Journal of information and communication convergence engineering
http://jicce.org

J. Inf. Commun. Converg. Eng. 17(2): 142-148, Jun. 2019 Regular paper

Android Application for Connecting Cycling Routes on Strava Segments

Intiraporn Mulasastra* and **Wichpong Kao-ian**

Department of Computer Engineering, Kasetsart University, Bangkok 10900, Thailand

Abstract

Relatively few countries provide separate bicycle lanes for cyclists. Hence, tools for suggesting cycling routes are essential for a safe and pleasant cycling experience. This study aims to develop a mobile application to build cycling routes based on user preferences, specifically location, search radius, ride distance, and number of optimal routes. Our application calls the Strava API to retrieve Strava cycling segments crowdsourced from the cycling community. Then, it creates a graph consisting of the start and end points of these segments. Beginning from a user-specified location, the depth-first search algorithm (DFS) is applied to find routes that conform to the user's preferences. Next, a set of optimal routes is obtained by computing a trade-off ratio for every discovered route. This ratio is calculated from the lengths of all segments and the lengths of all connecting paths. The connected routes can be displayed on a map on an Android device or exported as a GPX file to a bike computer. Future work must be performed to improve the design of the user interface and user experience.

Index Terms: Android application, Bike navigator, Depth-First Search, GPX, Strava segments

I. INTRODUCTION

A vibrant cycling culture is emerging in many countries, including Thailand, for both general transportation and recreation. However, according to research on cycling safety [1], there is a lack of adequate bicycling infrastructure due to limited public funding. However, fast mobile broadband connections and inexpensive rates have made access to mobile mapping applications affordable [2]. Hence, a cycling trip planner on mobile phones can alleviate the problem by recommending optimal routes for cyclists.

Google Maps APIs have been used to implement web-based mapping services in a wide range of applications [3]. However, information on cycling routes provided by Google Maps is only available in a handful countries and areas [4]. Unlike Google Maps, the Strava API provides cycling route information crowdsourced from cyclists in many countries and areas. Strava users can record and track cycling activities via the Global Positioning System (GPS) [5, 6], then share their datasets and vote on their favorite routes. These crowdsourced routes are typically safer than alternative routes [1].

However, the routes recommended by Strava are not always satisfactory to the cycling community due to being short or disconnected, especially in areas where there are only a few suggested routes. This study presents the development of a mobile application for creating optimal cycling routes based on user preferences, specifically location, search radius, ride distance, and number of routes. Applying the depth-first search (DFS) algorithm, our application will select riding segments from Strava and connect them to create a set of possible routes according to user preferences.

To find a set of optimal routes, we calculate a trade-off ratio for all of the discovered routes based on the lengths of

Received 01 November 2018, **Revised** 11 April 2019, **Accepted** 20 April 2019
***Corresponding Author** Intiraporn Mulasastra (E-mail: int@ku.ac.thm, Tel: +66-89-1205088)
Department of Computer Engineering, Kasetsart University, Bangkok 10900, Thailand

Open Access https://doi.org/10.6109/jicce.2019.17.2.142 print ISSN: 2234-8255 online ISSN: 2234-8883

all segments and the lengths of all connecting paths. The results can be displayed on a map on any Android device, and can also be exported as a GPX file to a bike computer. We discuss related work, the proposed system, and present the discussion and conclusions in the following sections.

II. RELATED WORK

This section describes related work that we use to implement our system. First, we explain the GPS exchange format. Then, we introduce the method for requesting GPS data from a server as well as the DFS algorithm.

A. GPS Exchange Format

GPS data consist of waypoints, tracks, and routes. A GPX file is an XML data format for the interchange of GPS data between applications and Web services on the Internet [7-9]. GPX files contain a description of their contents, allowing anyone to create a program that can read the data within [8]. Fig. 1 shows sample descriptions of GPS data, and explanations are shown in Table 1.

