Trajectories medoid and clustering

Mingyue Xie

Master's thesis

School of Computing

Computer Science

17.06.2019

The UNIVERSITY OF EASTERN FINLAND, Faculty of Science and Forestry, Joensuu
School of Computing
Computer Science
Mingyue Xie: Trajectories medoid and clustering
Master's Thesis
Supervisors of the master's Thesis: Pasi Fränti and Radu Mariescu-Istodor
February 2019
Abstract: This thesis is focusing on the developing of web page tool to calculate the representative among a set of trajectories by medoid method. It includes the tool introduction, trajectory distance calculation algorithm, medoid trajectory calculation, and similarity clustering. The work is based on Mopsi website.

Keywords: trajectory, medoid, similarity, clustering, representative

Foreword

I want to thank God who has protected me from different kinds of dangers and all difficult situations, who have given me the heart and wise to face those challenges.

I want to thank my thesis advisor Professor Pasi Fränti for reviewing my thesis draft and giving me a detailed suggestion to optimize my thesis. I am very grateful to Dr. Mariescu-Istodor for giving me many suggestions about programming and research. I want to thank the University of Eastern Finland and the IMPIT program for providing the chance to complete my master study.

I am also grateful to Dr. Oili Kohonen and my friend Nancy, who has helped me.

Finally, I would like to thank my family for giving me mental and financial support during my study life.

List of abbreviations

Ajax	Asynchronous JavaScript and XML
C-SIM	Cell-based similarity
DTW	Dynamic Time Warping
EDR	Edit Distance on Real Sequence
GPS	Global Positioning System
IRD	Interpolated route distance
JSON	JavaScript Object Notation
LCSS	Longest Common Subsequence
PAM	Partition Around Medoid algorithm
PNG	Portable Network Graphics
UTM	Universal Transverse Mercator
WGS	World Geodetic System
XML	Extensible Markup Language

Contents

1 Introduction

The *Global positioning system* (GPS) is a satellite-based navigation system made up of at least 24 satellites that transmits signal and orbital parameters, GPS receivers use this information to compute the precise location of the user's location and even the speed of movement [27]. With the development of satellite positioning technology, people can easily obtain a huge amount of trajectory data from moving objects. For example, in [28], the author mentioned that in Beijing, there are about 1.44 million personal trips generated by GPS-equipped taxis composed of GPS points.

When we have obtained many trajectories, we can analyze the data. An effective method is to find similar trajectories and group them into clusters, which helps to reduce the data by eliminating redundant information. In this thesis, we will study methods to find the representative trajectory in a given set. Visualizing the data is a method to find out hidden information from the data. In this thesis, we introduce a tool to visualize the trajectories and a user-friendly interface that provides many useful options in trajectory analysis, such as trajectory distance calculation and trajectory simplification.

A large amount of GPS data will bring problems such as displaying delays. For example, excessive storage size causes that the process of downloading data is not smooth and the waiting time is too long. When response time is important such as website interaction between the user and GPS data, it is necessary to simplify the data. A *polygonal approximation* method in [1] is used in our project. It reduces the trajectory points so that GPS trajectory's visualization quality is not compromised but the time cost on operating the trajectory is greatly reduced [1].

There are many definitions of *clustering*. In this thesis, clustering is defined as partitioning a set of trajectories into subsets, so that trajectories in the same group are more like each other than to those trajectories in the other groups.

We can apply various methods for trajectory clustering. In [2], the author introduced a *partition-and-group* framework for trajectory clustering. In [3], people apply *mixtures of regression models* on clustering. The result is a set of clusters which are composed of trajectories. Our research problem is to find a representative trajectory for each

cluster. Since compared with presenting many trajectories, a representative can help to achieve a good visualization and less downloading time on GPS data. In this thesis, our focus is on how to find out the representative for each trajectory cluster and visualize the result on the web page.

The user can directly see the trajectory cluster and its representative, so a map on the webpage is needed, such as Figure 1; also, interactive designs including user-click event and hovering event which target at operating data to get more detailed information are also important.

Based on those considerations, we have developed a tool in Mopsi platform which can: (1) Present a list of trajectories and plot them on a map. (2) Calculate the representative from trajectory set and highlight it on the map. (3) Provide means to select the trajectories to be processed. (4) Provide different options for processing the data and related representative calculation methods, so that user can compare how the calculation affects by applying various inputs. The tool helps the user to do further academic-related research. (5) Meanwhile, since trajectories have many points, but the algorithms for computing similarities are slow, we provide two ways to speed up the process: parallel computing in Section 5.5.1, and polygonal approximation in Section 2.4.

1.1 Mopsi

Mopsi [1] is a website that helps users to find where their friends are and what is around them [4]. There are trajectories recorded by users. Those trajectories are displayed on the map with detailed information, such as speed, traveled distance and user's transportation mode (walking, running, cycling, skiing) which is automatically inferred by the method in [41]. Mopsi allows the user to search trajectories in different ways. It also provides recommendations and tools for managing data collection.

[1] http://cs.uef.fi/mopsi

1.2 Data Collection

Mopsi data has two types: geo-tagged photos and trajectories. Geo-tagged photos contain information about their location and recorded time. The trajectory is a set of GPS point stored at a fixed interval. In this thesis, we use those trajectories as the data source. According to [12], there were more than 10000 trajectories recorded by more than 2400 users in 2017, and the number of users and trajectories have been increasing since then. Most of the trajectories are in Joensuu, Finland. The data structure is in Table 1.

Table 1. Data properties.

Column	**Type**	**Description**	**Example**
Latitude	Double	Latitude value of point	62.926880 (62° 55' 36.7674")
Longitude	Double	Longitude value of point	23.184691 (23° 11' 4.8876")
Timestamp	String	The timestamp for the point	1559983789 seconds
Altitude	Double	Altitude of point	-1.0 meter

1.3 Main work

The contribution of this thesis is the following. We have developed a web tool called Medoid page, which can calculate the representative trajectory for a set. The tool provides eight alternative measures on how to compute the distance between two trajectories, and an option for applying trajectory simplification. The tool also includes trajectory visualization and user interaction. In the Mopsi page, the user can also customize the trajectories set to calculate the representative. By applying this tool, one

can study the difference among those trajectory distance methods; find out the speed and quality effect of using trajectory simplification. The details are in Section 5.

Figure 1 shows four sample trajectories from the user Pasi; when applying the *IRD* distance (left) and *Fréchet* distance (right), distance here means the distance among trajectories. The brown curve is the medoid trajectory. So, when we use different distance methods, the distance between trajectories can be different so that the medoid result can be different. For example, the IRD method minimizes the sum of distances while Fréchet minimizes the maximum distance. In this way, just like Figure 1, the medoid trajectory may vary.

Figure 1. The same trajectory set has different medoid depending on whether we use IRD distance (left) or Fréchet distance (right).

1.4 Structure

There are eight Sections in the thesis. Section 1 contains the illustration of research background, Mopsi introduction, and an overview of our developed Medoid tool page. Section 2 contains the definitions of GPS point, trajectory, and trajectory simplification. Section 3 is about the trajectory distance. Section 4 is about trajectory clustering. Section 5 is about the implementation and technology used in the Medoid page. Section 6 are experiments; including efficiency comparison and accuracy comparison. Section 7 are conclusions.

2 Trajectory

In this Section, we will introduce GPS point, trajectory, route, and trajectory simplification. Next, we will explain those terms in a more detailed way. *GPS point* is a precise geographical location on the earth; it usually contains latitude and longitude and timestamp. By using GPS devices or positioning software, we can collect those GPS points. In our case, GPS points form the trajectory that we analyze in *World Geodetic System* [31].

2.1 Trajectory

A *trajectory* is a path that describes the movement of a user; it is composed of GPS points ordered by time. An example from Mopsi is in Figure 2. The timestamp is the number of seconds passed since midnight on 01/01/1970. It is useful because it can represent all time zones at once [29]. Timestamp 1552489200 represents Wednesday, March 13, 2019, 11:00:00 (am) in New York, and Wednesday, March 13, 2019, 17:00:00 (pm) in Helsinki.

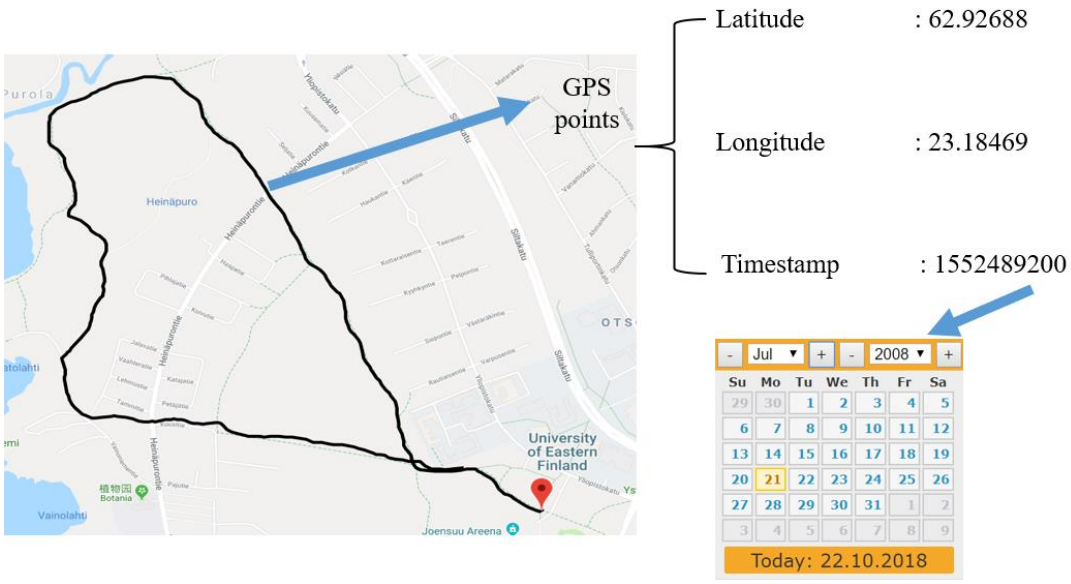

Figure 2. A trajectory on the map from The University of Eastern Finland to the Joensuu Areena with sector movement

2.2 Route

In this thesis, there is a difference between trajectory and route. A *route* is a selected course of the travel path. It refers to a representative of many trajectories, which can indicate the common movement among those trajectories.

Figure 3 shows an example of a Joensuu-Kuurna-Kulho route (right) and a set of similar trajectories (left) plotted on the same map. Although most parts in those trajectories are overlapping, we can find that there are many variations among those similar trajectories in Figure 3 (left). However, we can still identify the route which is in Figure 3 (right) by removing the variations.

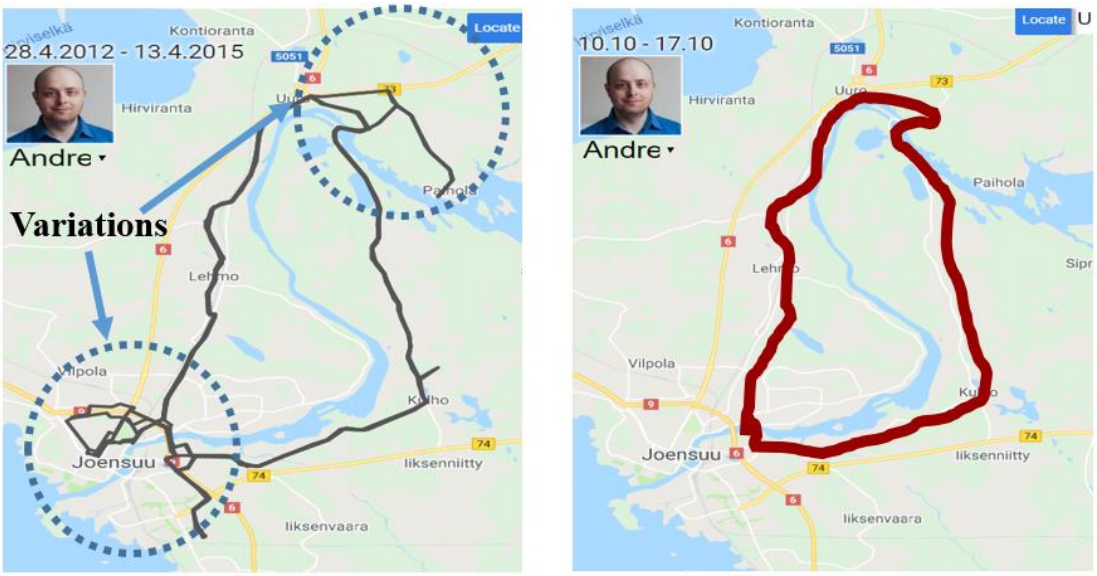

Figure 3. A set of similar trajectories (left) representing different cycling attempts on the Joensuu – Kuurna – Kulho trajectory (right)

We can easily find the representative route from Figure 3 (left) since there are few trajectories looks similar on the map. By treating them as a set, then select one of them to become the representative one; but when there are many trajectories such as Figure 4, how do we find out the similar trajectory set and extract the corresponding representative route from the set? It can be quite difficult to spot the similar ones. We, therefore, consider using clustering to find out similar trajectories.

Figure 4. Many trajectories on the map generated by Mopsi's user Pasi

2.3 Polygonal approximation

There are many methods in data simplification. *Polygonal approximation* [1] is one of the methods to help simplify GPS trajectory, and Mopsi has been using this method for trajectory simplification.

According to [12, 42], we know that polygonal approximation is very effective at reducing the number of GPS points while preserving the approximate shapes on the map. Meanwhile, the Mopsi system can display trajectories consisting of over 3.5 million points in less than 2 seconds. In this way, the performance of processing and visualization will improve greatly. In the Medoid page, the 'reduced trajectory' means those trajectories with applying polygonal approximation. Detailed implementation is in Section 5.3.

3 Trajectory similarity

When we consider finding the similarity between the two trajectories, we need to compute the distance. There are many methods proposed, of which I have chosen eight to implement on our Medoid page. In this section, all the figures that illustrate trajectory distance are from this Mopsi page[2].

