JOIN (Jurnal Online Informatika)
Volume 6 No.1 | June 2021: 25-32
DOI: 10.15575/join.v6i1.667
p-ISSN: 2528-1682
e-ISSN: 2527-9165

# Location Selection Query in Google Maps using Voronoi-based Spatial Skyline ($VS^2$) Algorithm

**Annisa Annisa[1], Leni Angraeni[2]**
[1,2]Department of Computer Science, IPB University, Indonesia

**Article Info**

***Article history:***

Received December 01, 2020
Revised April 22, 2021
Accepted May 03, 2021
Published June 17, 2021

***Keywords:***

Google Maps
Location selection
Skyline query
Spatial skyline query
Voronoi-based spatial skyline algorithm

**ABSTRACT**

Google Maps is one of the popular location selection systems. One of the popular features of Google Maps is nearby search. For example, someone who wants to find the closest restaurants to his location can use the nearby search feature. This feature only considers one specific location in providing the desired place choice. In a real-world situation, there may be a need to consider more than one location in selecting the desired place. Assume someone would like to choose a hotel close to the conference hall, the museum, beach, and souvenir store. In this situation, nearby search feature in Google Maps may not be able to suggest a list of hotels that are interesting for him based on the distance from each destination places. In this paper, we have successfully developed a web-based application of Google Maps search using Voronoi-based Spatial Skyline ($VS^2$) algorithm to choose some Point Of Interest (POI) from Google Maps as their considered locations to select desired place. We used Google Maps API to provide POI information for our web-based application. The experiment result showed that the execution time increases while the number of considered location increases.

***Corresponding Author:***

Annisa Annisa,
Department of Computer Science,
IPB University,
Jl Meranti Wing 20 Level 5 Kampus IPB Darmaga 16680
Email: annisa@apps.ipb.ac.id

## 1. INTRODUCTION

Google Maps is one of the popular location selection systems. Google Maps is used to search and select locations with several features that it has. One of the popular features of Google Maps is nearby search. For example, someone who wants to find the closest restaurants to his location can use the nearby search feature. This feature only considers one specific location in providing the desired place. In a real-world situation, there may be a need to consider more than one location in selecting the desired place.

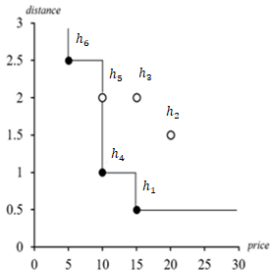
Assume there are employees from different branch offices of a company that want to hold a meeting in a restaurant. In choosing a restaurant, the meeting committee should consider traveling distances from the restaurant to all of the team members. In another example, a professor wants to participate in a conference outside the city. He would like to choose a hotel so that in addition to attending the conference he also can visit some places in the city like museum and souvenir stores. To choose the hotel, the professor should consider the location of the conference venue, museums, and souvenir shops. This type of query is significant in a real-world situation. Unfortunately, popular location selection platform like Google Maps does not support this type of query, thus the benefit of this query cannot be experienced by the wider community.

Spatial Skyline Query (SSQ) is a location selection method that considers several other locations. Skyline Query is the basic method of SSQ. Borzsonyi et al. [1] first introduced the concept of skyline query. Skyline query is a method of selecting interesting objects which are not dominated by other objects in the dataset. An object dominates other objects if it is as good or better in all dimensions and better in at least one dimension. Figure. 1 shows an example of a general skyline query. The table in Figure. 1 (a) shows a list of hotels with attribute price and distance to the beach. We assume that a smaller value is better in each attribute. Figure. 1 (b) shows that hotel $h_2$ and hotel $h_3$ are dominated by $h_1$, and hotel $h_5$ is dominated by $h_4$. Hotel $h_1$, $h_4$, and $h_6$ are skyline objects because $h_1$, $h_4$, and $h_6$ are not dominated by each other. Skyline query is applied in multi-criteria decision-making without using any specific objective function to determine the best results[2]. Many skyline algorithms introduced in [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11].

p-ISSN: 2528-1682
e-ISSN: 2527-9165

| Hotel | Price | Distance |
|---|---|---|
| $h_1$ | 15 | 0.5 |
| $h_2$ | 20 | 1.5 |
| $h_3$ | 15 | 2 |
| $h_4$ | 10 | 1 |
| $h_5$ | 10 | 2 |
| $h_6$ | 5 | 2.5 |

(a)




(b)

Figure 1. General Skyline Query

In the case of interesting spatial object selection like location, Sharifzadeh et al.[12] introduced the concept of skyline query using geometric elements in their solution, namely Spatial Skyline Query (SSQ). Spatial Skyline Query (SSQ) is a method for skyline query problems with spatial parameters. SSQ is used to find skyline objects in a dataset based on the given query points. Several studies have used the SSQ concept in [12], [13], [14], [15], [16].