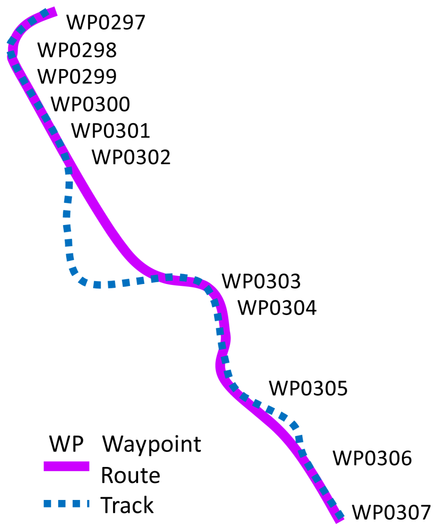


Fig. 1. GPS Exchange Format [6].

Table 1. GPS data

GPS Data	Description
Waypoint	An individual waypoint consists of the *GPS coordinates* of a point and other information (e.g., timestamp).
Track	An ordered list of points describing a path. Tracks are a record of where a person has been.
Route	An ordered list of route points leading to a destination. A route is a suggestion for where to go in the future.

Source: Wikipedia [9].

B. Requesting GPS Data from a Server

1) HTTP Request

An HTTP request message from a client to a server includes the method to be applied to the resource, the identifier of the resource, and the protocol version in use [10]. The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ends with Carriage Return and Line Feed (CRLF). The Method token indicates the method to be applied to the resource identified by the Request-URI, e.g., GET, POST, and CONNECT.

2) JavaScript Object Notation (JSON)

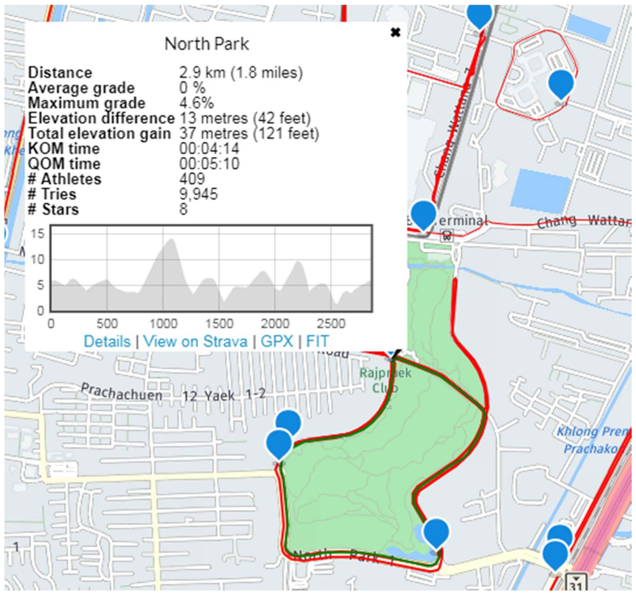
JSON is text used for exchanging data between a browser and a server. Any JavaScript object can be converted into JSON and sent to a server. In addition, any JSON received from a server can be converted into JavaScript objects.

JavaScript has a built-in function, JSON.parse(), to convert a string, written in JSON format, into native JavaScript objects. We can convert a JavaScript object into a string with JSON.stringify() [11].

3) Strava Explore Segments

Strava provides cycling route segments created by Strava cyclists who have previously traveled along those routes. Fig. 2 shows an example of Strava route segments [12].

Strava provides APIs for retrieving and exporting route segments as GPX files. An "exploreSegments" command returns the top 10 segments matching a specified query [13]. A sample command is shown below, and a sample of a command response is shown in Fig. 3.


North Park

Distance	2.9 km (1.8 miles)
Average grade	0 %
Maximum grade	4.6%
Elevation difference	13 metres (42 feet)
Total elevation gain	37 metres (121 feet)
KOM time	00:04:14
QOM time	00:05:10
# Athletes	409
# Tries	9,945
# Stars	8

Fig. 2. Strava route segments [12].

