Evaluating distance measures for trajectories in the mobile setting

Nikolaos Larios NLARIOS@DI.UOA.GR
University of Athens

Christos Mitatakis CMITATAKIS@DI.UOA.GR
University of Athens

Vana Kalogeraki VANA@AUEB.GR
Athens University of Economics and Business

Dimitrios Gunopulos DG@DI.UOA.GR
University of Athens

Abstract

Mobile devices, such as smartphones allow us to use computationally expensive algorithms and techniques. In this paper, we study algorithms in order to solve the problem of finding the most similar trajectory within a number of trajectories. We built a framework that enables the user to compare a trajectory Q with trajectories that have been generated and stored on mobile devices. The system returns to the user the most similar trajectory based on the algorithm that has been selected. The algorithms for the measurement of the trajectory similarity have been implemented for mobile devices running Android OS. We evaluate our algorithms with real geospatial data.

1. Introduction

The study of the similarity between trajectories is important in a plethora of application domains (e.g. Traffic management, Video analysis, Molecular Design, Similarity of Meteorological Phenomena etc.). More specifically, the trajectory similarity between moving objects is useful for applications in intelligent traffic systems, such as traffic navigation and traffic prediction, both of which require mining of past trajectories patterns, etc. The identification of similarities between moving objects is a challenging task, since not only their locations change but also because their speed and semantic features vary. Mobile devices and smartphones, in particular are rapidly emerging as a dominant computing and sensing platform. They are equipped with a variety of sensors such as GPS, cameras and accelerometers. This gives birth to several unique opportunities for data collection and analysis such as finding trajectory similarities between trajectories generated and stored on distributed smartphones. We developed a framework that allows users to find similar trajectories in a distributed environment. Our framework requires the participation of a number of users in order to find a solution of a query. In order to specify the way in which our framework works, every query is assigned by a user of our system and it is addressed to all the available users of our platform. The goal of our platform is to generate and store trajectory data on mobile devices (smartphones) and compare a query trace Q against the crowd of trajectories that is distributed on a large number of mobile devices.

Proceedings of the 2^{nd} International Workshop on Mining Urban Data, Lille, France, 2015. Copyright ©2015 for this paper by its authors. Copying permitted for private and academic purposes.

1.1. Related Work

There are many methods and techniques related to searching for similar moving object trajectories. Some previous methods were based on Euclidean distance space, but this is not suitable for road network space because is difficult to apply the distance of Euclidean space to road network space. Other methods considered only spatial similarity without considering temporal similarity to search for similar moving object trajectories. The methods that interest us the most are those that are used for measurement of the spatio-temporal similarity between trajectories. Although in this work we do not consider the privacy issues that come up when data from users become available in the system, there is work in the area that can be leveraged in a real system. In "Select-Organize-Anonymize"(Giorgos Poulis, 2013), the authors find similar trajectories, by using Z-ordering and data projections on subtrajectories. Then they organize the selected trajectories into clusters which they anonymize after that. Another work that the researchers propose a method to compare trajectories of moving object is in ”Shapes based trajectory queries for moving objects”.(Lin & Su, 2005) In this work, the writers introduce a new distance function and the evaluate the similarity measurement by using a grid representation to describe trajectories. The most similar work is the paper Crowdsourced Trace Similarity with Smartphones paper (Zeinalipour-Yazti et al., 2013) where the writers try to solve efficiently the problem of comparing a query trace against a number of traces generated by smartphones. They developed the SmartTrace+ framework which consists a distributed data storage model that trajectories are stored and top-K query processing algorithms that exploit trajectory similarity measures, elastic to spatial and temporal noise.

1.2. Our Contributions

In this paper, we build a distributed mobile platform that uses algorithms to search for trajectory similarity among a number of trajectories that are stored in distributed mobile devices. The trajectories which are stored have been collected by monitoring the movement of the mobile devices. The users choose whether their movement will be recorded by the application. So our platform consists of two parts, the one is where the users gather the trajectory data, so that can be created a distributed database where the trajectories are stored.

The second contribution of our platform is the mechanism that given a query trajectory Q, we want to find any trajectories of the mobile devices using our platform that follow a motion similar to Q. We initiate an experimental evaluation of three different trajectory similarity measures, namely Dynamic Time Warping (DTW), Longest Common Subsequence (LCSS), and Fréchet distance. Intuitively, Dynamic Time Warping (DTW) is the equivalent of the L_2 distance when stretching of the trajectories is allowed since DTW takes into account the individual differences of all matched points in the stretched sequences. Similarly, Longest Common SubSequence (LCSS) can be thought of as the equivalent of L_0 since it counts how many elements are the same. Completing the analogy, the Fréchet distance is equivalent to L_{inf} since it considers the maximum of the individual differences of the matches in the stretched sequences. Here we perform an evaluation on their relative accuracy and performance.