In reference [12], let the set $P$ contain points in the $d$-dimensional space $R^d$ and $D(..,..)$ be a distance metric defined in $R^d$ where $D(..,..)$ obeys the triangular inequality. Given a set of $d$-dimensional query points $Q=\{q_1,\ldots,q_n\}$ and the two object points $p$ and $p'$ in $R^d$, $p$ spatially dominates $p'$ with respect to $Q$ *iff* we have $D(p,q_i) \leq D(p',q_i)$ for all $q_i \in Q$ and $D(p,q_j) < D(p',q_j)$ for some $q_j \in Q$. Figure. 2 shows an example of object points and query points in SSQ problem.

| Hotel | X | Y |
|---|---|---|
| $p_1$ | 7 | 5 |
| $p_2$ | 7 | 7 |
| $p_3$ | 5 | 1 |
| $p_4$ | 5 | 6 |
| $p_5$ | 5 | 3 |
| $p_6$ | 9 | 3 |

(a) Object Points $P$

| *Query* | X | Y |
|---|---|---|
| $q_1$ | 4 | 6 |
| $q_2$ | 5 | 8 |
| $q_3$ | 8 | 4 |
| $q_4$ | 2 | 3 |
| $q_5$ | 4 | 4 |

(b) Query Points $Q$

Figure 2. Example of object points and query points in SSQ problem

Given the hotel dataset $P=\{p_1, p_2, p_3, p_4, p_5, p_6\}$ and query points $Q=\{q_1, q_2, q_3, q_4, q_5\}$ as locations to be considered in the selection of P, such as conference venues, airports, souvenirs shop, restaurant, or night markets. $X$ and $Y$ are the coordinates of $P$ and $Q$. Figure. 2 (a) shows a dataset of hotel $P$, and Figure. 2 (b) shows a dataset of query points $Q$. The spatial skyline of $P$ with respect to $Q$ is the set of those points in $P$ ($p$) which are not spatially dominated by any other point of $P$ ($p'$). The points $p$ is in the spatial skyline *iff* we have [12]:

$$\forall p' \in P, p' \neq p, \exists q_i \in Q \; s.t. \; D(p, q_i) \leqslant D(p', q_i) \quad (1)$$

Equation 1 shows that finding the spatial skyline object can be done by calculating the distance $p \in P$ to all $q \in Q$, then compare it with other $p$ ($p'$). If no $p'$ dominates $p$, then $p$ becomes the skyline object. The distance from one point to another is calculated using Euclidean Distance, with the results shown in Table 1.

Table 1. The result of distance calculation

| Hotel | q1 | q2 | q3 | q4 | q5 |
|---|---|---|---|---|---|
| p1 | 3.16 | 3.60 | 1.41 | 5.38 | 3.16 |
| p2 | 3.16 | 2.24 | 3.16 | 6.40 | 4.24 |
| p3 | 5.10 | 7.00 | 4.24 | 3.60 | 3.16 |
| p4 | 1.00 | 2.00 | 3.60 | 4.24 | 2.24 |
| p5 | 3.16 | 5.00 | 3.16 | 3.00 | 1.41 |
| p6 | 5.83 | 6.40 | 1.41 | 7.00 | 5.10 |

Table 1 shows that hotel $p_6$ is dominated by $p_1$ and $p_3$ is dominated by $p_5$. Hotels $p_1$, $p_2$, $p_4$, and $p_5$ are not dominated by each other, so those hotels are skyline objects. However, this method is inefficient because it requires high computational time to calculate distances from $P$ to all $Q$ and to compare all $P$ points. One efficient SSQ algorithm is Voronoi-based Spatial Skyline ($VS^2$) [12]. $VS^2$ uses three geometric structures to get the spatial skyline solution, namely the *Voronoi diagram*, *Delaunay Graph*, and *Convex Hull*. Currently, SSQ has not been implemented on Google Maps. This research implemented a web-based location selection system using $VS^2$ for location selection on Google Maps so that it can be used widely and easily by Google Maps users. Moreover, the web-based application also considers the non-spatial attribute of the selecting location, such as rating, which had not been considered previously in the original algorithm.