3.1 Cell Similarity

Cell Similarity utilizes a grid to compute the similarity between two trajectories. We can see an example in Figure 5. According to [5], it first retrieves the cell representation and then calculates the similarity measure using the cells. It calculates how many cells are in common relative to the total number of cells. The disadvantage is point's order is not used. According to [5], this method is the least affected by increasing or decreasing the sampling rate and performs well under noise and point shifting.

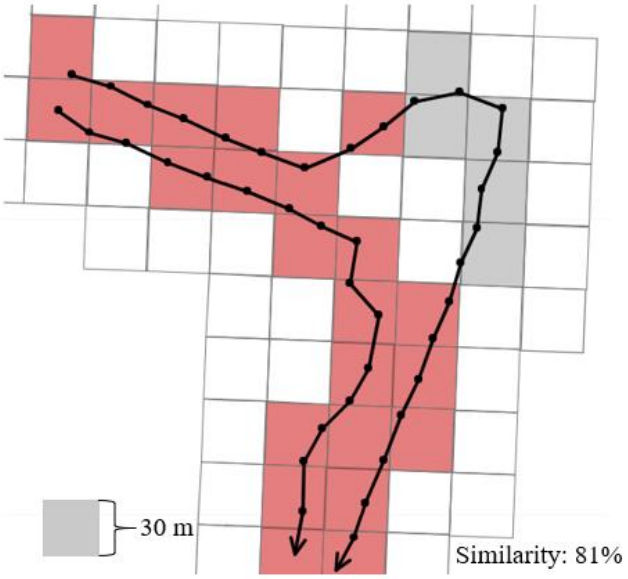

Figure 5. C-SIM distance with cell length: 30m

[2] http://cs.uef.fi/mopsi/routes/contextSimilarity/

According to [5], we use the *Jaccard index* to calculate trajectory distance. It measures the overlap degree of two trajectories by the following formula, where C_{A} and C_{B} represent the sets of cells approximate trajectory *A* and *B*. C_A^d and C_B^d represent the extra cells added from dilation.

$$S(C_{\mathrm{A}}, C_{\mathrm{B}}) = \frac{|C_{\mathrm{A}} \cap C_{\mathrm{B}}| + |C_{\mathrm{A}} \cap C_B^d| + |C_{\mathrm{B}} \cap C_A^d|}{|C_{\mathrm{A}}| + |C_{\mathrm{B}}| + |C_{\mathrm{A}}| + |C_{\mathrm{A}} \cap C_{\mathrm{B}}|} \quad (1)$$

3.2 Longest Common Subsequence

Longest Common Subsequence (LCSS) originates from string processing where it has been used for string similarity [13]. For example, the longest common subsequence for strings 'ABABC', ' BABCA' and 'ABCBA' is 'ABC'. When we calculate trajectories' distance, it allows some GPS points unmatched to match some sequences in trajectories [17]. The basic idea is ignoring the GPS points which are far away. The function below [14] describes the principle.

$$LCSS(A,B) = \begin{cases} 0, if\ n = 0\ or\ m = 0 \\ 1 + LCSS\big(Rest(A), Rest(B)\big), if\ d\big(Head(A), Head(B)\big) \le \varepsilon \\ max\Big(LCSS(Rest(A), B), LCSS\big(A, Rest(B)\big)\Big), otherwise \end{cases} \quad 2$$

In Figure 6, *m* and *n* are the lengths of two trajectories *A* and *B*; ε is a threshold to determine whether to take this point into account. If *A* is composed of a list of GPS points such as (a_1, …, a_n), then *Head*(*A*) is the first point a_1 in *A*. *Rest*(*A*) means the rest of points, which is (a_2, …, a_n), *d* measures the distance between two points.

For some trajectories, there exist noise. When a distance method requires pair-matching for every GPS point inside the trajectory, the noise will affect the distance result. The advantage of LCSS is: By ignoring 'far-away' points, it can measure the distance between those trajectories that may have lower quality or noises. So, it has better robustness against noise.

In Figure 6, we use LCSS to measure the similarity between two trajectories *A* and *B*. Here ε is 45 m, and the distance is 15. We can find out that a few points are ignored since they are far away.

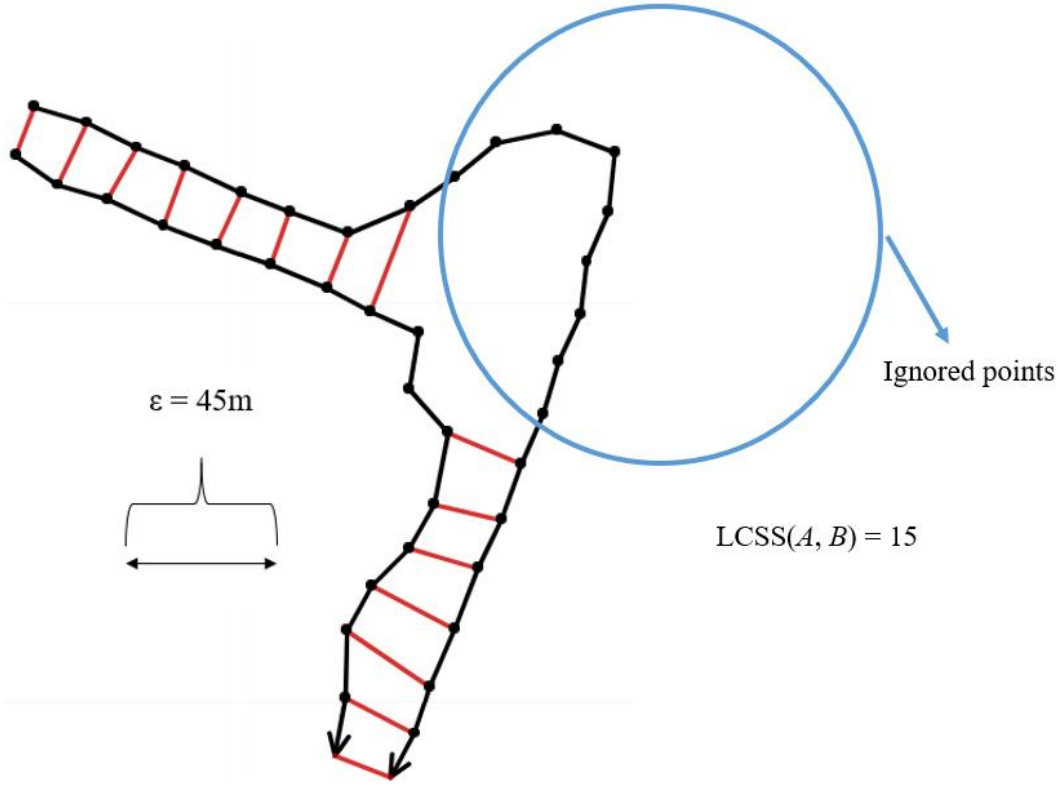

Figure 6. Use LCSS to measure two trajectories' similarity

The disadvantage is that it may lead to some inaccuracy. According to [17], for two trajectories *A* and *B*, *A* has a list of points $\{a_1, a_2, \ldots, a_n\}$, *B* is composed of $\{b_1, b_2, \ldots, b_n\}$, when we increase the sample rate of *A* and *B*, the *A'*s transformed trajectory A_i will be $\{a_1, a_{1.5}, a_2 \ldots, a_n\}$ and B_i transformed trajectory *B'* will be $\{b_1, b_{1.5}, b_2 \ldots, b_n\}$; when we decrease the sample rate of *A* and *B*, the transformed trajectories will be A_d $\{a_1, a_3, a_5, \ldots, a_n\}$, and $B_d\{b_1, b_3, b_5, \ldots b_n\}$. A_i and B_i have more points than A_d and B_d. Thus, A_i and B_i's common subsequence detected by LCSS will be larger than A_d and B_d 's common subsequence. So, the LCSS values will be different when the sample rates are different. In Mopsi, when we apply polygonal approximation on the trajectories for simplification, the LCSS value will be reduced compared with the original trajectories.

3.3 Edit Distance on Real Sequence

This method is *Edit distance on real sequence* [15]. It originates from the edit distance [16], which describes the number of times for inserting, deleting, and replacing to convert string *A* into string *B*. There are many definitions for the edit distance, for example, the *Levenshtein distance* contains removing, inserting and replacing a character in the string. When we try to convert "*bitten*" to "*sitting*". First we need to replace "*b*" to "*s*", so the string now is "*sitten*"; next we replace "*e*" with "*i*", the result will be "*sittin*"; finally, we need to insert a "*g*" at the end of the string, so the string now is "*sitting*". There are three operations in total, so the Levenshtein distance is 3.

EDR's definition is in [14] and equation 3. If we have trajectories *A* and *B* with the lengths *n* and *m*, EDR distance between *A* and *B* is the minimum number of inserting, deleting, and replacing operations to convert *A* to *B*. Two points are considered as different if distance between them is bigger than ε. Same as LCSS, *Rest*(*R*) means the rest of the points except the first one in trajectory *R*. Figure 7 displays the trajectories' distance calculated by EDR. Nine operations are required to convert *A* to *B* in this case, so the EDR distance is 9. We can see the definition below [14].

$$EDR(A,B) = \begin{cases} n, & if(m = 0) \\ m, & if(n = 0) \\ \min\{EDR(Rest(R), Rest(S) + subcost, & \\ \quad EDR(Rest(R), S) + 1, & \\ EDR\big(R, REST(S)\big) + 1\}, & otherwise \end{cases} \tag{3}$$

Where

$$subcost = \begin{cases} 0, & if(dist(Head(A), Head(B)) \leq \varepsilon \\ 1, & otherwise \end{cases} \tag{4}$$

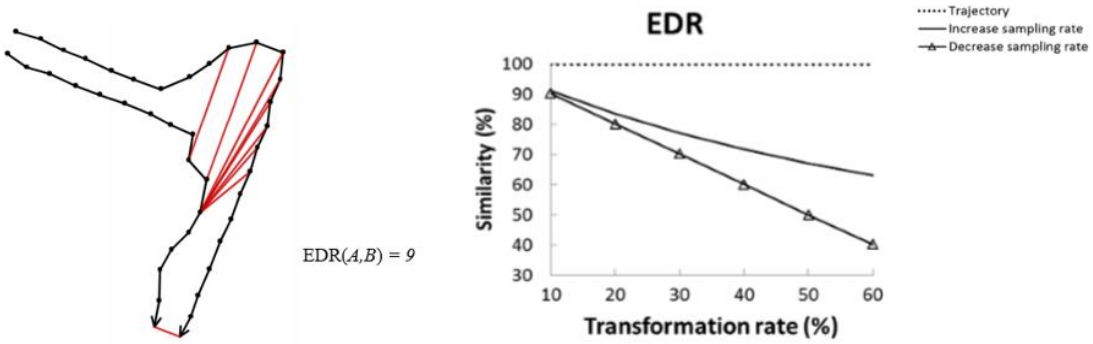

Figure 7. The EDR distance between trajectory *A* and *B* is 9(left). Decreasing sample rate will affect more on the similarity compared with increasing sample rate [5] (right)

There are many advantages of EDR mentioned in [15]. One of them is: it describes the distance between two points using 0 and 1, so compared with Euclidean distance, it reduces the influence of noise in the trajectory; but as is mentioned in [5], LCSS and EDR are very sensitive when we decrease the sample rate of trajectories compare with increasing the sample rate.

3.4 Interpolated Route Distance

It is a newly proposed measure called Interpolated Route Distance (IRD) [18]. The basic idea is to find the interpolated point from a trajectory to another one, and make those two points as a pair. If we can't find the interpolated from trajectory *A* to *B*, then vice versa, until the last point of the shorter trajectory has been visited, the average of summation of the distances between those paired points is the IRD distance.

Latitude and longitude are on the earth's ellipsoidal surface. However, when we calculate the IRD distance, it requires us to find the interpolated point from a GPS point towards a specific trajectory. So it's a good way to transform the GPS points with latitude and longitude into locations on a plane [32].

According to [18], if we have a point *a* in trajectory *A*, the interpolated point for *a* in trajectory *B* is b_p, the distance between *a* and b_p should be the shortest among all pairs between point *a* and other points in *B*. So, we use projection to calculate this point. The

problem is to find out the orthogonal projection onto a line. In linear algebra, the orthogonal projection of $\vec{v}$ onto a line spanned by nonzero $\vec{s}$ is below.

$$proj_{[\vec{s}]}(\vec{v}) = \frac{\vec{v} \cdot \vec{s}}{\vec{s} \cdot \vec{s}} \vec{s} \tag{5}$$

In this case, $\vec{v}$ is generated by the point a (a_x, a_y) and the previous point a_0 (a_{0x}, a_{0y}) on trajectory A, so $\vec{v}$ is (a_x - a_{0x}, a_y - a_{0y}), $\vec{s}$ is generated by point b and the previous point b_0 on trajectory B, so $\vec{s}$ is (b_x - b_{0x}, b_y - b_{0y}), the target is generating the projected point b_p on the segment composed of b_0 and b. We assume b_p is (b_{px}, b_{py}) then $proj_{[\vec{s}]}(\vec{v})$ is generated by (b_{px} - b_{0x}, b_{py} - b_{0y}), so with the formula 5 and known point b_0, we can calculate b_p easily. Meanwhile, when $proj_{[\vec{s}]}(\vec{v}) > 0$ and length of the projected vector $proj_{[\vec{s}]}(\vec{v})$ is greater than $\vec{s}$, or $proj_{[\vec{s}]}(\vec{v}) < 0$, this projected point b_p is not on the segment formed by b_0 and b. So, we choose the point that has a shorter distance with point a from b_0 and b.