2. Problem Definition

Trajectory: Trajectory or trace of a moving object is a set of consecutive positions in space as a function of time. Due to limitations on the acquisition and storing of data, is very difficult and expensive to accurately record an entire trajectory. In spatio-temporal databases a trajectory is represented as a set of discrete samples of the positions of the moving object of the form (x, y, t), i.e. a sequence of ordered pairs (x, y), each characterized by a timestamp. As described above our goal is to develop a distributed platform for spatio-temporal similarity search between trajectories generated and stored on distributed mobile devices. Specifically, given a query trajectory Q, we want to find any trajectories of the mobile devices using our platform that follow a motion similar to Q.

The mobile application should be able to store locally the trajectories that follow the mobile device, to receive a query Q from another mobile device, through the server of the platform, to evaluate if the mobile device has followed a similar trajectory to Q and return its result back to the specific user through the server again. The platform should be able to compare a query trajectory Q that is submitted by a user, simultaneously against a number of trajectories that is stored on a large crowd of mobile devices. The most similar trajectory of every user with the query Q should be received and managed from the server application ensuring the privacy of the users of the platform. Then the server finds the most similar trajectory within the results that have been collected and sends it back to the user who submitted the query Q. Our approach exploits the fact that nowadays, mobile devices have exceptional computational power and storing capabilities. Running the algorithms in a distributed mobile environment grants our platform the necessary performance and scalability in order to support a great number of users and spatio-temporal data. Moreover, having the trajectories stored in the mobile devices the privacy of the users is protected. The definition of the term similarity between trajectories may vary between different problem cases. For example in most cases the most similar trajectory should be determined by using not only the spatial shape of the trajectories but also their evolution in time. In other cases the timing of the recording of the trajectory or the time intervals between points of interest that are defined by the query may be of greater importance.

3. System Overview

The system has two basic functionalities. The first one consists from the monitoring procedure, where the user can monitor his movement. With this procedure, the users trajectories are stored in the mobiles database creating the necessary data. The second functionality contains the query submission from one user and the search for results, using a selected algorithm, in the devices registered in the system. The architecture of our system is described by figure 1.

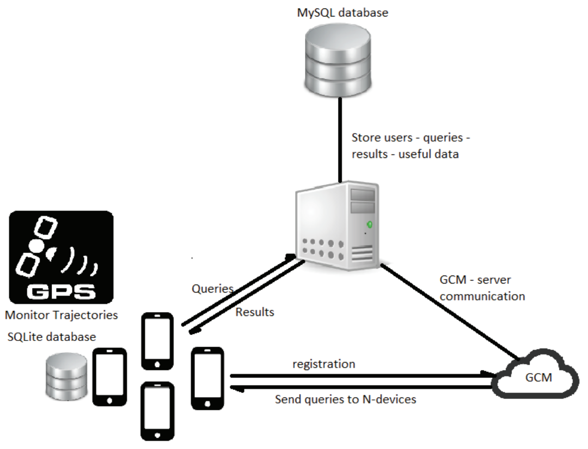
MySQL database
Store users - queries - results - useful data
GPS
Monitor Trajectories
SQLite database
Queries
Results
GCM - server communication
registration
GCM
Send queries to N-devices

Figure 1. System's architecture

3.1. System Components

The goal of the system is to record and store efficient (in time and resources) large amounts of data from N mobile devices and support the communication between them and a main server in order to send queries on the data recorded and send their results. The way that data are stored must be defined, i.e. the structures that will be used in mobile devices and in the server for data storage. Our system consists by the following: A main application server, which is responsible for receiving and forwarding queries between mobile devices using the application. Furthermore, it is responsible for the registration of mobile devices in the system. This application is hosted on an Ubuntu server. As mentioned above in the system will be registered N mobile devices using the application's mobile platform which is responsible for data recording, sending queries, receiving and handling queries other devices and send data if the query result was successful.

3.2. Users

The system consists of N users and those users participate in it through their devices (mobile, tablets, etc.). For the efficient operation and acceptable results of the system, it requires a sufficient number of users. The more users register to our platform the precision of the system will be increased due to the available set of data for each geographic area and time. User data are generated by the sensors of the devices and are stored locally on each device. Since users are not constantly connected to the system due to intermittent connectivity for their mobile devices respectively no data will be offered to our system without their agreement.