The paper is organized with the following structure. Section I introduced the background and contribution of this work. Section II briefly explained about $VS^2$ algorithm. We described our methodology in Section III. The result of our work is presented in Section IV. Finally, we concluded our work in Section V.

## 2. VORONOI-BASED SPATIAL SKYLINE ($VS^2$) ALGORITHM

To improve the efficiency of the SSQ algorithm, Sharifzadeh et al. [12] utilized three geometric structures: Voronoi diagram, Delaunay graphs, and Convex hull. One of the proposed algorithms is Voronoi-based Spatial Skyline ($VS^2$). Voronoi diagram provides an efficient data structure to compute the nearest Voronoi point for a given query point *q* [17]. Zhu et al. [18], Safar et al. [19], Agarwal et al. [20], and Arefin et al. [17], used Voronoi diagrams for location selection.

In reference [12], $VS^2$ reduces the amount of checking for dataset *P* (based on Lemma 1, Theorem 1, and Theorem 3) and calculates the distance to *Q* (based on Theorem 2).

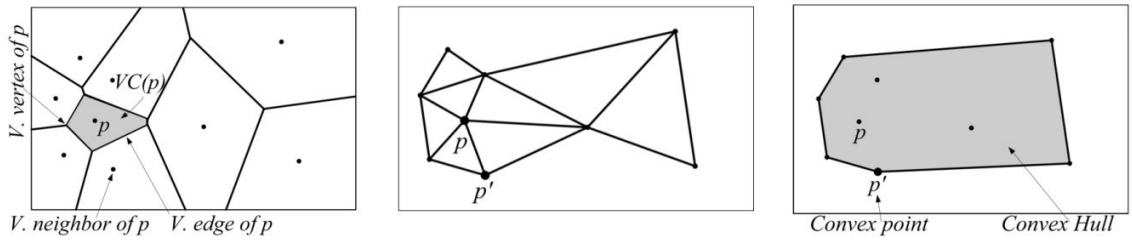
Lemma 1. *For each qi ∈ Q, the closest point to qi in P is a skyline point*.

Theorem 1. *Any point p ∈ P, which is inside the convex hull of Q, is a skyline point*.

Theorem 2. *The set of skyline points of P does not depend on any non-convex query point q ∈ Q*.

Theorem 3. *Any point p ∈ P whose Voronoi cell V C(p) intersects with the boundaries of the convex hull of Q is a skyline point*.

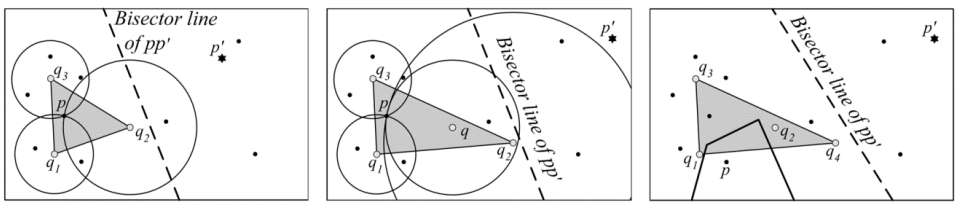
Figure. 3 (a) shows the Voronoi diagram of a set of nine points *P*. The point *p* is a Voronoi point which is the object point that constructs the Voronoi diagram. The grey area enclosing point *p* is Voronoi cell of *p* or *VC*(*p*), which is a convex polygon with the edge is a line that divides two Voronoi points with the same distance to the line. For every two points, *p* and *p'* with the same edge in its Voronoi cell means that p is Voronoi neighbor of *p'*. Figure. 3(b) shows that there is an edge connecting *p* and *p'* in undirected graph *G* iff *p'* is Voronoi neighbor of *p* in the Voronoi diagram of *P*. The graph *G* is called the Delaunay Graph of points in *P*. Figure. 3 (c) shows the Convex hull of the points *P* or *CH*(*P*) which is the smallest convex polygon that contains all the points in *P*, and the vertices of this polygon or $CH_v(P)$ called convex point.




