Skylining with Flexible Dominance Relation, *Proc. 24th International Conference on Data Engineering (ICDE)*, April 7-12, Cancun, Mexico, pp. 1397–1399 (2008).

[22] G.-W. You, M.-W. Lee, H. Im and S.-W. Hwang: The Farthest Spatial Skyline Queries, *Information Systems*, Vol. 38, No. 3, pp.286–301 (2013).

[23] B. Zhang, S. Zhou and J. Guan: Adapting skyline computation to the MapReduce framework: Algorithms and experiments, *EDBT*, 2011.

[24] C. Zhang, S. Zhou and J. Guan: Adapting skyline computation to the MapReduce framework: Algorithms and experiments. In *DASFAA Workshops*, pp. 403–414 (2011).