3.3. Data Storage

Each device (mobile, tablet, etc.) can produce tuples of data through its sensors (eg GPS sensors, accelerometer, camera, microphone, etc.).The data that are stored in the mobile devices corresponds to the trajectories that have been performed by the device while the user have selected to record his movement.

The form and amount of data depend on the application of the general form to is <Id, Latitude, Longitude, Timestamp>where:

- The Id refers to the unique code that corresponds to the current trajectory that is recorded. Every time the user selects to monitor his movement, a new id is initialized which define the recording.
- The Latitude and Longitude determine the geographical location of the tuple.
- The Timestamp refers to the time of recording the current tuple.

Examples of data form is: <1, 23.32134, 37.566643, 2014-10-24 16:52:01>

Every trajectory consists of a number of tuples like the example above. Each tuple is recorded periodically every 2 seconds. This period can be changed accordingly to our preferences or the storage capabilities of the device. The trajectories that are recorded are stored in the devices internal memory. For the storage of data in the mobile devices is used SQLite which is the default SQL database engine for android powered devices.

3.4. Querying component

The Android application we have developed is responsible for the collection and storing of trajectory data by recording the movement of the mobile devices. Moreover, the algorithms that we use for the measurement of trajectory similarity have been implemented in the Android application.

A query Q when is formed by a user of our platform is forwarded by the server to other registered mobile devices. When a mobile device receives a query it runs the selected algorithm and searches the data that has been collected for the most similar trajectory, according to the selected algorithm. Let $m_1, m_2, ..., m_i$ denotes a set of i mobile devices. The server sends the query Q to this set of mobile devices where Q is a sequence of points $\{p_1, p_2, ..., p_j\}$ that describes a trajectory. The application compare the trajectory Q with each trajectory that is stored in the database of the device (e.g. $\{T_1, T_2, T_3, ..., T_k\}$), in order to find the most similar one. All devices that have a result sends their answers to the server.

3.5. Server

The server of our platform is the middle-ware responsible for the communication between the mobile devices and stores vital data in its database. The server application forwards the queries of the users to the mobile devices which are registered to our platform. Then it receives the results of each device and the best one, according to the algorithm's results, is sent back to the user who submitted the Query.

4. Trajectory Similarity Measures

In this section we review the trajectory similarity algorithms we use in the system to find the most similar trajectories to the query. **Dynamic Time Warping:** The Dynamic Time Warping (DTW) (Donald J. Berndt, 1994) is an algorithm for measuring similarity between two temporal sequences which may vary in time or speed. The DTW is widely used in the comparison of time series, appropriately aligns the sequence of points of the two rails, so that the total distance as a sum of individual distances is minimized.

$$DTW(P_{1..n}, Q_{1..n}) = \\ |P_n - Q_n| + min \begin{cases} DTW(P_{1..n-1}, Q_{1..m-1}) \\ DTW(P_{1..n-1}, Q_{1..m}) \\ DTW(P_{1..n}, Q_{1..m-1}) \end{cases}$$

where P1 .. n-1 the subsequence P1 .. n that include elements (points) for time periods of 1 up to n-1.

Longest Common SubSequence: The Longest Common SubSequence (LCSS) (Michail Vlachos, 2002)problem is to find the longest subsequence common to all sequences in a set of sequences (often just two). The LCSS algorithm, using dynamic programming fits best points of the two trajectories based on a tolerance parameter time and a tolerance parameter space ε. It considers that the points do not exceed the tolerance parameters fit and attaches similarity value equal to 1. If they do not match they are assigned the value 0. Specifically, the LCSS distance between two real-valued sequences S1 and S2 of length m and n respectively is computed as follows:

$$Lcss(P_{\delta,s}(S_1, S_2)) = \\ \begin{cases} 0, if \quad n = 0 \quad \text{or} \quad m = 0 \\ 1 + Lcss_{\delta,s}(HEAD(S_1), HEAD(S_2)) \\ max(Lcss_{\delta,s}(HEAD(S_1), S_2), \\ Lcss_{\delta,s}(S_1, HEAD(S_2))) \\ otherwise \end{cases}$$

where HEAD(S1) is the subsequence $[S_{1,1}, S_{1,2}, ..., S_{1,m-1}]$ and δ is an integer that controls the maximum distance in the time axis between two matched elements and is a real number 0<ε<1 that controls the maximum distance that two elements are allowed to have to be considered matched. One drawback of this measurement is that it ignores distance gaps between subsequences, that allows to obtain some points of the track when alignment. Consequently, it can lead to inaccuracies in the similarity analysis.