(a) (b) (c)

Figure 3. Three geometric structures in Sharifzadeh et al. [12] (a) Voronoi Diagram, (b) Delaunay Triangulation, (c) Convex Hull

Given two sets of points which are object points $P = \{p_1, p_2, \ldots, p_n\}$ and query points $Q = \{q_1, q_2, \ldots, q_n\}$, figure. 4 (a), (b), and (c) shows that the bisector line of *pp'* is a perpendicular line for each two points *p* and *p'* where the distance from the point *p* and point *p'* to the bisector line are equal. This line divides the two regions for each point *p* and *p'* where all points in the same region as point *p* will be closer to the point *p* compared to the point *p*' and vice versa. The circle is constructed with the center at *qi* and the radius *D*(*qi*,*p*) or called *C*(*qi*,*p*), which means that $q_i$ is closer to any point inside *C*(*qi*,*p*) than to *p*. Figure. 4 (a) shows that *p* is a skyline point because it is inside the convex hull of *Q*, so it will not be compared with other object points. Figure. 4 (b) shows that the *C*(*q*,*p*) entirely inside the union of the circles $C(q_2,p)$ for $qi \in CHv(Q)$ [12], so the non-convex point *q* have no impact on the result of the spatial skyline. Thus, it can reduce the time required to do the distance computation because we just need to compute the distance from object points to convex points, and we can ignore the non-convex point of *CH*(*Q*). Figure. 4 (c) shows that the point *VC*(*p*), which intersects with the boundaries *CH*(*Q*), is the skyline point, so the point *p* will not be compared with other object points.




(a) (b) (c)

Figure 4. Theorems in Sharifzadeh et al. [12] (a) Theorem 1, (b) Theorem 2, and (c) Theorem 3

p-ISSN: 2528-1682
e-ISSN: 2527-9165

## 3. METHOD

The methodology of this research consists of: creating a data collection module, creating an SSQ module using VS$^2$, system development, system testing, and experiments.

### 3.1. Data Collection from Google Maps

The data processed by every run of VS$^2$ is dynamic because the data is based on user preferences. The data consists of Point of Interest (POI) in Google Maps with spatial and non-spatial information. This data was obtained by utilizing Google Maps API. The user determines the type of POI targeted for search, such as restaurants, hotels, etc. After that, a data request is made to obtain spatial information and non-spatial information. The spatial information are latitude and longitude of POI, and the non-spatial information is POI rating according to the type of POI entered. This is the basic concept of the data collection module.

There are two Google Maps APIs used in this module. First, Geocoding API is used to get location information (latitude, longitude) from user queries. The required parameter is the name of the query location. An example of the data collection result can be seen in Table 2. In this example, there are five query locations ($Q$), namely SMAN 48 Jakarta, Lubang Buaya Jakarta, Taman Mini Indonesia Indah, SPBU Pinang Ranti 2, and Yayasan Bamadita Rahman.

Table 2. The result of query data collection

| Name of query | Latitude | Longitude |
|---|---|---|
| SMAN 48 Jakarta | -6.287359 | 106.882821 |
| Lubang Buaya Jakarta | -6.293907 | 106.903398 |
| Taman Mini Indonesia Indah | -6.302446 | 106.895156 |
| SPBU Pinang Ranti 2 | -6.291301 | 106.888318 |
| Yayasan Bamadita Rahman | -6.293165 | 106.893153 |

Second, Places API Nearby Search is used to get non-spatial information based on the type of POI. The information includes name, rating, and location (latitude, longitude). The parameters used are the type of POI (restaurants, hotels, supermarkets, etc.) and the search location points (latitude and longitude) obtained by taking the average value of the location of the query. For example, for query data with five places in Table 2, a search location point is obtained at location latitude -6.2936352 and longitude 106.892569 shown in Figure 5 with the X sign. After getting the location search point, a restaurant data request is made using a nearby search with a radius of 5000 meters from that location point. The total data obtained from the request for one type of POI is 60 data. Examples of the POI processing results data can be seen in Table 3.