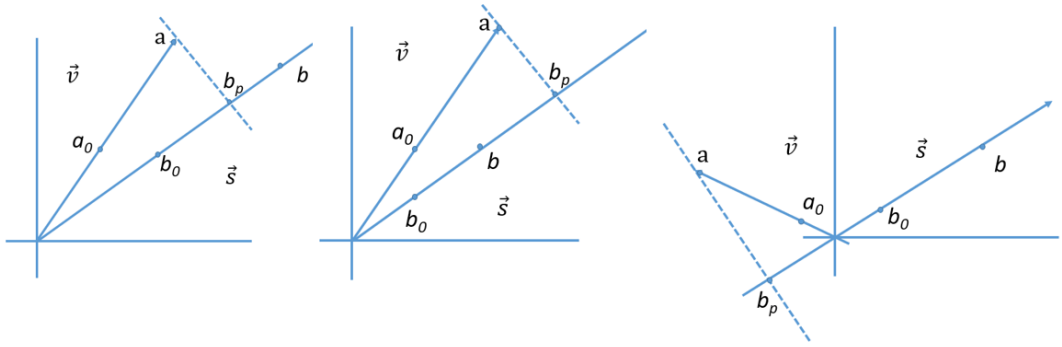

Figure 9. Calculate interpolated point b_p by orthogonal projection (Three possible cases); in the first case, b_p is in the segment between b_0 and b; the second one and third one b_p is outside of the segment between b_0 and b.

As is mentioned in [18], it fixes the problem that different sample sizes cause different similarity. So it is better than LCSS and EDR, since these two methods will be affected by the decreasing of sample size greatly. Meanwhile LCSS and EDR's time complexity are both $O(N^2)$, but IRD's time complexity is $O(l_1+l_2)$ where l_1, l_2 are number of points in A and B. For each point, it needs to find out the closest point by comparing the one with others for many times, which leads to inefficiency. The IRD distance is below. When we sum up distances generated by those paired points, then divide it by the times, we can get the IRD distance.

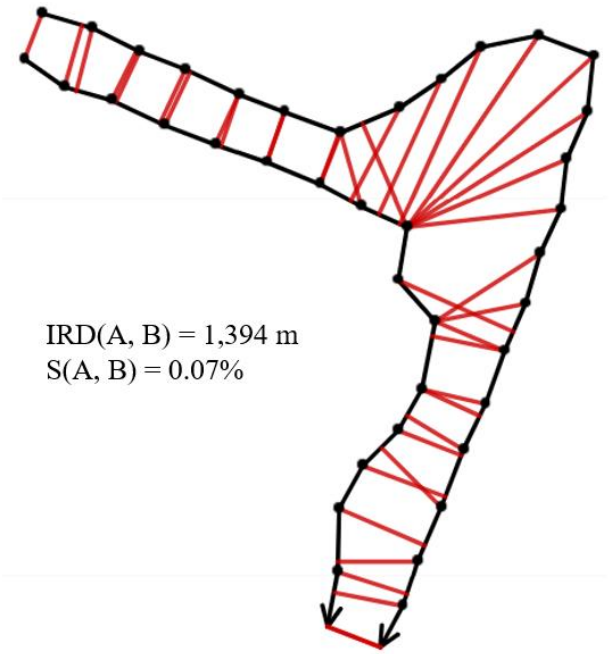

Figure 10. IRD distance

3.5 Fréchet Distance

When a person is walking a dog, we consider this person is on one curve, and the dog is on another curve, there is a leash between them. When backtracking is not allowed, what is the minimum length that is enough for traveling both curves? Calculating that length is the intuitive definition of Fréchet distance.

In [20], it introduces a distance method called Fréchet distance that can solve this problem. The idea is firstly computing proper polygonal approximations to the curves, then compute their coupling distance. This measure takes both the location and order of the points into consideration. The definition is in formula 6. In this case, *A* and *B* are the trajectories, if *S* is a metric space, *d* is the distance function of *S*, α and β are the arbitrary and continuous, nondecreasing functions from [0,1] onto [*a*, *b*]. Then the distance between *A* and *B* is the infimum over all reparameterizations α and β of the maximum over all t $\epsilon[0,1]$ of the distances in S between $A(\alpha(t))$ and $B(\beta(t))$. We can assume *t* is the time point, when we reach time t, the chosen point on trajectory *A* is $A(\alpha(t))$, and the chosen point on trajectory *B* is $B(\beta(t))$. We iterate the interval [0,1] to find out two points every time, if we use Euclidean distance, then it is easy to define $d(A(\alpha(t), B(\beta(t)))$, the Fréchet distance would be the value that makes the maximum distance reaches the greatest lower bound with this sampling method. It is sensitive to the change of sample rate and noise [5].

$$F(A,B) = inf_{\alpha,\beta}\, max_{t\epsilon[0,1]}\{d(A(\alpha(t), B(\beta(t)))\} \quad (6)$$

We can see in Figure 11 (left); there are two trajectories, and the location where the shortest 'leash' reaches its maximum required length. The distance is plotted red on the left. The right part illustrates the definition, for each pair of the point, we can always find out the biggest distance during the walking process, by changing the point we can make the 'leash' get shortest.

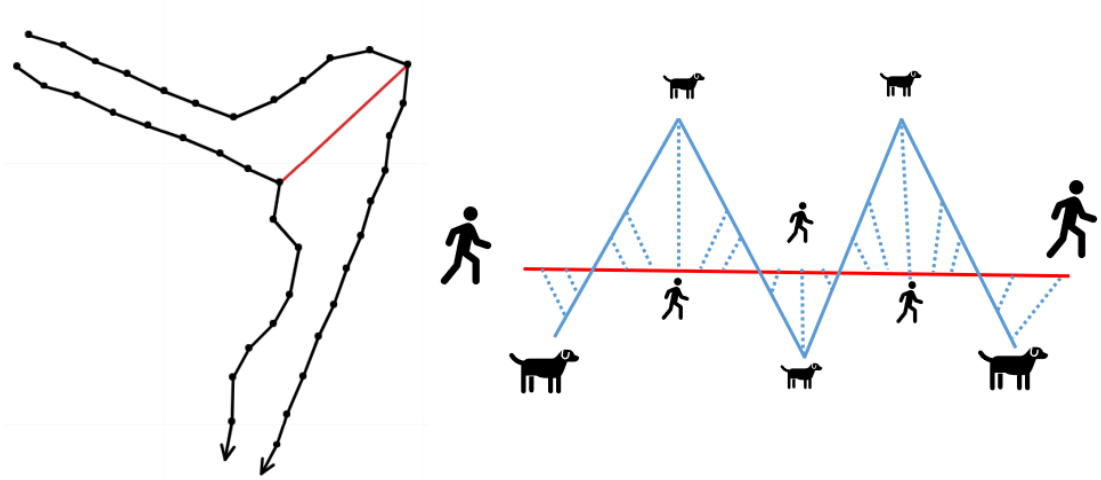

Figure 11. Fréchet distance for two trajectories on the left, the right one illustrates the definition of this method. The person is moving on the red line; the dog is moving on the blue curve, the sample point is taken within time range [*0,1*]

3.6 Hausdorff Distance

Hausdorff distance measures how far two subsets of a metric space are from each other. It is the maximum of all the distances from a point in one set to the closest point in another set. In [21], we know that the Hausdorff distance's definition is as follow:

$$D(A,B) = \max\{h(A,B), h(B,A)\} \tag{7}$$

The definition of the distance between trajectory *A* and trajectory *B* is:

$$h(A,B) = max_{a\in A}\, min_{b\in B}\, \|a-b\| \tag{8}$$

h(*A*, *B*) is directed Hausdorff distance from *A* to *B*. In *h* (*A*, *B*), for every point in trajectory *A*, it finds out the point *a* in *A* that has max distance to any point in *B*, then measures the distance from *a* to its nearest neighbor in *B*, which is *h (A, B)*. Vice versa to calculate *h* (*B*, *A*). By comparing *h* (*A*, *B*) and *h* (*B*, *A*), the bigger one is the Hausdorff distance.

Same as Fréchet distance, it works in the field needs shape comparison, but as said in [34], both of them fail to compare the trajectory as a whole because of the method definition. Meanwhile, the Hausdorff distance requires more computation than the Fréchet distance. We can see that in Figure 12, trajectories' Hausdorff distance is in red.

The results are the same for Hausdorff distance and Frechet distance, but according to [34], compared with Hausdorff distance, the discrete Frechet distance is not a metric. Meanwhile, the Frechet distance needs less computation. In Figure 13 from [34], we will find out that the distance variation between these two methods. The time complexity is $O(N^2)$.

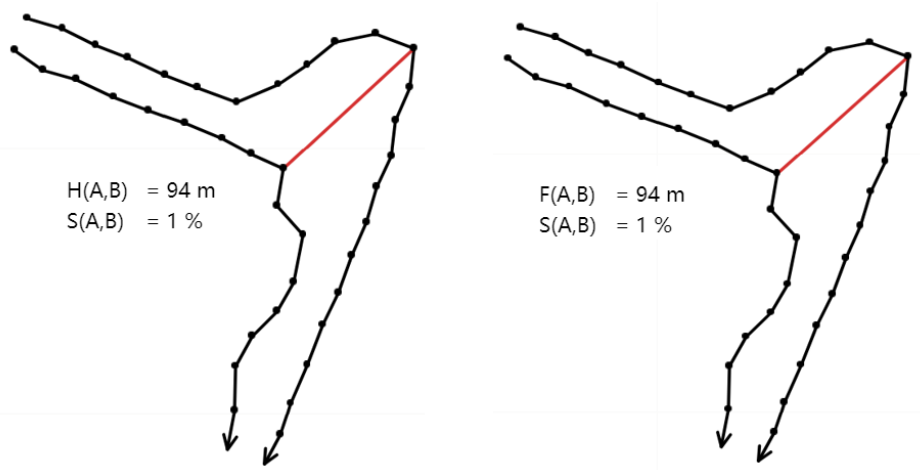

Figure 12. Hausdorff distance (left) and Fréchet distance (right), this case they have the same results.

In Figure 13, we can find out the drawback for Hausdorff and Fréchet distance, as is discussed in [34], these two methods can't take the trajectory as a complete part to do distance calculation. Although we can observe that T^1 and T^2 are the most similar pair among the three trajectories, but in Fréchet, they are the farthest because the maximum distance is six at the end of the trajectory. Meanwhile, Hausdorff distances for those three trajectories are almost the same, which also means low accuracy. These two methods are good at fields related to shape comparisons such as image comparison [35].

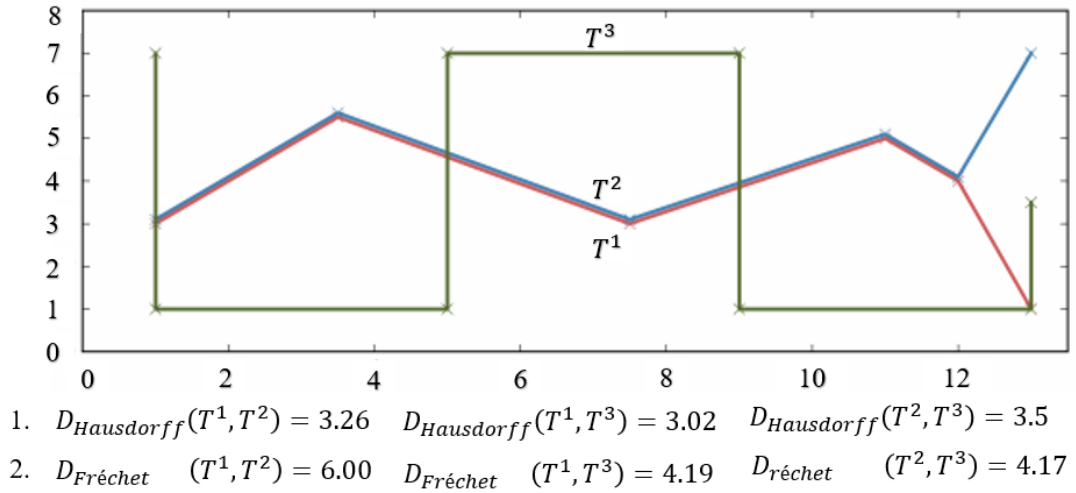

Figure 13. Three trajectories distances calculated by Hausdorff and Fréchet distance [34], T1 and T2 are more similar, but there is no clear difference in Hausdorff distance; Fréchet distance even indicates they are the least similar.

3.7 Euclidean Distance

Euclidean distance is the distance between two points in *Euclidean space*. In geometry, the Euclidean space has a two-dimensional plane. According to the *Pythagorean formula* [36], point p (p_1, p_2... p_n) and point q (q_1, q_2... q_n) are two points in the Euclidean n-dimensional space, the distance from p to q is as the formula below.

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} \tag{9}$$

For trajectories, as we have discussed in the IRD distance, we should transform latitude and longitude into Cartesian coordinates, then we can apply the distance function above. Since the time complexity is $O(N)$, N is the number of points for the shorter trajectory. When we have many points in trajectories, it would be much quicker to apply the Euclidean distance compared with others that have higher time complexity. However, according to [5], we know that Euclidean distance is sensitive to the local alignment. When the sampling rate of the trajectory is increasing or decreasing, or noise occurs, or there are some shifted points inside, it will lead to a large impact on the Euclidean distance, because misalignment will happen.

In Figure 14 (left), each GPS point is paired and connected with a GPS point in another trajectory. It is evident that for the longer trajectory, some GPS points are left with no point to get paired with. Which also might lead to lower accuracy. The difference between Euclidean distance and IRD distance is the type of aligned points. In the Euclidean distance, we use the original point in the trajectories to form a pair; in the IRD distance, we use the original point and its interpolated point to form a pair. S(*A*, *B*) is the reciprocal of distance. It measures the similarity between two trajectories. In Figure 14, *S*(*A*, *B*) denotes the trajectory similarity, Euclidean distance has less similarity since it ignored some points for the longer trajectories in the calculation.