5. Fréchet Distance

Given two curves, A, B in a metric space, the Fréchet distance, $d_F(A, B)$ is defined as

$$d_F(A, B) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \{d(A((t)), B((t)))\}$$

where , range over all monotone reparameterizations and d(,) represents the Euclidean distance, and inf is the infimum. The discrete Fréchet distance d_F between two polygonal curves $a : [0, m] \to R^k$ and $b : [0, n] \to R^k$ is defined as:

$$d_F(a, b) = \\ \min_{\substack{\sigma:[1:m+n]\to[0:m], \\ \beta:[1:m+n]\to[0:n]}} \max_{s\in[1:m+n]} \{d(a(\sigma(s)), b(\beta(s)))\}$$

where σ and β range over all discrete non-decreasing onto mappings of the form $\sigma : [1 : m + n] \to [0 : m], \beta : [1 : m + n] \to [0 : n]$.

We first consider the corresponding decision problem. That is, given $\delta > 0$, we wish to decide whether $\delta_F^+(A, B) \leq \delta$. Consider the matrix M as defined in the subsection 5.1. In the two-sided version of Discrete Fréchet Distance with Shortcuts (DFDS), given a reachable position (a_i, b_j) of two pointers, the A-pointer can make a skipping upward move, as in the one-sided variant, to any point $a_k, k > i$, for which $M_{k,j} = 1$. Alternatively, the B-pointer can go to any point $b_l, l > j$, for which $M_{i,l} = 1$; this is a skipping right move in M from $M_{i,j} = 1$ to $M_{i,l} = 1$, defined analogously. Determining whether $\delta_F^+(A, B) \leq \delta$ corresponds to deciding whether there exists a sparse staircase of ones in M that starts at $M_{1,1}$, ends at $M_{m,n}$, and consists of an interweaving sequence of skipping upward moves and skipping right moves (see Figure 2)..

5.1. Basic Algorithm

The implementation of the algorithm of Fréchet Distance relied on the publication "The Discrete Fréchet Distance with Shortcuts via Approximate Distance Counting and Selection" (Anne Driemel),(Rinat Ben Avraham, 2014).

Specifically, the algorithm which was implemented is the two side - DFDS who faces the problem of outliers which is sensitive the Fréchet Distance. The result of the algorithm is the lowest Fréchet distance which satisfies the two trajectories. The pseudocode of the algorithm is represented in Algorithm 1. The steps of the basic algorithm are the following:

1. Implementation of binary search and sequential executions of the algorithm of Fréchet Distance until the optimal distance e is found. The initial value which is given to the middle in the binary search is 0.025. The binary search is terminated when the distance between low and high values of the middle is smaller than 0.001.

2. Then the table is created, whose columns correspond the points (latitude, longitude) of trajectory A and the lines the points of the trajectory B. The values taken by the M matrix is either 0 or 1 depending on the distance apart of these two points together, and are given by the following formula:

$$E_{\delta}^{+} = \\ ((a_i, b_j), (a_k, b_j))|k > i, ||a_i - b_j||, ||a_k - b_j|| \leq \delta \\ \cup((a_i, b_j), (a_i, b_l))|l > j, ||a_i - b_j||, ||a_i - b_l|| \leq \delta$$

3. Forthwith after, the directed graph G is created, based on the table M which was formed above. The nodes of the graph are the positions of the table M. Correspondingly it is created an edge between two nodes if the nodes are in the same row or column of the table and their values are 1 (from the previous to the next).

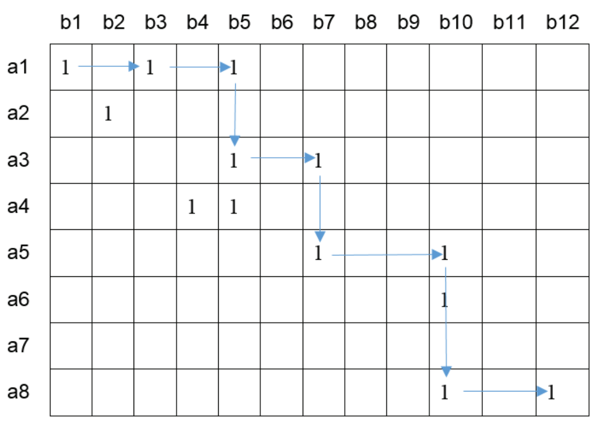

Figure 2. M table

4. In the figure 2, the algorithm Shortest Path is performed to see if there is a path from the first position of the table (0,0) to the last (N, M) wherein N and M are the dimensions of the two trajectories which were compared. If the algorithm of Shortest Path has a solution we check if the value of is between the limits we have set. In case it is the binary search terminates. Otherwise, it will perform again with latest prices. If not the execution of the algorithm terminates.