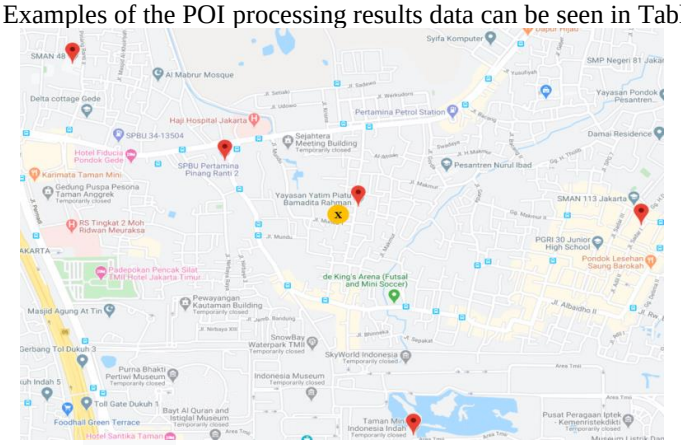



Figure 5. Example of location points on Google Maps

Table 3. The result of POI data collection

| Name of POI | Rating | Latitude | Longitude |
|---|---|---|---|
| Masakan Wong Solo | 5.0 | -6.303482 | 106.901074 |
| Pecel Madiun | 4.4 | -6.303058 | 106.901850 |
| Sambara | 3.9 | -6.304384 | 106.887348 |
| H.E.M.A Resto | 4.2 | -6.303929 | 106.886587 |
| Ayam Bakar Riri | 4.8 | -6.290319 | 106.882738 |

### 3.2. SSQ Implementation

The Spatial Skyline Query (SSQ) module is created according to the pseudocode shown in Figure 6. The pseudocode starts from computing convex hull ($CH$) of query points ($Q$), and initialization for the data structure that will be used include the results of skyline $S(Q)$, minimum heap tree $H$, *Visited*, and *Extracted*. Initialization is done by randomly selecting one query point ($q_1$) and determining the point $p \in P$ with the closest distance to the query point $q1$ (*Nearest Neighbor* $q_1$). Iteration begins by checking the first point in $H$

(*p*). When checking is complete, continue searching and checking *Voronoi neighbors of p*. If *H* is empty, then the iteration stops, and VS$^2$ returns the *S*(*Q*) value as the skyline.

We modified the pseudocode implementation by adding conditions for filtering POI data according to the minimum rating from the user, which is shown in Figure 7. Input data for this module is POI and query data obtained from the data collection module and minimum rating information from user input. Information used from POI data and queries is the location (latitude and longitude). The distance calculation used is Euclidean Distance, according to the distance calculation used in the VS$^2$ algorithm. The results of this module are skyline objects of POI.

```
01.  compute the convex hull CH(Q);
02.  set S(Q) = {};
03.  Heap H = {(NN(q1), mindist(NN(q1), CHv(Q)))};
04.  set Visited = {NN(q1)}; set Extracted = {};
05.  box B = MBR(SR(NN(q1), Q));
06.  while H is not empty
07.     (p, key) = first entry of H;
08.     if p ∈ Extracted
09.        remove (p, key) from H;
10.        if p is inside CH(Q) or p is not dominated by S(Q)
11.           add p to S(Q);
12.     else
13.        add p to Extracted;
14.        if S(Q) = Ø or a Voronoi neighbor of p is in S(Q)
15.           for each Voronoi neighbor of p such as p'
16.              if p' ∈ Visited, discard p';
17.              if p' is inside B or VC(p') intersect with B
18.                 add p' to Visited;
19.                 add (p', mindist(p', CHv(Q))) to H;
20.                 B = B ∩ MBR(SR(p', Q));
21.  return S(Q)
```

Figure 6. Pseudocode of VS$^2$ [12]

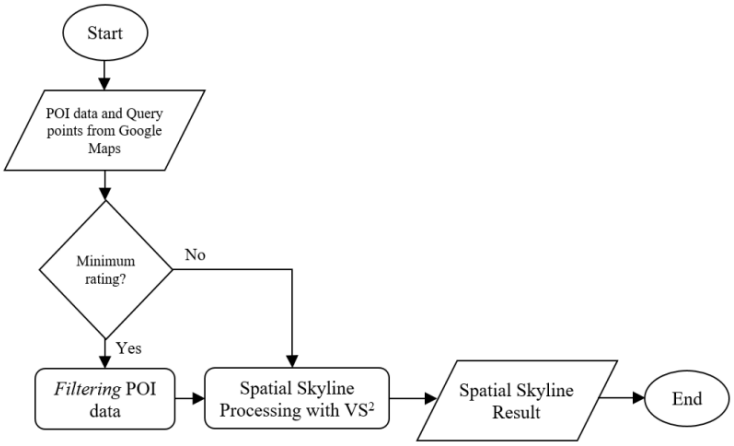



Figure 7. Modification of VS$^2$ implementation in Google Maps

### 3.3. System Development

The design and development of web-based location selection systems are carried out in this stage. The design phase consists of making activity diagrams and system interface designs. After that the system is developed, referring to the design that has been made. Google Maps API that is used in the system are Places API and Maps JavaScript API. Figure. 8 shows the activity diagram of the system.