Sample response:

```
 {"segments" : [ {
"id" : 229781,
"resource_state" : 2,
"name" : "Hawk Hill",
"climb_category" : 1,
"climb_category_desc" : "4",
"avg_grade" : 5.7,
"start_latlng" : [ 37.8331119, -122.4834356 ],
"end_latlng" : [ 37.8280722, -122.4981393 ],
"elev_difference" : 152.8,
"distance" : 2684.8,
"points": // encoded polyline of the segment
     "}g|eFnpqjVl@En@Md@HbAd@d@^h@Xx@VbARjBDh@OPQf@w
     @d@k@XKXDFPH\\EbGT`AV`@v@|@NTNb@?XOb@cAxAWLuE
     @eAFMBoAv@eBt@q@b@}@tAeAt@i@dAC`AFZj@dB?~@[h@M
     bAVn@b@b@\\d@Eh@Qb@_@d@eB|@c@h@WfBK|AMpA?VF\\\\
     t@f@t@h@j@|@b@hCb@b@XTd@Bl@GtA?jAL`ALp@Tr@RXd@
     Rx@Pn@^Zh@Tx@Zf@`@FTCzDy@f@Yx@m@n@Op@VJr@"
"starred" : false }]
}
```

Fig. 3. Sample response to an exploreSegments command [13].

apiInstance.exploreSegments (bounds, activityType, minCat, maxCat)

The four parameters are defined as:

bounds = [SW corner latitude, SW corner longitude, NE corner latitude, NE corner longitude],
activityType = running or riding,
minCat = the minimum climbing category, and
maxcat = the maximum climbing category.

4) Encoded Polyline Algorithm

Google Maps applications define the concept of a polyline as a list of points, where line segments are drawn between consecutive points [14]. Polyline encoding is a compression algorithm for converting a series of coordinates (latitude and longitude points) into a single string for data interchange in text format [15]. This algorithm is particularly useful when there are many waypoints because the URL is significantly shorter when using an encoded polyline. This is beneficial because URLs are limited to 8192 characters for all web services [16]. Strava uses the Google encoded polyline algorithm format to encode the latitude and longitude coordinates of points belonging to segments returned by Strava API requests.

C. Depth-First Search Algorithm

In a graph, DFS "explores edges out of the most recently discovered vertex that still has unexplored edges leaving it" [17]. After exploring all of the edges, it backtracks to explore edges radiating from the vertices from which it originated [17]. As shown in Fig. 4, a recursive DFS algorithm explores a graph G by starting from a point v by processing v and then recursively traveling to all adjacent points [18].

```
DFS(G,v):
  label v as discovered
  for all edges from v to w in G.adjacentEdges(v) do
    if vertex w is not labeled as discovered then
      recursively call DFS(G,w)
```

Fig. 4. Depth-first search algorithm [18].

III. Our Proposed System

We develop a mobile application for creating cycling routes based on user preferences, namely location, search radius, number of optimal routes, and riding distance. Our application is an Android application written in Java using Android Studio. We use an Android smartphone to test our program and a bike computer (Garmin Edge 510) to test the exporting of data.

The Strava API is called to find the most popular riding segments in the location chosen by the user, which may be unconnected. Our application connects these segments to create suitable routes by applying the DFS algorithm, as explained in Section II, and a trade-off ratio. A suitable cycling route recommended by our system in GPX format, which can be shown on a map on any Android device.

This section presents our overall system (Fig. 5), which consists of three major tasks: acquiring route segments from Strava datasets, connecting segments and finding a set of optimal routes, and exporting cycling routes in GPX format. The following figure presents these tasks.