6. Evaluation

6.1. Fréchet Distance Performance

To measure the performance of the algorithm of the Fréchet Distance we performed experiments comparing time series with length ranged from 20000 to 100000 points. From the paper of Rinat Ben Avraham et all know that the complexity of the algorithm of Fréchet Distance with Shorcuts (DFDS) is $O((m^{2/3}n^{2/3} + m + n)log^3(m + n))$. The main difference with the previous version of the algorithm is that we calculate table T and simultaneously create also the graph G. In addition we set a parameter δ which represents the percentage of table points that we wish to compare. In this way, we improved the time execution of the algorithm of Fréchet Distance. Also due to the sampling of δ points in all time series, the complexity of the algorithm is reduced to linear.

The figure below shows the performance of the implementation of the Fréchet Distance algorithm compared with the length of the trajectories. In our experiment that is summarized by figure 3 we compared 50 trajectories with lengths from 20,000 points to 100,000 points. This experiment was performed 10 times and for each length of trajectories it was chosen the average execution time. Also, the chart shows the dispersion of runtime for any length of time series on error bars. From figure 3, we can conclude that the complexity of the algorithm is linear.

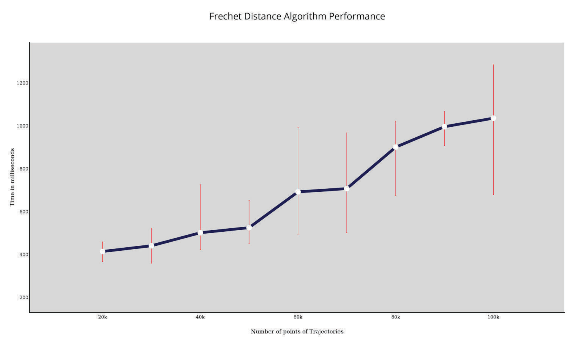

Figure 3. Frechet performance

Algorithm 1 Fréchet Distance

```
Input: trajectory $t_i$, trajectory $query$,int delta
Binary Search
repeat
  $rightTable[0..trajectory.length][0..1] = null$
  $downTable[0..trajectory.length][0..1] = null$
  for $i = 1$ to $t_i.length$ do
    $start \leftarrow i - delta$
    if $start < 0$ then start $\leftarrow$ 0
    $stop \leftarrow i + delta$
    if $start > query.length$ then stop $\leftarrow$ query.length
    for $j = start$ to $stop$ do
      if $x_i > x_{i+1}$ then
        $M[i][j] \leftarrow 1$
        if firstExecution then
          if M[0][0] == 0 then
            $return$
          else
            $graph.addVertex(0, 0)$
          end if
          $firstExecution \leftarrow$ false
        end if
        if rightTable[i][0] != ll and rightTable[i][1] != null then
          $graph.addVertex((i, j))$
          $graph.addVertex((rightTable[i][0], rightTable[i][1]))$
          $graph.addEdge((i, j), (rightTable[i][0], rightTable[i][1])))$
          $rightTable[i][0] \leftarrow i$
          $rightTable[i][1] \leftarrow j$
        else
          $rightTable[i][0] \leftarrow i$
          $rightTable[i][1] \leftarrow j$
        end if
        if downTable[i][0] != null and downTable[i][1] != null then
          $graph.addEdge((i, j), (downTable[i][0], downTable[i][1])))$
          $downTable[i][0] \leftarrow i$
          $downTable[i][1] \leftarrow j$
        else
          $downTable[i][0] \leftarrow i$
          $downTable[i][1] \leftarrow j$
        end if
      else
        $M[i][j] \leftarrow 0$
      end if
    end for
  end for
  // Find the shortest path if exists on graph from the
  // first to the last position of table M
  $graph.findShortestPath()$
until $shortestPath$ is $false$ and $frechetDistance \in$ BinarySearchRange
```

6.2. Algorithm Comparison

In this section we show preliminary experimental results that compare the three different trajectory similarity methods we have described. The focus of our evaluation is to show that all methods can be used in the limited resources environment of a smartphone. For this reason we also compare the run time results with the simpler (easier to implement and efficient to run) Euclidean distance that serves as a basic comparison point. It is difficult to compare the three methods because they have been implemented with different optimizations, and also our Fréchet distance implementation computes only a bound, and necessitates the use of several runs to compute the exact Fréchet distance. We use two different datasets for the comparisons. We also show a sample result of a query and the most similar answers that we get using the three methods.