p-ISSN: 2528-1682
e-ISSN: 2527-9165

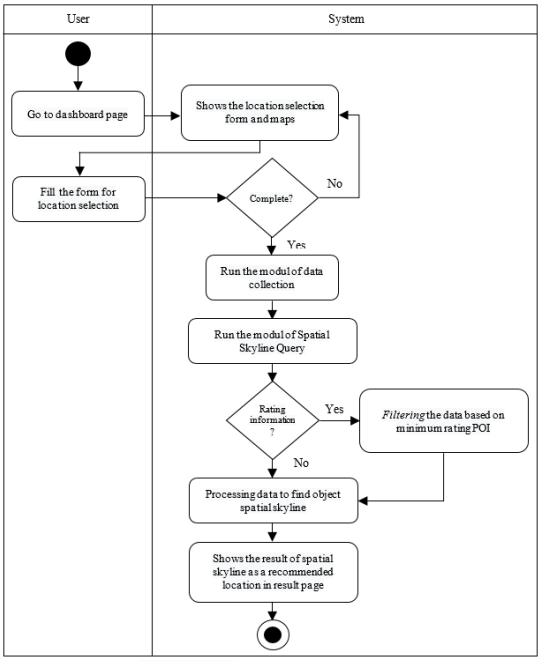



Figure 8. Activity Diagram

### 3.4. System Testing

After system development has been completed, we conducted testing. The testing is performed by doing an interaction simulation with the system, from filling out the location search input form to obtaining a set of skyline locations.

### 3.5. Experiment

In the experiment, we observed the performance of the SSQ query in Google Maps and recorded the time required to complete the search. The experiment also observed the effect of a different numbers of queries on the system performance.

## 4. RESULTS AND DISCUSSION

A web-based location selection system is designed so that VS$^2$ algorithm implementation can be used directly by the user. There are two main pages in this system, the input form for location search and the spatial skyline results page, which displays recommended location for users. Figure. 9 shows the input page for location search based on the preferences of the user. The input form includes three input fields. First is the type of POI (restaurant, supermarket, hotel, etc.) that would be searched. The second field is selecting the minimum rating of the POI, which is optional. Third are query (considered) locations from the user, between 2 to 5 locations maximum. Then the user clicks on submit button, and input data from the user will be processed to find the final spatial skyline. Figure. 10 shows an example of a result page that recommends POI for the user. There are three colors on the result page, blue is query (considered) locations from the user, red is all POI data collected from Google Maps based on query locations, and green denotes the POIs that are the final spatial skyline.