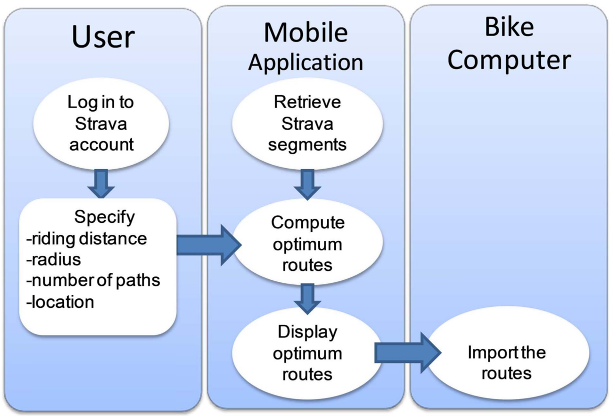


Fig. 5. Our overall system.

A. Acquiring Route Segments from Strava

The two main classes that we use to acquire route segments from Strava are the GetLocation and JSON Task classes. Our system obtains a location on Google Maps (for Android) from a user and then uses the GetLocation class to obtain the GPS location [19]. Then, the JSON Task.execute command is invoked to execute the segment Explore command from the Strava API. The command returns the most popular segment objects, each of which consists of the ID, name, average grade, latitude, and longitude of the start and end points of each segment (Fig. 3).

The Explore command returns a maximum of 10 segments regardless of the size of the area specified. If the specified area is large, 10 segments will be too few to find an optimal route. Hence, if the user-specified area is larger than eight square kilometers, we divide that area into four grids with the user-specified location at the center (Fig. 6). Then, we call the Explore command four times (once per grid square), yielding a total of up to 40 segments. Fig. 7 shows pseudocode for the algorithm.

When acquiring Strava segments, we create a special segment graph G(V,E) for later use in finding all possible cycling routes. V is a set of vertices consisting of a start location (with latitude and longitude coordinates), as well as a start and end point of all segments returned by Strava API calls. The segment graph is created as described below.

$G = (V, E)$
$V = \{\text{a user-specified location}\} \cup W$
$W =$ a set of start points and end points of Strava segments returned by the Acquiring Route Segments algorithm.
$E = \{(v_i,v_j) \mid f(v_i,v_j) = \text{true and } \forall v_i \in V, \forall v_j \in V\}$

The function f returns true if v_i is a start point and v_j is an end point of the same Strava segment. Additionally, we store segment information for each edge.

B. Finding all Possible Cycling Routes

We apply the DFS algorithm to find all possible cycling routes based on user preferences in our segment graph G, which results from the Acquiring Route Segments algorithm (Section A). In graph G, starting from a location *p*, we search for the nearest segments within a user-specified radius around the point *p* (Fig. 8).

When we arrive at a point *p*′, we check if the total route distance plus the distance from *p* to *p*′ is less than the target distance. If it is, we proceed to the other endpoint of the *p*′ segment and then recursively travel to all points adjacent to *p*′. To calculate the straight-line distance between two geolocations over the Earth's surface, we use the Haversine formula [Chris Veness as cited in 4].

Fig. 9 presents our proposed algorithm, the Radius Search algorithm, which is derived from the DFS algorithm. Before the first call to RadiusSearch (*p*), we create an empty list (a_route) to store a list of route segments and another empty list (routes_list) to store a list of discovered routes.

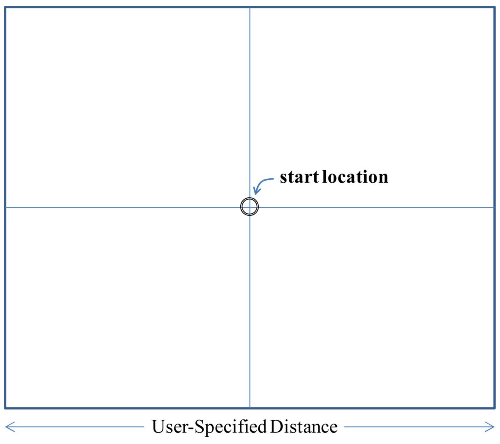


Fig. 6. Grid with four squares to find Strava segments [20].

```
• Repeat
        Get the GPS location of a location specified by a user
        If can_get_location Then
           If specified_area > 8 //km.
              Calculate locations of 4 grids (user location is center)
                    (top right and left, bottom right and left.)
           Else
              There will be only one grid
        For each grid
           Execute http requests to fetch JSON segments from
                                                   Strava API
• Until can_get_location
```

Fig. 7. Acquiring Route Segments algorithm.