The datasets which were used in the experiments are from the chorochronos.org and Dublin Bus GPS sample data from Dublin City Council (Insight Project). The parameters with which we will compare the algorithms performance are their time execution and their results. The name of the dataset from chorochronos.org is Trucks and contains 50 trajectories. The name of the dataset from Insight project is Siri and contains 30 bus trajectories. For the experiments, the first trajectory of the dataset was used as a query and compared with the remaining 49 trajectories. The experiments were performed for trajectories with length 2048 points each.

At the end of each experiment the program returns the graphs of execution time of each algorithm and the most similar trajectory chosen by each algorithm. The algorithms we compare are:

- Dynamic Time Warping (DTW)
- Longest Common Subsequence (LCSS)
- Fréchet Distance: DFDS to find the smaller Fréchet distance. In order for DFDS to find the smaller Fréchet distance, the algorithm should be executed multiple times for each trajectory that is compared as the algorithm described above suggests. (DFDS)
- Fréchet Distance - one execution of the algorithm (DFDS one time)

Bellow we present the results from the data provided by Insight Project. This dataset containts trajectories recorded from Dublin buses across Dublin City. In comparison with 2048 points per trajectory, the execution time of the algorithms is presented in **figure 4**.

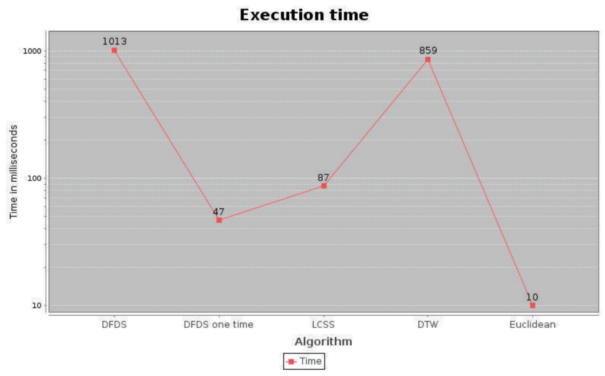

Figure 4. Algorithms execution time for 2048 points (Dublind Bus GPS sample)

The results of each algorithm that correspond to the most similar trajectory are listed below.

In **figure 5** is the most similar trajectory according to Discrete Fréchet Distance algorithm.

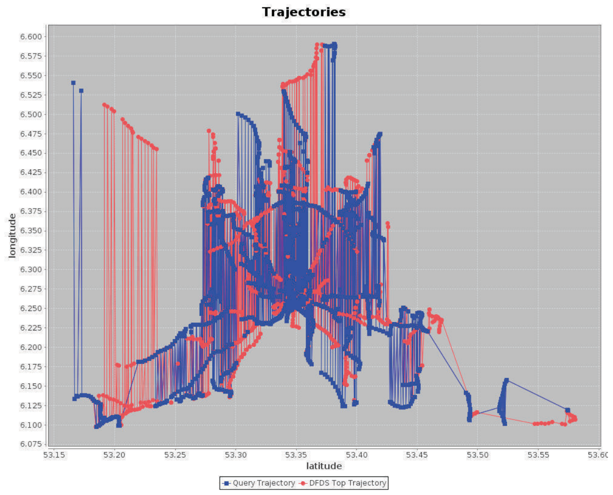

Figure 5. DFDS most similar trajectory for 2048 points (Dublind Bus GPS sample)

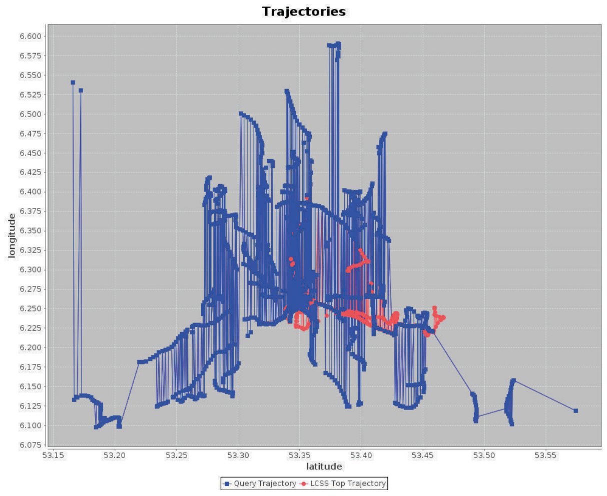

Figure 6. LCSS most similar trajectory for 2048 points (Dublind Bus GPS sample)