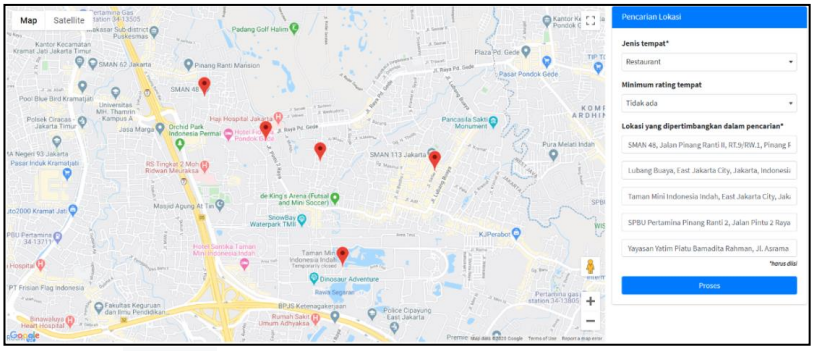



Figure 9. Dashboard page for SSQ location selection query

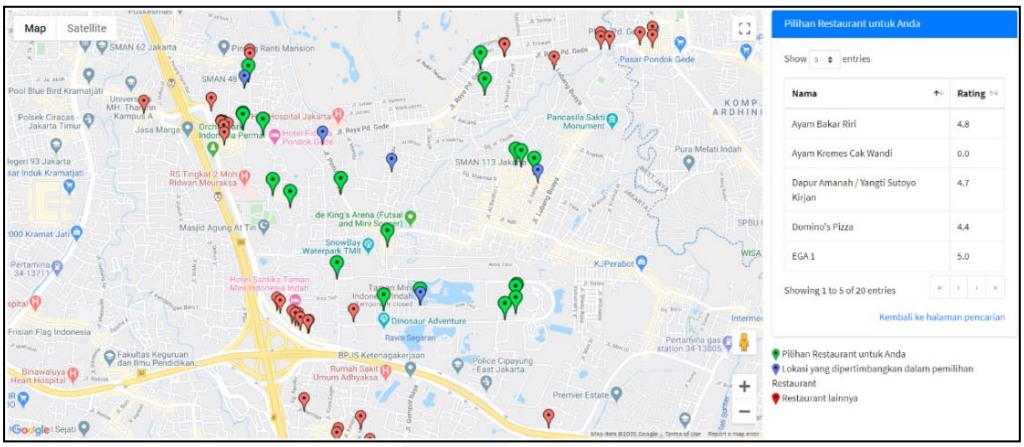



Figure 10. Result page

Next, the system is tested using the *black-box* method to determine whether all the functions that have been developed are working as expected. Table 4 shows the result of testing the three functions of the system, the form filling, autocomplete location fields on the form, and markers on the map when filling in the location fields.

Table 4. The result of system testing

| Function | Scenario | Expected Result | Result |
|---|---|---|---|
| Fill out the form for location search. | Complete the input data on form and click the "Proses" button. | Redirect to the result page and show the spatial skyline points as a recommended location | Success |
| Autocomplete location field | Typing the location for query | Show the recommended location based on typed word | Success |
| Push marker when filling out the location of a query | Choose one location from the recommendation of *autocomplete* | Add marker at the location on the maps | Success |

Execution time for the VS$^2$ algorithm can be seen in Figure.11. The total POI data used to calculate the execution time is fixed at 60 restaurants with varying the number of queries from 2, 4, 6, 8, to 10. Figure.11 (a) shows the execution time for the VS$^2$ algorithm to find the spatial skyline object on existing data. Figure 11 (b) shows execution time from collecting data until finding spatial skyline object. Based on the execution time, we can see that as the number of queries increases, the processing time required to find spatial skyline objects also increases. There is an additional execution time required for data collection and filtering before running the VS$^2$ algorithm. The time complexity of the VS$^2$ algorithm is $O(|S(Q)|^2|CH_v(Q)| + \Phi(|P|))$ where $|S(Q)|$ is the number of solution, $|CH_v(Q)|$ is the number of vertex convex hull of $Q$ ($|CH_v(Q)| \leq |Q|$), and $\Phi(|P|)$ is the complexity of finding the data point from which VS$^2$ starts visiting inside $CH(Q)$ (e.g., $NN(q_i)$) [12].

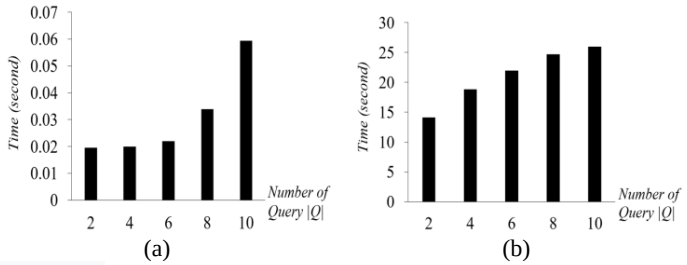



Figure 11. Execution time (a) only SSQ module, and (b) Collecting data + SSQ module

## 5. CONCLUSION

This research has been successfully implemented the Voronoi-based Spatial Skyline (VS$^2$) algorithm for location selection based on spatial criteria on Google Maps. The data used is the Point of Interest (POI) dataset from Google Maps obtained by utilizing the Google Maps API implemented in the data collection module. The execution time is affected by the amount of object data and the number of queries. In the future, we would like to develop this web application into a mobile application so that it is more accessible for many users. Implementation of a newer skyline algorithm to improve execution time can also be researched. In further research, the data collection algorithm can be modified by other methods that can perform faster. In addition to that, the VS$^2$ algorithm that currently still uses Euclidean distance can be researched further to using road map distances that correspond to the real-world situations.