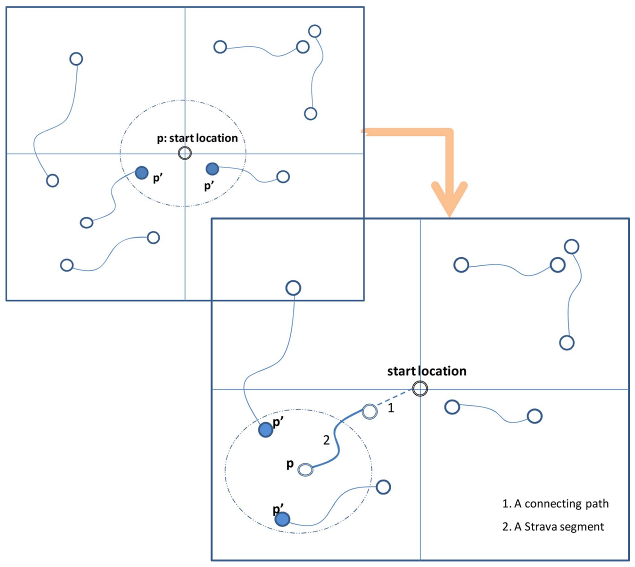


Fig. 8. Sample route graphs.

```
RadiusSearch (point p) {
   S =  a set of nodes (p') in G within a radius x km around p
   for each segment //ordered by segment lengths
    //a segment is an edge connecting p' in S to the other endpoint
    { if not (segment.visited) and
          (a_route.distance+distance(p, p') < target_distance)
          { a_route.push(segment) // p' is an endpoint of this segment
            segment.visited = True
            RadiusSearch( the other endpoint of p') }
    }
    if S is empty
       routes_list.append(a_route)
    a_route.pop() }

Main
   Graph G = Acquiring route segments
   target_distance =  user specified route distance
   routes_list = [ ]
   a_route = [ ], a_route.distence =0
   p = a user specified location
   RadiusSearch (p)
```

Fig. 9. Radius Search algorithm.

C. Finding the Optimal Routes

Our graph may have a maximum of 40 segments, each treated as a single node. Moreover, we assume that there exist connecting routes from each node to every other node. Hence, our graph is a complete graph in which every pair of distinct vertices is connected by an edge [18]. Given n nodes, there is a maximum of $n(n-1)/2$ edges.

In the worst-case scenario, there are 820 edges and 41 nodes (40 segments plus a starting location). This scenario occurs when a user specifies a search radius that covers the entire area. Traversing the entire graph requires 861 steps (41+820), resulting in a large number of calls to the Google API to find the lengths of all of the connecting paths. However, Google limits the number of free API requests to only 1,000 per day. Thus, instead, we use the straight-line distance to calculate the length, as explained in Section B. To find the optimal routes, we estimate the trade-off ratio for every route in routes_list as follows:

$$\text{Trade-off Ratio} = \frac{\text{length of all Strava segments}}{\text{length of all connecting paths}}. \tag{1}$$

Then, we rank all routes by their ratios and select only the top x optimal cycling routes (where x is specified by the user). Next, we connect a pair of segments in the resulting routes by calling the Google navigation API, the Directions service, which usually returns only one fastest route if we pass alternatives = false [20].

D. Displaying the Resulting Routes

As mentioned above, we can find optimal routes that consist of two kinds of segments: Strava and Google. For each route, we must integrate both the Strava segments and the Google connecting routes before displaying them as a single route on a map and exporting it to a bike computer. To export the Strava segments, we use the Strava API (getRouteAsGPX), which requires the identifier of a route as a parameter and returns a GPX file corresponding to the route [21]. To draw polyline overlays on an Open Street Map, we apply a Polyline class as follows [22]:

```
for each route in encodedRoutes
{ var coordinates =
    L.Polyline.fromEncoded(route).getLatLngs();
    L.polyline(coordinates, {color:'blue'} ).addTo(map);
}.
```

However, Google Maps does not use the GPX file format. Therefore, we must convert GPX files to KML files before calling the Google API to display the routes [23].