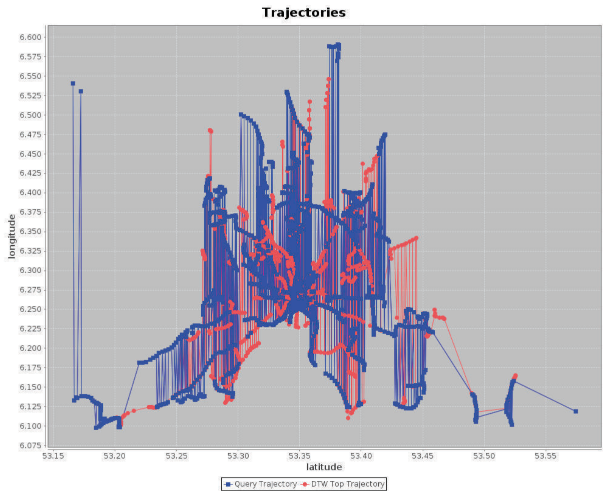

Figure 7. DTW most similar trajectory for 2048 points (Dublind Bus GPS sample)

In **figure 6** is the most similar trajectory according to LCSS algorithm.

In **figure 7** is the most similar trajectory according to DTW algorithm.

We conducted the same experiments with the dataset provided by chorocronos.org. Specifically with trajectories that contain 2048 points. In comparison with 2048 points per trajectory the execution time of the algorithms are presented in **figure 8**.

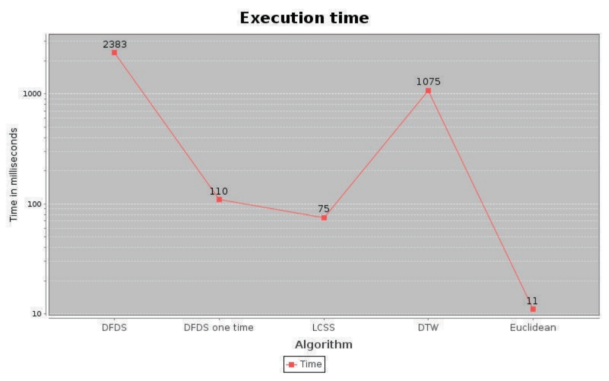

Figure 8. Algorithms execution time for 2048 points (chorochronos.org)

The results of each algorithm that correspond to the most similar trajectory are listed below.

In **figure 9** is the most similar trajectory according to Discrete Fréchet Distance algorithm

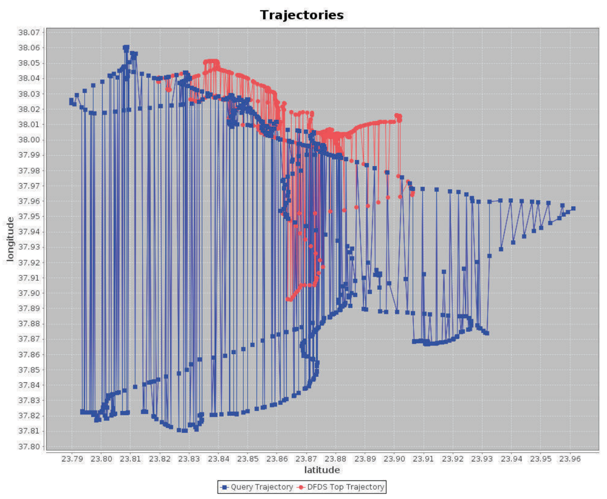

Figure 9. DFDS most similar trajectory for 2048 points (chorochronos.org)

In **figure 10** is the most similar trajectory according to LCSS algorithm

In **figure 11** is the most similar trajectory according to DTW algorithm

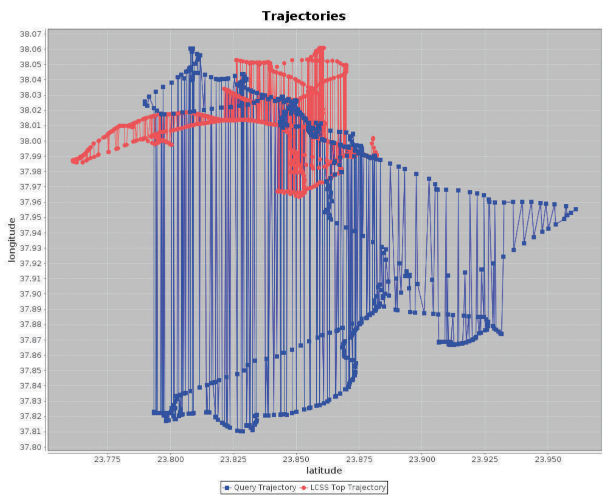

Figure 10. LCSS most similar trajectory for 2048 points (chorochronos.org)