## 6. REFERENCES

[1] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings - International Conference on Data Engineering*, 2001, pp. 421–430.
[2] C. Kalyvas and T. Tzouramanis, "A Survey of Skyline Query Processing," Apr. 2017.
[3] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, "Skyline query processing for incomplete data," in *Proceedings - International Conference on Data Engineering*, Apr. 2008, pp. 556–565.
[4] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proceedings - International Conference on Data Engineering*, 2003, pp. 717–719.
[5] I. Bartolini, P. Ciaccia, and M. Patella, "SaLSa: Computing the skyline without scanning the whole sky," in *International Conference on Information and Knowledge Management, Proceedings*, 2006, pp. 405–414.
[6] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03*, 2003, p. 467.
[7] A. L. Ramdani, T. Djatna, and H. Sukoco, "Selecting user influence on twitter data using skyline query under MapReduce framework," *Telkomnika (Telecommunication Comput. Electron. Control.*, vol. 16, no. 3, pp. 1416–1425, 2018.
[8] A. Zaman, Md. Anisuzzaman Siddique, Annisa, and Y. Morimoto, "Finding Key Persons on Social Media by Using MapReduce Skyline," *Int. J. Netw. Comput.*, vol. 7, no. 1, pp. 86–104, 2017
[9] Annisa, M. A. Siddique, A. Zaman, and Y. Morimoto, "A Method for Selecting Desirable Unfixed Shape Areas from Integrated Geographic Information System," in *Proceedings - 2015 IIAI 4th International Congress on Advanced Applied Informatics, IIAI-AAI 2015*, Jul. 2016, pp. 195–200.
[10] X. Lin, J. Xu, and H. Hu, "Range-based skyline queries in mobile environments," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 835–849, Apr. 2013.
[11] Y. W. Lin, E. T. Wang, C. F. Chiang, and A. L. P. Chen, "Finding targets with the nearest favor neighbor and farthest disfavor neighbor by a skyline query," in *Proceedings of the ACM Symposium on Applied Computing*, Mar. 2014, pp. 821–826.
[12] M. Sharifzadeh and C. Shahabi, "The spatial skyline queries," in *VLDB '06 Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 751–762.
[13] K. Kodama, Y. Iijima, X. Guo, and Y. Ishikawa, "Skyline queries based on user locations and preferences for making location-based recommendations," in *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, 2009, pp. 9–16.
[14] X. Guo, Y. Ishikawa, and Y. Gao, "Direction-based spatial skylines," in *MobiDE 2010 - Proceedings of the 9th ACM International Workshop on Data Engineering for Wireless and Mobile Access, in Conjunction with ACM SIGMOD / PODS 2010*, 2010, pp. 73–80.
[15] Z. Chen, M. S. Arefin, and Y. Morimoto, "Skyline queries for spatial objects: A method for selecting spatial objects based on surrounding environments," in *Proceedings of the 2012 3rd International Conference on Networking and Computing, ICNC 2012*, 2012, pp. 215–220.
[16] Annisa, A. Zaman, and Y. Morimoto, "Area skyline query for selecting good locations in a map," *J. Inf. Process.*, vol. 24, no. 6, pp. 946–955, 2016.
[17] M. S. Arefin, G. Ma, and Y. Morimoto, "A Spatial Skyline Query for a Group of Users," *J. Softw.*, vol. 9, no. 11, pp. 137–142, Nov. 2014.
[18] L. Zhu, Y. Jing, W. Sun, D. Mao, and P. Liu, "Voronoi-based aggregate nearest neighbor query processing in road networks," in *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, 2010, pp. 518–521.
[19] M. Safar, D. Ibrahimi, and D. Taniar, "Voronoi-based reverse nearest neighbor query processing on spatial networks," in *Multimedia Systems*, 2009, vol. 15, no. 5, pp. 295–308.
[20] R. Agarwal and D. Garg, "Finding nearest facility for multiple customers using voronoi diagram," in *Souvenir of the 2014 IEEE International Advance Computing Conference, IACC 2014*, 2014, pp. 641–646.