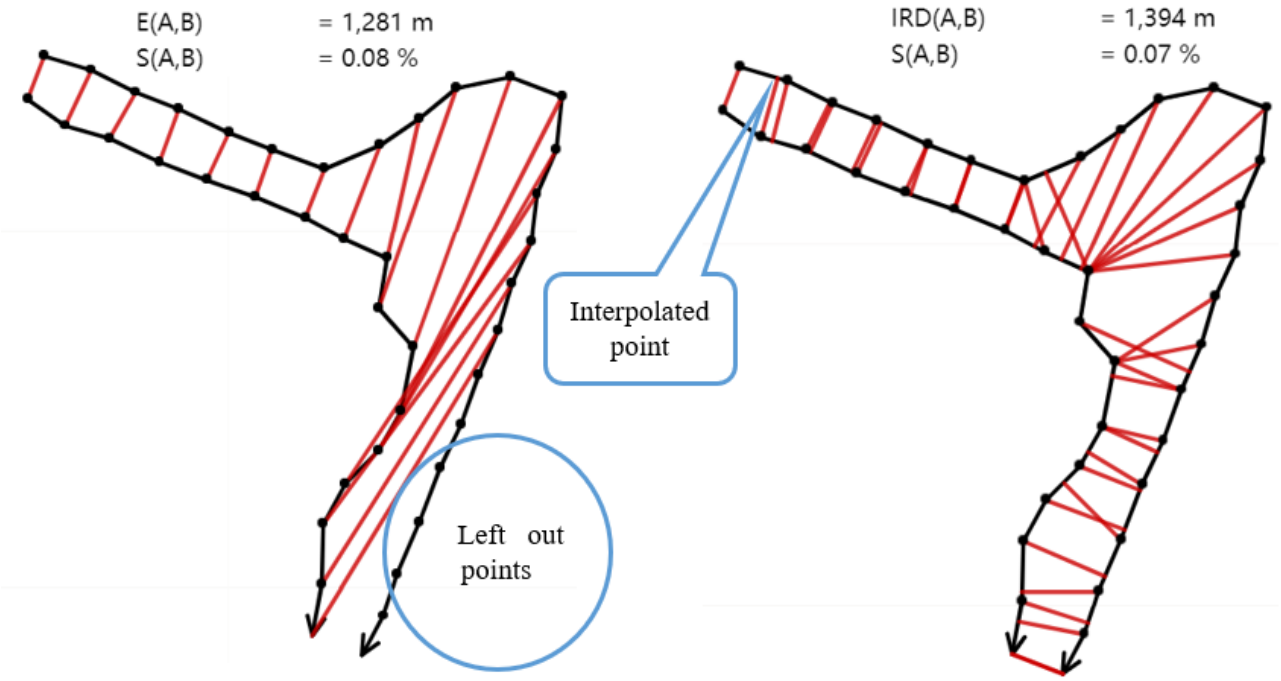

Figure 14. Euclidean distance (left) and IRD distance (right)

3.8 Dynamic Time Warping Distance

DTW [14] is a distance method that measures the similarity between two sequences; the basic idea is to calculate the optimal match between two given sequences. The optimal match means the match satisfies all the restrictions and the rules. It also has minimal cost. The definition [14] is below:

$$\mathrm{DTW(A,B)} = \begin{cases} 0, if\ n = m = 0 \\ \infty, if\ n\ = 0\ or\ m = 0 \\ dist\big(Head(A), Head(B)\big) + min\begin{cases} DTW\big(A, Rest(B)\big) \\ DTW\big(Rest(A), B\big) \\ DTW\big(Rest(A), Rest(B)\big) \end{cases} , otherwise \end{cases} \quad (10)$$

As is introduced in [14], the basic idea is to allow using the same point many times to get the best alignment, which is different from the Euclidean distance. If trajectory *A* is composed of GPS points $\{a_1, \ldots, a_n\}$, *Head*(*A*) means a_1 and *Rest*(*A*) means $\{a_2, \ldots, a_n\}$. In Figure 15, two trajectories' DTW distance are in red (left), we will find that for DTW method, some points are aligned with more than one point, which is also shown in the circle. It is widely used in speech pattern recognition of time series [37, 38]. However, as is mentioned in [5], DTW distance is sensitive to the increasing and decreasing of sample rate, which is the disadvantage.

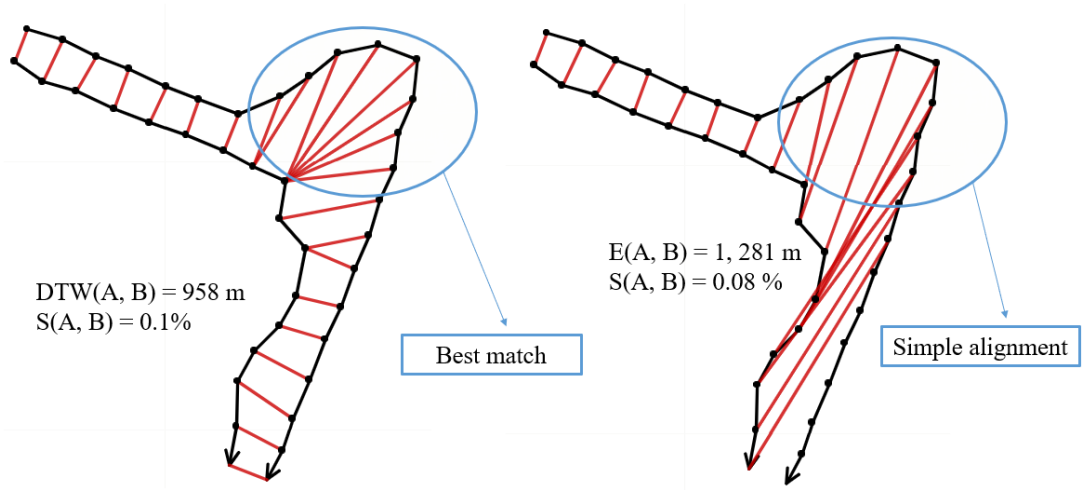

Figure 15. DTW method

4 Clustering

Clustering is the task to partition the objects that are more similar to the same group. Such kind of group is called cluster. There are many clustering algorithms nowadays, such as *K-means* [37], which iterates between optimal partitioning and centroid reloading [6]. Hierarchical clustering is a method that recursively merges objects in bottom-up order (agglomerative hierarchical clustering) or partitions objects in a top-down order (divisive hierarchical clustering) [7]. One of our goals is to find out the representative from the trajectories' cluster, so it is a prerequisite to get the trajectory cluster first.

4.1 K-medoids algorithm

K-medoids algorithm groups the data based on their distance to each other [37]. If we partition *n* objects into *k* clusters, each cluster has a *medoid* as the representative. Every object will be assigned to the cluster that has the minimum distance between this object and the medoid.

Medoid is the representative of a cluster that has the maximum sum of similarity to others in the cluster [9]. It should be one of the objects in the set, which is different from *Median* [43]. Median is the value separates the higher half from the lower half in the set, it could be the average of two middle numbers if set size is even number, otherwise it is the only middle one. As is introduced in [37], K-medoids is more robust compared with the K-means algorithm because it reduces the influence from outlier and noise; but the time complexity is $O(k(n-k)^2)$, k is the number of clusters, n is the number of objects, so it is high complexity. In [39], the author gives us a method to speed up the K-medoids algorithm, so that the time complexity could be reduced to $O(n^2)$.

There are many versions about K-medoids clustering, such as the algorithm uses Voronoi iteration [38], but the most common one is the *Partition Around Medoid* algorithm (PAM), the basic idea of PAM clustering is below:

Partition Around Medoid: Get clusters by K-medoids clustering

Input: K: the number of clusters; D: the dataset of n objects

Output: K clusters

Algorithm:

1. Randomly select K medoids to initialize K clusters
2. Assign each object to the cluster that has the minimum distance between the object and medoid
3. While the cost of the configuration decreases:
    (1) For each medoid m and each non-medoid object o:
    (2) Swap m and o, map each object to the closest medoid, recompute the cost (Sum of the distance of objects to their medoid)
    (3) If the total cost of the configuration increased in the previous step, then undo the swap.

4.2 Trajectory clustering

When we do trajectory clustering, the similarity among trajectories is the key to make the partition. As is introduced in Section 3, many methods can calculate the trajectories' similarity.

We have selected ten trajectories from Mopsi, after applying K-medoids, we got three clusters in Figure 16, the similarity measure is Fréchet distance, and we can find that those similar trajectories have been partitioned into the same clusters. Different distance methods can lead to different clustering result. The detailed comparison is in Section 6.5.

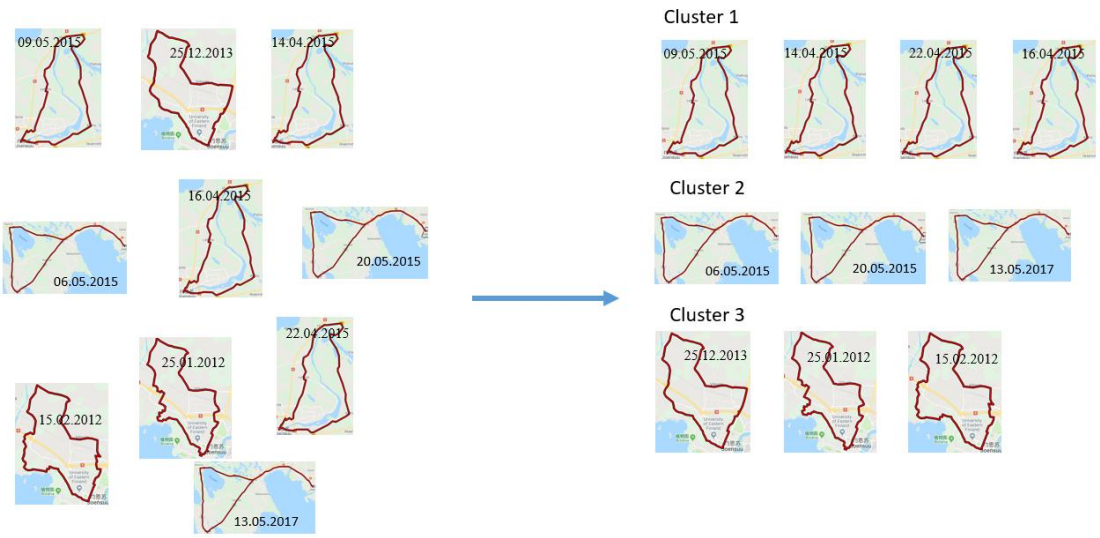

Figure 16. Clustering trajectories into clusters by K-medoid clustering

4.3 Representative-Medoid

For many clustering measures, it's common to get the representative for each cluster by computing the arithmetic mean value such as K-means. We can see this method in Formula 10.

$$\bar{x} = \frac{(x_1 + x_2 + \cdots + x_n)}{n} \tag{10}$$

From [2] we know that trajectory representative can illustrate the overall movement of the set. So, finding out the representatives of trajectories help us obtain the practical potential from a trajectory set.

In this case, x_i represents trajectory in the cluster, it is composed of a set of GPS points ordered by time. According to Formula 10, if we calculate the arithmetic mean value for trajectory cluster, trajectories' lengths should be the same. However different trajectory usually consists of a different number of points. So, it's hard to compute the average trajectory. According to [10], for the clusters that are hard to compute the average or unnecessary to calculate the average centroid, such as gene regression, it is common to choose medoid as the representative. So, we apply K-medoid in trajectory

clustering. In the K-medoids algorithm, the output is *k* clusters with *k* medoids as the representatives. The related experiment is in Section 6.

5 Implementation

In this Section, we will introduce the functions of the Medoid page. Section 5.1 illustrates how to choose trajectory cluster from Mopsi and my related work. Section 5.2 is about using reduced trajectory to calculate the medoid trajectory. Section 5.3 is about choosing a different method to calculate medoid; Section 5.4 is about technologies that we have used.

5.1 Choosing trajectory cluster

In this thesis, there are two methods to get the trajectory cluster. First, in Mopsi, each trajectory has its similarity list. When a user clicks the button, which is in Figure 17, a list of similar trajectories will be shown such as in Figure 18.

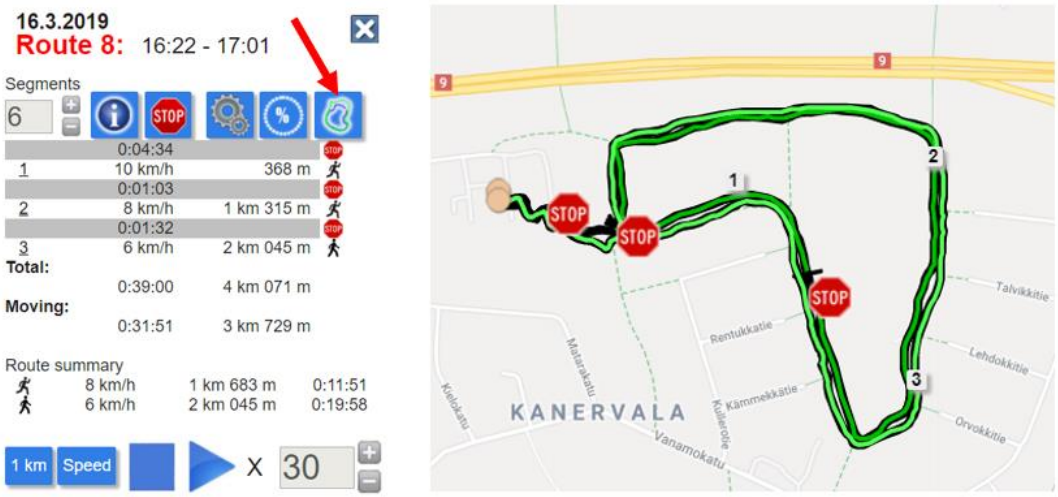

Figure 17. Click this button to find similar trajectory list for the trajectory on the right.

In Figure 18, those trajectories are in decreasing order of similarity. The information contains the user name, recorded time, transportation modes such as running and cycling. The percentage such as 99% shows the similarity between the corresponding trajectory and the original trajectory.