IV. DISCUSSION AND CONCLUSIONS

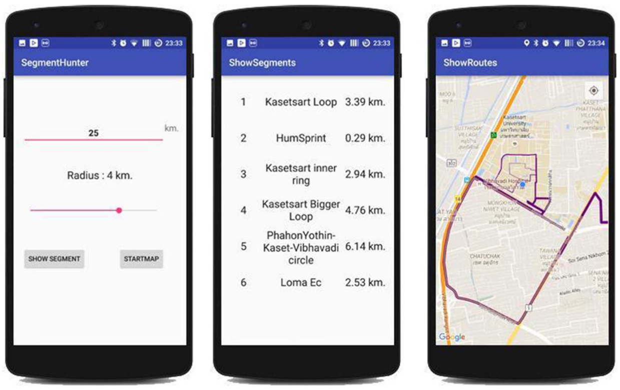
Here, we have developed an application to enable cyclists to find user-friendly cycling routes. Our app makes use of available social media data in the sports sector from the Strava website. The cycling route data suggested by the cycling community are typically safer than other routes [1], whereas existing cycling route planners (e.g., Google Maps) have issues with finding routes in private areas and around tourist attractions [3].

We wrote this application in Java using Android Studio to allow it to be used on mobile phones. A discovered route can be shown on a map on an Android device and exported as a GPX file to a bike computer.

By calling the Strava API, our application can retrieve popular route segments in a user-specified area. Then, it creates a graph consisting of a start and end points of those segments. We apply the DFS algorithm to find possible routes in the graph based on user preferences, specifically the route distance, search radius, number of optimal routes, and a starting location.

We obtain a set of optimal routes by applying a trade-off ratio to every discovered route. The calculation is based on the lengths of all segments and the lengths of all connecting paths. A given route may consist of unconnected parts, which our application connects using Google navigation.

To improve the performance of our search algorithm, we take into account user preferences. Instead of exhaustively searching all the nodes in a segment graph, we only visit



1	Kasetsart Loop	3.39 km.
2	HumSprint	0.29 km.
3	Kasetsart inner ring	2.94 km.
4	Kasetsart Bigger Loop	4.76 km.
5	PhahonYothin-Kaset-Vibhavadi circle	6.14 km.
6	Loma Ec	2.53 km.

Fig. 10. Our application shown on a mobile phone [24].

nodes within the search radius indicated by the user. Furthermore, we use the straight-line distance to calculate the distance between nodes and allow the user to specify the number of optimal routes to display, reducing the number of calls to the Google APIs.

Future work must perform usability analysis to improve the design of the user interface and user experience (UI/UX). In addition, user preferences on a cycling route (e.g., segment elevation) should be taken into account.

ACKNOWLEDGEMENTS

An earlier version of this work using a different algorithm was presented at the International Conference on Convergence Technology (ICCT 2018) in Tokyo with the title “Creating Cycle Routes on Strava Segments” [24]. The authors would like to acknowledge all referenced researchers for their contributions. We would also like to thank Assistant Professor Chaiporn Jaikaew for his useful advice on exporting data to a bike computer.

REFERENCES

[1] M. Loidl and H. H. Hochmair, “Do online bicycle routing portals adequately address prevalent safety concerns?,” *Safety*, vol. 4, no. 9, pp. 1-13, 2018. DOI: 10.3390/safety4010009.

[2] M. Schmidt and P. Weiser, “Web mapping services: development and trends,” in *Online Maps with APIs and WebServices,* Berlin, Heidelberg: Springer, pp. 13-21, 2012. DOI: 10.1007/978-3-642-27485-5_2.