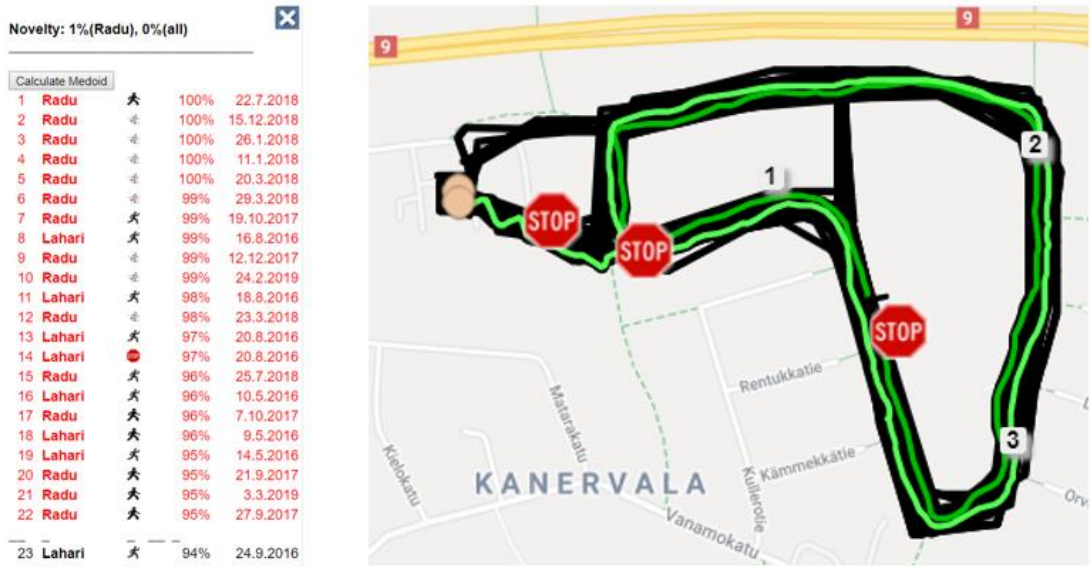

Figure 18. A list of similar trajectories plotted on the map with black.

So, these trajectories can work as a cluster to help to find out the representative. However, sometimes there can be more than 2000 similar trajectories, which means that if we want to analyze or visualize them, it will take a long time to process. For this reason, in [40], the author uses clustering to limit to the most similar trajectories from this list.

The basic idea is to use *Random Swap* clustering [8] on those similarities; after clustering, each cluster consists of trajectories with similarities, then we calculate the average similarity for each cluster, and choose the cluster with the highest average similarity. In Figure 18, this "Most similar" cluster is in a red font; the first 22 trajectories are the most similar ones. It was plotted with black on the map, compared with the original trajectory which is green in Figure 18, they look similar.

Based on this method, the user can change the lower boundary of this "Most similar" cluster, when the mouse is hovering on a trajectory in the similarity list, an arrow button appears, then we click it, this trajectory will work as the lower boundary of the "Most similar" cluster. We can see this process in Figure 19.

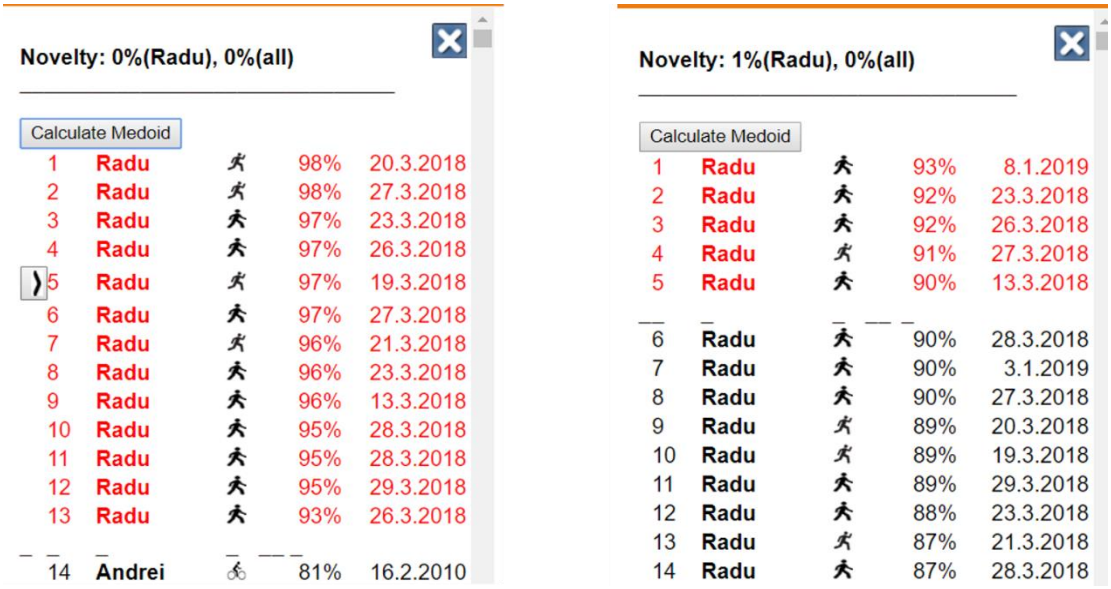

Figure 19. The user can change the boundary of the most similar trajectory set

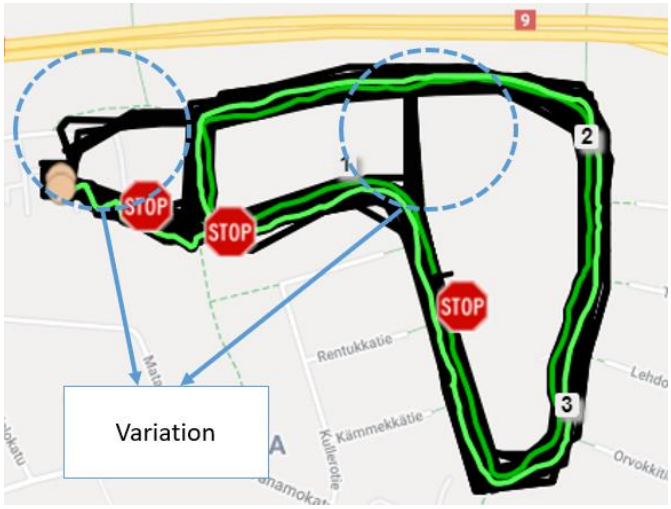

Figure 20. Original trajectory plotted with green color and "Most similar" trajectories plotted with black color.

After we have decided the "Most similar" cluster, it plotted the correlated trajectories on the map in Figure 20. We can see that the original trajectory almost covers those trajectories in the cluster; which indicates that those trajectories are similar to each other. The second method of choosing trajectory cluster is using recorded time, as is shown in Figure 21, "Most recent", "Week", "Year", "All", "Select dates", those buttons provides the trajectories according to the recorded time; after we choose the time, we can see there is a list of trajectories on the page, they are ordered by the time correlation,

from the newest to the oldest. We can also find the information on those trajectories, the date, time cost, sports mode, and length of the trajectory.

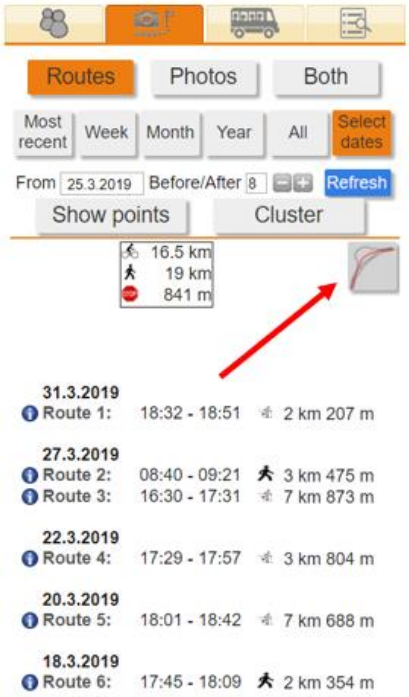

Figure 21. Choosing trajectory cluster by setting time

In Figure 21, we set the time zone to be eight days before or after 25.03.2019. Then the detailed information will be shown and plotted on the map. Above all, we can obtain the trajectories' cluster. Then we can click the button pointed by the red arrow in Figure 21 or the "Calculate Medoid" button on the developed Medoid page.

5.2 Trajectory distance methods

As we have introduced in Section 3, there are eight methods in my work to calculate the trajectory distance. Usually, we can use the Medoid page to compare the time efficiency and similarity on those methods. In Figure 22, it shows we can select the method by the drop-down menu.

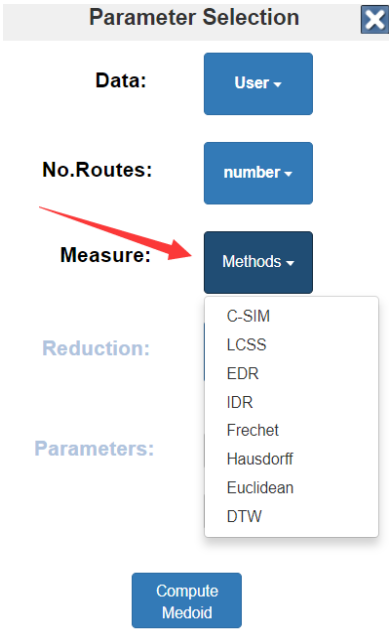

Figure 22. Choosing the trajectory distance method from Medoid page, there are eight options

A different method may lead to different similarity results and efficiencies. In Figure 23, those similarities are in a table of the Medoid page. Each similarity corresponding to the sum of similarities from a trajectory to others in the cluster.

Medoid index: 11
(No.0: Original trajectory)

No.	User Id	Traj Id	Similarity
0	260	1554046355581	0.44
1	260	1515689409383	0.35
2	260	1521561774984	0.44
3	260	1524667468196	0.37
4	260	1532241644762	0.36
5	260	1534178066495	0.32

Figure 23. Similarity table for every trajectory in the cluster. Similarity column means the sum of similarity to all others, medoid has has the max value, for some distance methods, we use distance column to represent the sum of distance to all others, then medoid has the min value

5.3 Trajectory simplification

In Mopsi, we can reduce the number of points in trajectory by using the polygonal approximation. In Figure 24, there is a menu for the user who is going to calculate the Medoid trajectory among a set of trajectories. If user hopes to reduce the number of points in trajectories, they can choose “Reduction” menu and different level from 1 to 5. Level 5 is the highest degree of polygonal approximation that leads to the smallest number of points. The reduced data points are already calculated and stored in the Mopsi server.

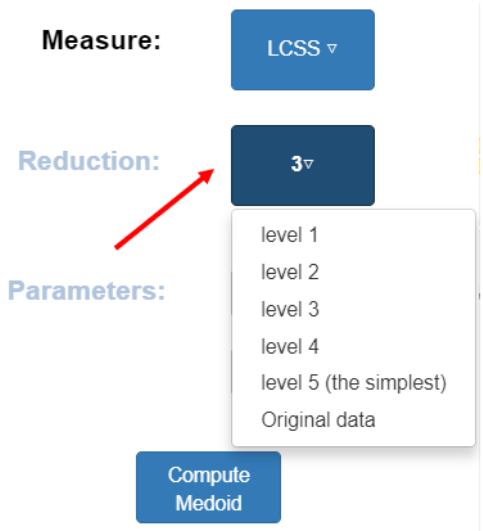

Figure 24. Polygonal approximation option with five levels, level ‘5’ means the simplest trajectory

When we select the “Measure” option on the medoid page, there will appear eight distance methods. We can directly input the required parameter and do the medoid calculation. Most trajectories contain multiple GPS points; maybe thousands of points or even more. However, many of them are not necessary. Meanwhile, it costs a long time in data processing if we have too many points, which will be an obvious drawback for an application such as a webpage. In our Medoid page, it also has the slow-processing problem if we apply those trajectories directly, the user does not hope to wait for a long time until the result shows up. Considering this problem, we adopt the polygonal approximation to reduce the number of GPS points in a trajectory.

In Figure 25, the trajectory has different reduction levels, which leads to slightly different appearances and different numbers of points. Those reduced trajectories take less time to calculate distance compared with the original trajectory, which has 1773 points.

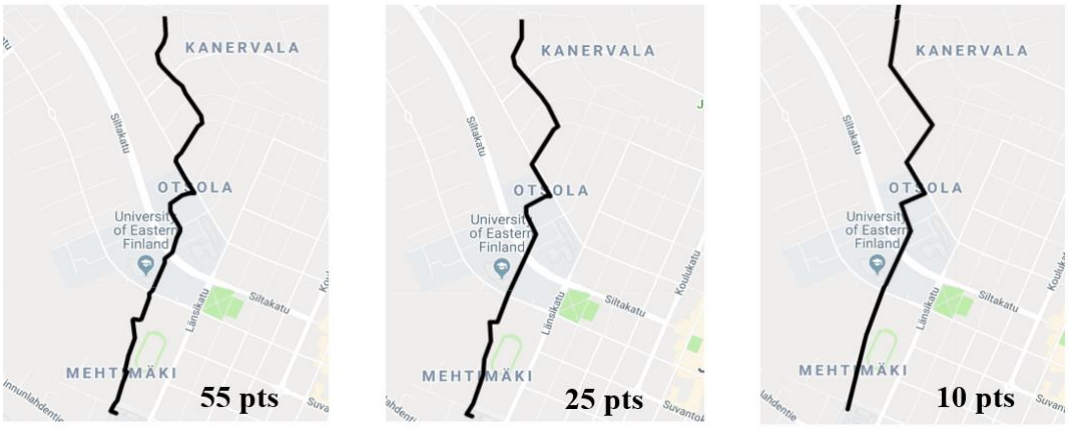

Figure 25. Polygonal approximation at three different levels. Original trajectory has 1773 points. From left to right, the reduced trajectory is at a different level, and the numbers of points are different.