[3] S. Hu and T. Dai, “Online map application development using Google Maps API, SQL database, and ASP.NET,” *International Journal of Information and Communication Technology Research*, vol. 3, no. 3, pp. 102-110, 2013.

[4] A. Pinandito, A. P. Kharisma, and R. S. Perdana, “Framework design for map-based navigation in Google Android platform,” *Journal of Telecommunication*, vol. 10, no. 1-8, pp. 35-40 , 2018.

[5] Strava, en.wikipedia.org, 2017, [online] Available: https://en.wikipedia.org/wiki/Strava.

[6] USAF, “Global positioning system standard positioning service signal specification,” *US Government*, 1995.

[7] GPX: the GPS exchange format, 2017, [online] Available: http://www.topografix.com/gpx.asp.

[8] The GPS exchange format, 2017, [online] Available: http://www.topografix.com/gpx_for_users.asp.

[9] GPS exchange format, 2017, [online] Available: https://en.wikipedia.org/wiki/GPS_Exchange_Format.

[10] HTTP/1.1: Request, 2017, [online] Available: https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5.

[11] J. Lennon, “Introduction to JSON,” in *Beginning CouchDB*, New York, NY: Apress, pp. 87-105, 2009. DOI: 10.1007/978-1-4302-7236-6_6.

[12] C. Bell, Strava Segments, [online] Available: https://www.doogal.co.uk/strava.php.

[13] Explore segments, Strava Developer, [online] Available: http://developers.strava.com/docs/reference/#api-Segments-exploreSegments.

[14] Polyline, Google APIs for Android, 2017, [online] Available: https://developers.google.com/android/reference/com/google/android/gms/maps/model/Polyline.

[15] Encoded polyline algorithm format, Google Maps APIs, 2017, [online] Available: https://developers.google.com/maps/documentation/utilities/polylinealgorithm.

[16] Best practices using google maps APIs web services, Google Map Platform, [online] Available: https://developers.google.com/maps/documentation/roads/web-servise-best-practices.

[17] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., Cambridge, MA: MIT Press and McGraw-Hill, 2009.

[18] R. Lafore, *Data Structures and Algorithms in Java*, 2nd ed. Indiana: Sams Publishing, 2002.

[19] Google direction library, 2017, [online] Available: http://www.akexorcist.com/2015/12/gogole-direction-library-for-android-th.html

[20] Developer guide, Google Maps Platform, [online] Available: https://developers.google.com/maps/documentation/directions/intro#RequestParameters.

[21] Polyline, [online] Available: https://docs.eegeo.com/eegeo.js/v0.1.680/docs/leaflet/L.Polyline/.

[22] M. Needham, Leaflet: Mapping Strava runs/polylines on Open Street Map, [online] Available: https://markhneedham.com/blog/2017/04/29/leaflet-strava-polylines-osm/.

[23] KML and GeoRSS layers, Google Maps Platform, [online] Available: https://developers.google.com/maps/documentation/javascript/kmllayer.

[24] W. Kao-Ian and I. Mulasastra, “Creating cycle routes on Strava segments”, in *Proceeding of the 2nd International Conference on Cultural Technology*, Tokyo, Japan, pp. 143-146, 2018.

Intiraporn Mulasastra
Assistant Professor, Department of Computer Engineering, Faculty of Engineering, Kasetsart University, Bangkok 10900, Thailand
Education:
2016 | Ph.D. in Management of Technology, Asian Institute of Technology, Thailand.
1990 | MS in Computer Science, University of Maryland Graduate School, USA.
Research Interest: Data quality, Data analytics, e-Government, Convergence Technology

Wichpong Kao-ian
A systems engineer at Kasikorn Business-Technology Group, Bangkok, Thailand.
Education:
2016 | Bachelor of Engineering in Computer Engineering, Faculty of Engineering, Kasetsart University, Bangkok 10900, Thailand
Research Interest: Artificial Intelligence, Data analysis, Cloud computing