When we apply the polygonal approximation on the distance calculation, there is a significant decrease of time cost. Meanwhile, we use parallel computing to calculate the distances. In Section 6, we will show that in parallel computing, those trajectories applied with polygonal approximation have much less processing time than the original ones. So, the “Reduction” menu can be a good option if the user wants to reduce the time cost. If accuracy is necessary, users can ignore this option and calculate medoid trajectory after choosing the method. Detailed efficiency comparison is given in Section 6.

5.4 The workflow of medoid page

As has been introduced in the previous Section, after we have chosen the trajectories’ cluster and simplification level, we can calculate the medoid trajectory. Figure 26 illustrates the basic process of calculating the medoid trajectory in the Medoid page.

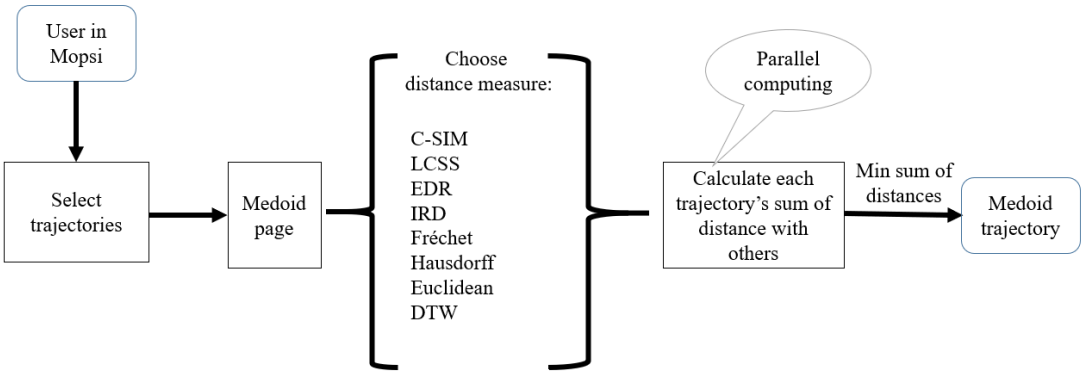

Figure 26. The process of calculating the medoid trajectory on the web page.

5.5 Technology

We have applied several technologies in this Medoid page. For the front-end, it contains JavaScript, Ajax, Google Map API; for the back-end, Java, PHP, and parallel computing. We will introduce parallel computing in detail.

5.5.1 Parallel computing

Parallel computing is the simultaneous use of multiple computer resources to solve a computational problem [22]. The basic idea is to partition the whole task into several tasks. Then each task will be separated as a series of instructions; different processors will process those instructions at the same time. So, the time cost will reduce, which is helpful in the processing and displaying of GPS data. The process is in Figure 27.

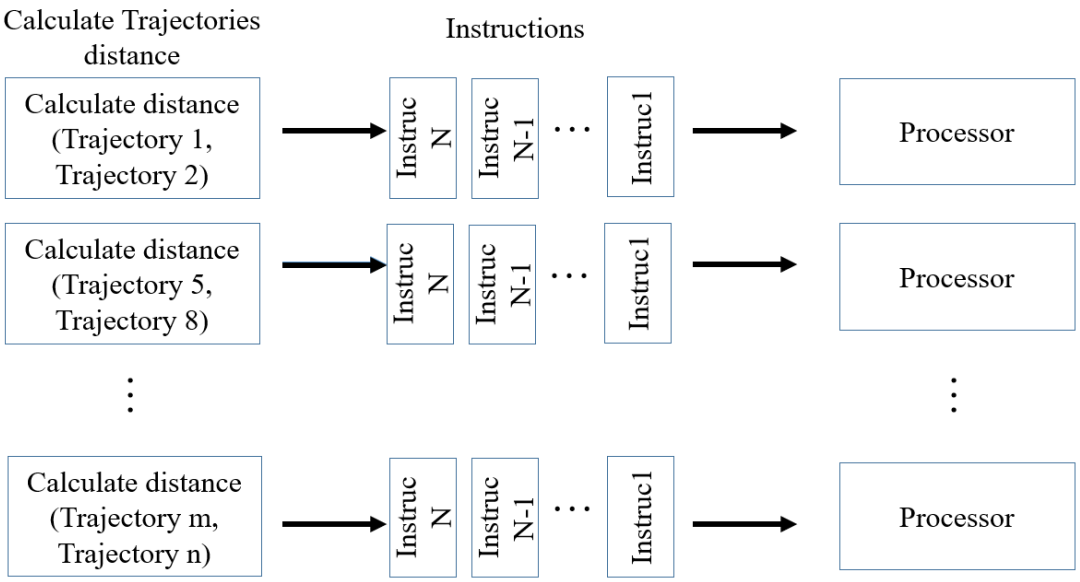

Figure 27. Parallel Computing in trajectory distance.

We use parallel programming in calculating trajectory distances. Assume there are N trajectories; if we hope to find each trajectory's sum of distances with other trajectories, the total times of calculating distances between two trajectories should be the combined value in mathematics, which means take two trajectories from N trajectories without repetition. It should be $(N(N-1)/2)$ times of distance calculation in order to get all similarities for N trajectories. In this thesis, parallel computing has been applied in the Medoid page's server side to reduce calculating time, the related experiment is in Section 6.

In the *operating system*, a thread is the smallest part of processing. For parallel computing, we need to choose the optimal number of threads to utilize computer resource mostly. From [23] we know that for the compute-intensive task such as calculation, the optimal number of threads is $N+1$, N means the number of CPUs. After choosing the number of threads, we can apply parallel computing in trajectory distances.

5.5.2 Ajax

Ajax is “Asynchronous JavaScript and XML” [24]. It is used for creating a fast and dynamic web page by data exchange with the server at the back-end. It can do an asynchronous update, which means it can update part of the web page without loading the whole page, for the traditional web page, only the whole page is reloading, the update can complete. We chose Ajax because it is efficient and more concise operation. The process is: when browser has generated an event, the client will generate an XML HTTP Request object, and send it to the server, when the server receives this, it will process the data and send the response back to the client, the browser process the response data by JavaScript, then update the page content.

The response data is wrapped into JSON (JavaScript Object Notation), “JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format” [25]. It has a concise and clear structure, which is not only easy for reading and writing but convenient for analyzing and generating by machine. It is efficient in network transmission. One example of JSON is: var JSON = {"route_id": 1552544427340,"user_id": 260}. The process is in Figure 28.

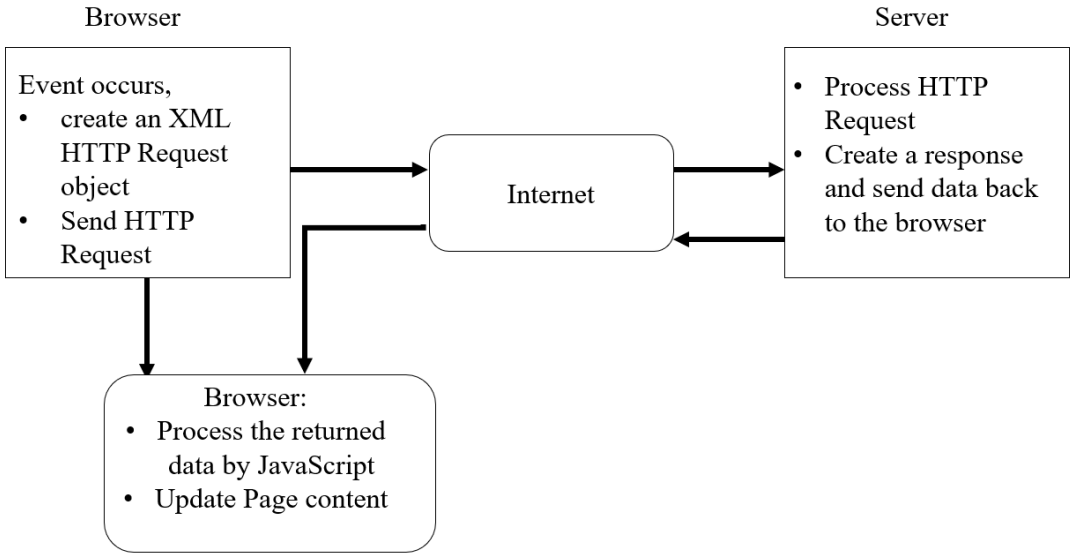

Figure 28. The workflow of the Medoid page

5.5.3 Google Map

Google Maps is a web mapping service developed by Google [11]. There are many APIs provided by Google Maps, the services we used in this thesis is plotting trajectory, customizing marker, line, color and so on. Google Map is helpful in the representation of our research work.

6 Experiments

There are several experiments performed: 1) Parallel computing experiment, 2) Methods' efficiency experiment, 3) Accuracy on cluster representative, 4) K-medoids experiment. All the experiments are performed in MACHENIKE laptop; the software is Windows 10, WinSCP, Spyder, and Chrome. The data are trajectories in Mopsi; each trajectory has a different number of GPS points. We calculate the distances among those chosen trajectories and make further analysis. The data format is the same as Table 1, latitude and longitude are used in the calculation.

6.1 Parallel computing experiment

In this experiment, we try to evaluate the benefit of using parallel computing in the distance calculation. Figure 30 shows the time cost with different number of trajectories when we use parallel computing or not. Since the polygonal approximation method can reduce the number of points in trajectory while keeping their shape, it helps to increase the efficiency in trajectory processing. And it has been already applied in the trajectory visualization and calculation in Mopsi, so we use those reduced trajectories in this experiment to compare the effect of parallel computing and serial computing.

We are going to analyze the processing time for those trajectories in Figure 29. The trajectory with green color and its eleven similar trajectories. Then we find the medoid by using LCSS distance; we have chosen a different number of them to see the time cost on parallel and serial computing.

1	Radu		82%	11.1.2018
2	Radu		81%	25.4.2018
3	Lahari		81%	16.8.2016
4	Radu		81%	13.8.2018
5	Radu		81%	22.7.2018
6	Radu		81%	20.3.2018
7	Lahari		80%	20.8.2016
8	Radu		80%	15.12.2018
9	Lahari		80%	12.5.2016

...

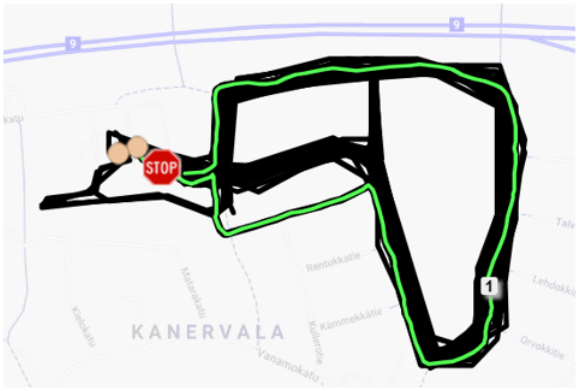

Figure 29. Calculating trajectories' distances by parallel and serial way. There are 12 trajectories (Including the green one on 09/04/2019 recorded by Mopsi user Radu)

In Figure 30, we can find that parallel computing can improve the efficiency of calculations greatly. It is significant and helpful to adopt parallel computing in the trajectory distance calculation.

In my experiment, we have chosen LCSS distance in calculating the medoid among trajectories with different number of trajectories. In Figure 30, it is obvious that by applying parallel computing, which is multi-threading technology in this case, it helps to improve the time efficiency greatly, for example when there are 70 trajectories, parallel computing has decreased by around 58% time cost compared to serial computing. With the increasing of the number of trajectories, the advantage of parallel computing becomes clearer.

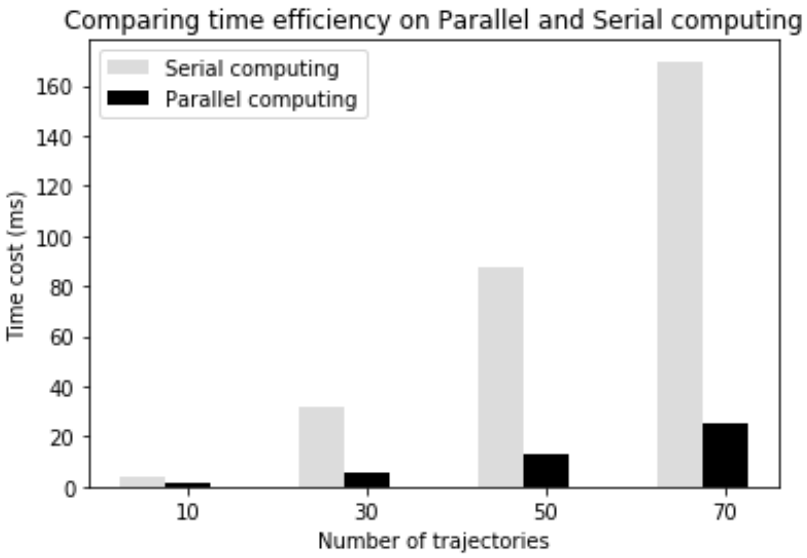

Figure 30. Serial and parallel time cost with a different number of trajectories (milliseconds)

We adopted java language in this part. It uses executor to calculate the distance between two trajectories. Sum class implements the Callable interface, which executes the parallel work inside the *call()* method. We initialize $N(N-1)/2$. Sum classes are processed by *m* threads. The pseudocode is in the Appendix.

6.2 Efficiency experiment

As has been analyzed previously, parallel computing has a significant improvement in the distance calculation. So, in this experiment, we will user parallel computing to compare the differences among those distance methods. Please check Figure 31 below. Since the simplification of the reduced trajectory varies from level 1 to 5 in increasing order, we choose level 3 in this experiment as the representation of the reducing level. Then we calculate the time cost when calculating the medoid trajectory with different methods on five trajectories. We get those trajectories from the Mopsi user Pasi, they are five similar trajectories which are also in Figure 33. The detailed data description is in Table 2. We plot the experiment result as a bar chart in Figure 31.

Table 2. Data description.

Trajectory's index	**Starting timestamp**	**Number of GPS points**
A	1495599253572	3199
B	1508387097655	2709
C	1509513965456	2917
D	1510807697590	2709
E	1514438747265	3429

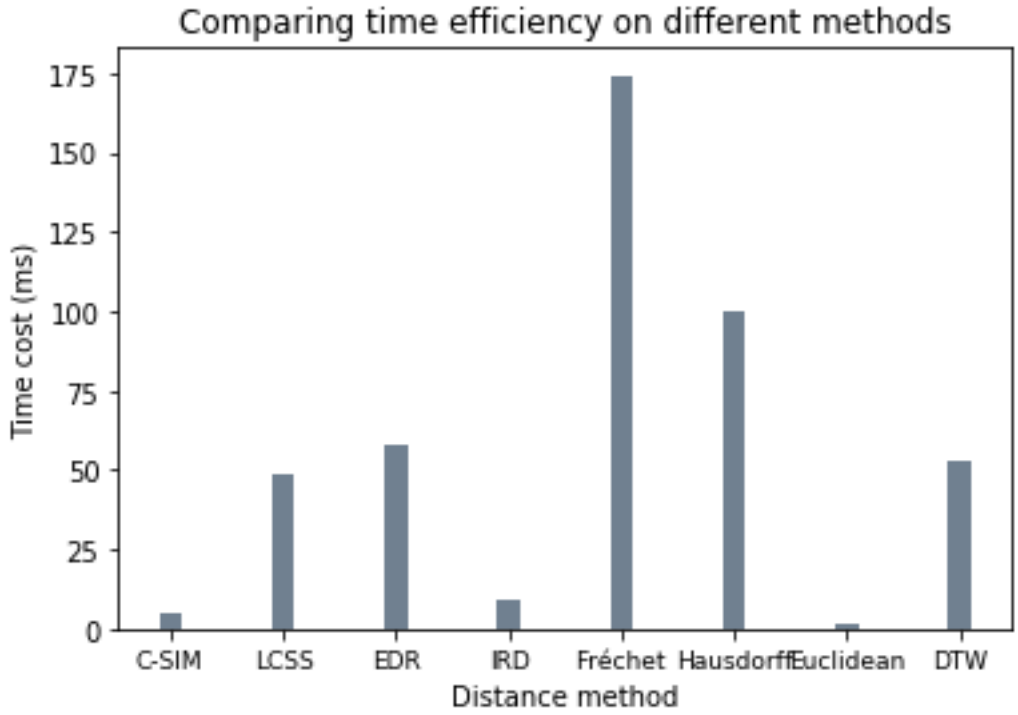

Figure 31. Time cost (millisecond) of the medoid calculation with the different methods for five trajectories. Parameter L = 45 m and ε = 90 m

Euclidean distance is fastest to compute, followed closely by C-SIM and IRD. Then, at a different order of magnitude: LCSS, EDR, DTW, Hausdorff and Frechet.

6.3 Accuracy with short segments

In this Section, we will analyze the accuracy of using different methods to calculate the representative for trajectories cluster. The University of Eastern Finland held the *Average GSS segments contest* [3]. We have 100 sets that contain a different number of trajectories, and each set has the ground truth of the representative. The accuracy of similarity to ground truth is using a parameter independent varient of C-SIM, which is entitled Hierarchical Cell Similarity (HC-SIM) [4]. It will be documented in a paper: P Fränti and R. Mariescu-Istodor, Averaging GPS segments: challenge Manuscript 2019. (submitted)

We next consider the following methods for calculating the representative:

[3] http://cs.uef.fi/sipu/segments
[4] http://cs.uef.fi/sipu/segments/results.html

- Simple averaging heuristic
- Shortest trajectory [26]
- Medoid with any of the distance measures from [5]
- Jiawei's method
- Combining Jiawei's idea with medoid

The *simple averaging heuristic* works as follows. It first makes every trajectory in the set to have the same length by adding points to the trajectories. For trajectories *A* and *B*, *A* is composed of (a_1, a_2 … a_n), *B* is composed of (b_1, b_2 ... b_m). If *A* has more points than *B*, we add those extra points in *A* to the end of *B*. So, we get *n* pairs from *A* and *B*, such as (a_i, b_i). Since these points are geographical coordinates, we convert them to UTM coordinates. We then compute the average of each pair, so that we have *n* averages as the result. Finally, we convert them back to longitude and latitude. In Figure 32, we can see the description of this method. The result is composed of (avg_1, avg_2 … avg_n). Comparing to the ground truth, the method has an accuracy of 53.74%.

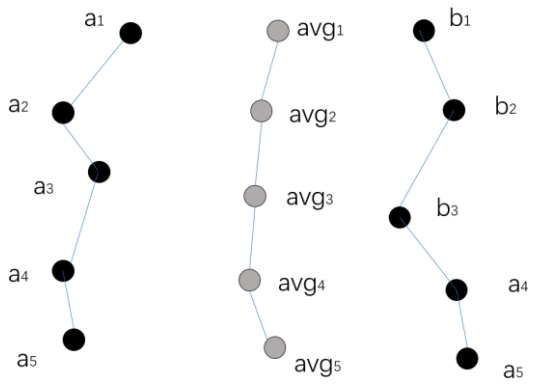

Figure 32. Trajectory *A* is on the left has 5 points; trajectory *B* on the right has 3 points. We add point a_4, and a_5 from *A* to the end of trajectory *B*. The average points make up the averaging trajectory which is in the middle.

The second method (shortest trajectory) is given by Fathi and Krumm from [26]. They take the shortest trajectory from a set as the representative. This is the simplest method available. When applying this method, we get the accuracy of 54.16%. It is slightly better than the averaging heuristic.

The third method is *Medoid* using any of the distance methods introduced in Section 2. This results in seven different variants in total. The best result is obtained using IRD distance measure, which achieves accuracy of 60.82%.

The fourth method (*Jiawei's method*) is an invention within the Machine learning group, recently proposed by Jiawei Yang. Since it is not published as a paper, I describe it only briefly without all the details. The method partitions the points into three sets: *source*, *median* and *destination* points, denoted as *S*, *M*, and *D* respectively. The grouping is done by clustering the points. The representative of the set is then constructed using three sampling points: the arithmetic averages of the three sets. These three points also form a triangle in the space. If the cosine value formed by $\overrightarrow{SM}$ and $\overrightarrow{SD}$ is bigger than 0.994, return trajectory formed by S, M, D. Otherwise, the median trajectory is used instead. The way we calculate median trajectory: firstly, we remove the trajectory with maximum number of points, then if the number of trajectories is smaller than five, the median is the shortest trajectory in the set, otherwise it is the trajectory with least points. This method reaches accuracy of 67.2%.

The fifth method is a hybrid of Jiawei's method and Medoid (*Method 5*). Different from the fourth method, if the cosine value is bigger than 0.994, we use the medoid as the representative instead of the median. We can apply any distance method in the calculation of medoid. The detailed results are in Table 3.

Table 3. Using different methods to get representative from trajectory set, compare the result with ground truth, L= $\mathcal{E}$=10%

	C-SIM	LCSS	EDR	IRD	Fréchet	Hausdorff	Euclidean	DTW
Medoid	57.6%	58.9%	58.1%	60.82%	59.17%	60.36%	57.58%	56.64%
Method 5	67.2%	67.1%	67.4%	67.0%	68.1%	67.9%	67.7%	67.0%
Jiawei's method	67.2%							

simple averaging heuristic	53.74%
Shortest trajectory	54.16%

In Table 3, we see that the simple averaging heuristic has the lowest accuracy. Then the shortest trajectory has the second lowest accuracy. Among different Medoid variants with different distance methods, IRD has the highest accuracy compared with the other distance methods: 60.82%. DTW has the lowest accuracy: 56.64%. Medoid has better accuracy compared with the simple averaging heuristic and the shortest trajectory method.

Jiawei's method, on average, behaves better than the first three methods. Its accuracy is 67.2 %, which is much higher than that of Medoid, but in Method 5, we will notice that EDR, Fréchet, Hausdorff and Euclidean have higher accuracy than Jiawei's method. With Fréchet, it leads to the highest accuracy of 68.1%, which is slightly better than Jiawei's method. Above all, Method 5 has the highest accuracy when we apply Fréchet, and this method has the highest accuracy on average.

When each trajectory has a small number of points, such as Set 1 and Set 89 in Figure 35 (first line and second line), Jiawei's method (78% and 86%) behaves more accurate than Medoid (35% and 56%). Method 5 provides the same result as Jiawei's method. Since we need to compute the medoid in Method 5 instead of the median in Jiawei's method, it takes more time to calculate medoid ($O(n^2)$) than median $O(n)$, where n is the number of trajectories in the set. So Method 5 is slower than Jiawei's method.

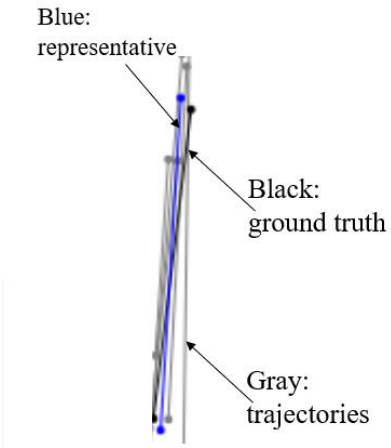

Figure 33. Example result of Jiawei's method on Set 42. The blue line is the representative; gray ones are the elements in the set; the black one is the ground truth.

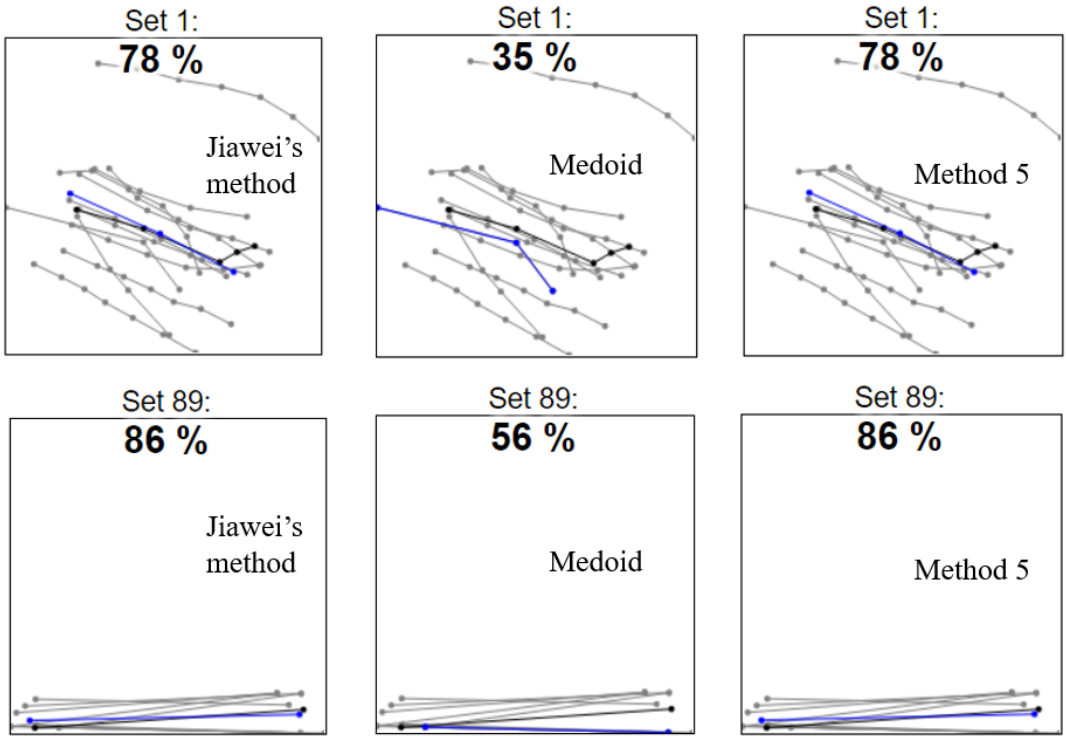

Figure 34. Set 1 and Set 89 have only few points in each trajectory. Jiawei's method and Method 5 have higher accuracy (78% and 86%) than Medoid in both sets. Method 5 and Medoid applies Euclidean distance

When we have multiple points in each trajectory (Set 16), Medoid (73%) and Method 5 (73%) gives a better result than Jiawei's method (37%). An example is given in Figure 35. So, in this case, Method 5 combines the advantages of Jiawei's method and Medoid;

it behaves better than Medoid when there are only few trajectory points. Meanwhile, it behaves better than Jiawei's method when we have many points in a trajectory.

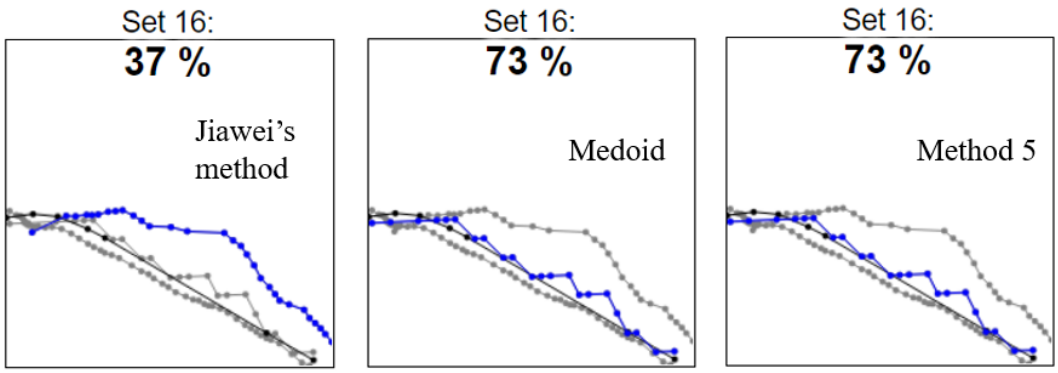

Figure 35. Set 16 has around 25 points per trajectory. In this case, Medoid or Method 5 is better.

6.4 Accuracy with complete trajectories

Now let's see the accuracy with complete trajectories. Since we do not have ground truth for these data, we just calculate the average of the sum of distances to all others in the cluster to evaluate the goodness of the representative; the smaller, the better. We denote this as *average distance*. When we apply medoid on Mopsi trajectories, for example, we find the representative from five trajectories of user Radu. The result is shown in Figure 36. The brown curve indicates the representative.

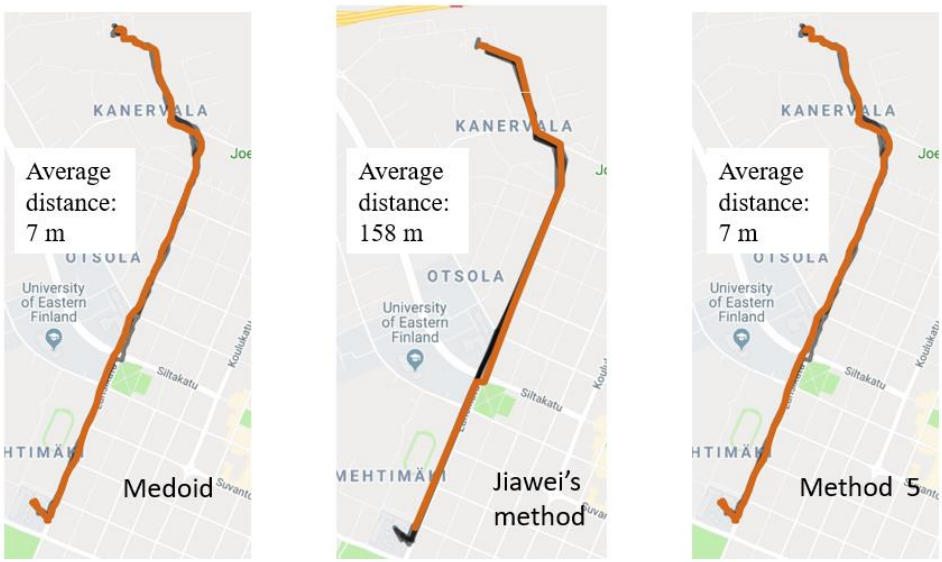

Figure 36. Apply Medoid, Jiawei's method and Method 5 to Mopsi trajectories set, average distances among the representative and others are 7 m, 158 m, 7 m respectively

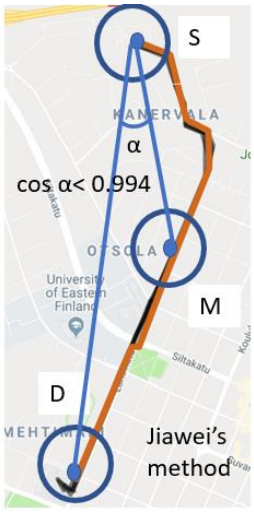

Figure 37. Angle α is formed by $\overrightarrow{SM}$ and $\overrightarrow{SD}$, cos α is 0.990 , so the median trajectory is chosen

Figure 37 illustrates the principle of calculating the representative by Jiawei's method, where the median trajectory is the trajectory with the middle number of points among all trajectories in the set. In reality, the angle α is more likely to be bigger than 5°, which means cos α is more likely to be smaller than 0.994. So in Jiawei's method, the median is more likely to be chosen as the average, but as seen in experiments in Section 6.3 and Figure 36, we know that Medoid has better accuracy and performance in the representative calculation. So method 5 is a good replacement for Jiawei's method in calculating representative. Meanwhile, since it uses [S, M, D] as the representative when the cos α is bigger than 0.994, it will save time compared with Medoid, which needs to do similarity calculation for each trajectory (O (1) vs. O (N). N is number of trajectories)

In Jiawei's method, the average distance is around 158 m, which is much more than that of Medoid and Method 5 (7 m). Meanwhile, let's assume when α is very small so that Jiawei's method decides to use [S, M, D] as the representative, but if those points which are not S, M or D, they can form very complicated shape (such as circle shape exist between S and M or M and D), they will be ignored. So, Jiawei's method and Method 5 are not suitable for representative in complicated shaped trajectories.

Above all, for those set that trajectories do not have big changes in directions (such as Figure 34 and Figure 35), Method 5 is a good option. When the trajectories have a complex shape, or big change in direction, Medoid is a better option.

7 Conclusions

According to the experiments, compared with other methods, IRD can produce the highest accuracy (60.82%) in the calculation of medoid compared with LCSS, EDR, Fréchet, Hausdorff, Euclidean, and DTW. Meanwhile, it is faster than methods such as C-SIM, and LCSS. Although the Euclidean method is faster than IRD, it has lower accuracy. So, in trajectory distance measure and medoid calculation, IRD distance is a good option.

When we apply parallel computing in the trajectory distance calculation, the time efficiency is greatly improved. For example, when we calculate the distances among 70 trajectories, the time cost has reduced from 170 milliseconds to 26 milliseconds, which is 5.5 times faster. So, in practice, parallel computing (multi-threading in this case) is suggested in those trajectory calculation work.

We have also used Jiawei's method, Medoid, and Method 5 (Proposed by Pasi Fränti) to calculate the representative of a trajectory set. By comparing the accuracy and effect among those methods, we find that different methods have different behavior depending on the trajectory. Medoid behaves better in case of Mopsi trajectories; it is suitable for complicated shaped trajectories set. Jiawei's method is good for trajectories with only few points. Method 5 combines the advantages of Medoid and Jiawei's method, so it has better average behavior with different kinds of trajectories whose direction does not change greatly. Otherwise, Medoid is better.

In this thesis, we have introduced a Mopsi page to calculate the representative trajectory. The user can change the method and input data. This tool is very useful when the user wants to analyze the effect of distance methods. The similarities and trajectories will be visualized on the page, which helps the user to understand data and make comparisons.

8 References

[1] Chen, M., Xu, M., & Fränti, P. (2012). A fast O(n) multiresolution polygonal approximation algorithm for GPS trajectory simplification. *IEEE Transactions on Image Processing*, 21(5), 2770-2785.

[2] Lee, J. G., Han, J., & Whang, K. Y. (2007, June). Trajectory clustering: a partition-and-group framework. *In Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (pp. 593-604).

[3] Gaffney, S., & Smyth, P. (1999, August). Trajectory clustering with mixtures of regression models. In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 63-72).

[4] Mariescu-Istodor, R. & Fränti, P. (2018). Detecting user actions in location-based systems. *Int. Conf. on Location Based Services (LBS),* Adjunt proceedings, Zürich, Switzerland, 1-6, January 2018.

[5] Mariescu-Istodor, R., & Fränti, P. (2017). Grid-based method for GPS route analysis for retrieval. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 3(3), 8.

[6] Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., ... & Zhou, Z. H. (2008). Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1), 1-37.

[7] Rokach, L., & Maimon, O. (2005). Clustering methods. In *Data mining and knowledge discovery handbook* (pp. 321-352). Springer, Boston, MA.

[8] Fränti, P. (2018). Efficiency of random swap clustering. *Journal of Big Data*, 5(13), 1-29.

[9] Struyf, A., Hubert, M., & Rousseeuw, P. (1997). Clustering in an object-oriented environment. *Journal of Statistical Software*, 1(4), 1-30.

[10] Laan, M., Pollard, K., & Bryan, J. (2003). A new partitioning around medoids algorithms. *Journal of Statistical Computation and Simulation*, 73(8), 575-584.

[11] Verma, P., & Bhatia, J. S. (2013). Design and development of GPS-GSM based tracking system with Google map-based monitoring. *International Journal of Computer Science, Engineering and Applications*, 3(3), 33.

[12] Mariescu-Istodor, R. (2017) *Efficient management and search of GPS routes*. Ph.D. thesis, University of Eastern Finland.

[13] Ichiye, T., & Karplus, M. (1991). *Collective motions in proteins: a covariance analysis of atomic fluctuations in molecular dynamics and normal mode simulations. Proteins: Structure, Function, and Bioinformatics*, 11(3), 205-217.

[14] Zheng, Y., & Zhou, X. (Eds.). (2011). *Computing with spatial trajectories*. Springer Science & Business Media.

[15] Chen, L., Özsu, M. T., & Oria, V. (2005, June). Robust and fast similarity search for moving object trajectories. *In Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (pp. 491-502).

[16]. Sutinen, E., & Tarhio, J. (1996, June). Filtration with q-samples in approximate string matching. *In Annual Symposium on Combinatorial Pattern Matching* (pp. 50-63). Springer, Berlin, Heidelberg.

[17] Wang, H., Su, H., Zheng, K., Sadiq, S., & Zhou, X. (2013, January). An effectiveness study on trajectory similarity measures. *In Proceedings of the Twenty-Fourth Australasian Database Conference-Volume* 137 (pp. 13-22). Australian Computer Society, Inc.

[18] Trasarti, R., Guidotti, R., Monreale, A., & Giannotti, F. (2017). Myway: Location prediction via mobility profiling. Information Systems, 64, 350-367.

[19] El-Rabbany, A. (2002). *Introduction to GPS: the global positioning system*. Artech House.

[20]. Eiter, T., & Mannila, H. (1994). Computing discrete Fréchet distance. Tech. Report CD-TR 94/64, Information Systems Department, Technical University of Vienna.

[21] Huttenlocher, D. P., Rucklidge, W. J., & Klanderman, G. A. (1992, June). Comparing images using the Hausdorff distance under translation. *In Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 654-656).

[22] Barney, B. (2010). Introduction to parallel computing. *Lawrence Livermore National Laboratory*, *6*(13), 10.

[23] Goetz, B., Peierls, T., Lea, D., Bloch, J., Bowbeer, J., & Holmes, D. (2006). *Java concurrency in practice*. Pearson Education.

[24] Klein, J., & Spector, L. (2007, July). Unwitting distributed genetic programming via asynchronous JavaScript and XML. *In Proceedings of the 9th annual conference on Genetic and evolutionary computation (pp. 1628-1635).* ACM.

[25] Crockford, D. (2006). The application/JSON media type for javascript object notation (JSON) (No. RFC 4627).

[26]. Fathi, A., & Krumm, J. (2010, September). Detecting road intersections from GPS traces. *In International conference on geographic information science* (pp. 56-69). Springer, Berlin, Heidelberg.

[27] Hofmann-Wellenhof, B., Lichtenegger, H., & Collins, J. (2012). *Global positioning system: theory and practice*. Springer Science & Business Media.

[28] Zheng, Y., Liu, Y., Yuan, J., & Xie, X. (2011, September). Urban computing with taxicabs. *In Proceedings of the 13th international conference on Ubiquitous computing* (pp. 89-98). ACM.

[29] Matthew, N., & Stones, R. (2008). *Beginning Linux programming*. John Wiley & Sons.

[30] Robinson, M. T. (1990). The temporal development of collision cascades in the binary-collision approximation. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 48(1-4), 408-413.

[31] "NGA Geomatics - WGS 84". *earth-info.nga.mil*. Retrieved 2019-03-19.

[32] Snyder, J. P., & Voxland, P. M. (1989). *An album of map projections* (No. 1453). US Government Printing Office.

[33] Blu, T., Thévenaz, P., & Unser, M. (2004). Linear interpolation revitalized. *IEEE Transactions on Image Processing*, 13(5), 710-719.

[34] Besse, P. C., Guillouet, B., Loubes, J. M., & Royer, F. (2016). Review and perspective for distance-based clustering of vehicle trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 17(11), 3306-3317.

[35] Huttenlocher, D. P., & Rucklidge, W. J. (1992). *A multi-resolution technique for comparing images using the Hausdorff distance*. Cornell University.

[36] Orwant, J., Hietaniemi, J., & Macdonald, J. (1999). *Mastering Algorithms with Perl*. O'Reilly Media, Inc.

[37]. Arora, P., & Varshney, S. (2016). Analysis of k-means and k-medoids algorithm for big data. *Procedia Computer Science*, 78, 507-512.

[38] Park, H. S., & Jun, C. H. (2009). A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, 36(2), 3336-3341.

[39] Schubert, E., & Rousseeuw, P. J. (2018). Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. *arXiv preprint arXiv:*1810.05691.

[40] Mariescu-Istodor, R., & Fränti, P. (2016, November). Gesture Input for GPS Route Search. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)* (pp. 439-449). LNCS 10029, 439-449, Springer, Cham.

[41]. Waga, K., Tabarcea, A., Chen, M., & Fränti, P. (2012, October). Detecting movement type by route segmentation and classification. In *8th International Conference on Collaborative computing: networking, applications and worksharing (CollaborateCom)* (pp. 508-513). IEEE.

[42]. Waga, K., Tabarcea, A., Mariescu-Istodor, R., & Fränti, P. (2013, May). Real Time Access to Multiple GPS Tracks. *Int. Conf. on Web Information Systems & Technologies (WEBIST'13),* Aachen, Germany, 293-299

[43] Ho, A. D., & Yu, C. C. (2015). Descriptive statistics for modern test score distributions: Skewness, kurtosis, discreteness, and ceiling effects. *Educational and Psychological Measurement*, 75(3), 365-388.

9 Appendix (Code for parallel computing)

```
import java.util.*;
import java.util.concurrent.*;
import static java.util.Arrays.asList;

public class Sums {

    static class Sum implements Callable<Long> {
        private final long from;
        private final long to;
            Sum(long from, long to) {
            this.from = from;
            this.to = to;
        }

        @Override
        public Long call() {
            long acc = 0;
            for (long i = from; i <= to; i++) {
                acc = acc + i;
            }
            return acc;
        }
    }

    public static void main(String[] args) throws
Exception {

        ExecutorService executor =
Executors.newFixedThreadPool(2);
        List <Future<Long>> results =
executor.invokeAll(asList(
            new Sum(0, 10), new Sum(100, 1_000), new
Sum(10_000, 1_000_000)
        ));
        executor.shutdown();

        for (Future<Long> result : results) {
            System.out.println(result.get());
        }
    }
}
```