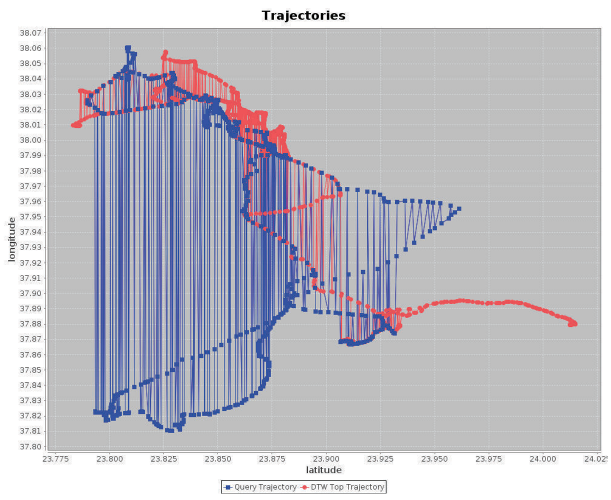

Figure 11. DTW most similar trajectory for 2048 points (chorochronos.org)

We observe that the distance measurement that have been implemented (Fréchet Distance) have similar execution time with the implementation of the LCSS algorithm. However, the final results of the algorithm differ. The execution time of a single run of Fréchet Distance algorithm is similar to the LCSS's algorithm execution time. The results of Fréchet Distance algorithm are based on the similarity of the shape of each trajectory in contrast with the other algorithm which are restricted to specific points comparison. Furthermore, we avoid the problem of outliers that Fréchet distance is sensitive to, thereby improving significantly the final results. Thus, we see that in comparison that we are interested more at having a similar shape rather than smaller distance of each point from the points of the query trajectory, Fréchet Distance produces better results than the other algorithms. Hence, Fréchet Distance is ideal for similarity measurement of the shape of trajectories This conclusion can be valuable for later implementations and research.

7. Conclusion

In conclusion, the trajectory similarity search problem have a plethora of applications, fact that gives the platform described in this work great potentials. Our system allows the user to send a query that contains a trajectory in order to receive a number of the most similar trajectories that have been recorded by other mobile devices that are registered to our system. We described the algorithms that the mobile application uses in order to measure the similarity between trajectory and finally search for the most similar ones. Moreover, the distributed architecture of our system grants it great capabilities such as scalability. We have evaluated our platform and the implemented algorithms using real spatio-temporal data. Our experiments prove the satisfying performance of our implementation. We compared each algorithm that we implemented and we evaluate both their performance and their final results according to the similarity between the selected trajectory and the query. We extracted useful conclusions about each algorithm's functionality and results, such as the different cases where the Fréchet distance algorithm is more suitable than the other distance measures. To sum up the final system that was implemented gives a number of final responses to queries submitted by its users, according to the selected algorithm in a satisfying time. A future goal is to extend the number of distance measurements that are implemented in order for the system to have more reliable and accurate final results.

8. Acknowledgements

We would like to thank Demetris Zeinalipour-Yazti, ”University of Cyprus” for his assistance by providing us the source code of the implementation of LCSS algorithm. This research was supported by the ARISTEIA MMD, FP7 INSIGHT, ERC IDEAS NGHCS and THALIS GeomComp projects.

References

Anne Driemel, Sariel Har-Peled, Carola Wenk. *Approximating the Frchet Distance for Realistic Curves in Near Linear Time*.

Donald J. Berndt, James Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop*, pp. 359–370. AAAI Press, 1994.

Giorgos Poulis, Spiros Skiadopoulos, Grigorios Loukides Aris Gkoulalas-Divanis. Select-organize-anonymize: A framework for trajectory data anonymization. In *Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on*, pp. 867 – 874, Dallas, TX, 2013. IEEE.

Lin, Bin and Su, Jianwen. Shapes based trajectory queries for moving objects. In *Proceedings of ACM GIS*, pp. 21–30. IEEE, 2005.

Michail Vlachos, Dimitrios Gunopulos, George Kollios. Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pp. 673 – 684, San Jose, CA, 2002. IEEE.

Rinat Ben Avraham, Omrit Filtser, Haim Kaplan Matthew J. Katz Micha Sharir. The discrete frchet distance with shortcuts via approximate distance counting and selection. In *SOCG'14 Proceedings of the thirtieth annual symposium on Computational geometry*, pp. 377, New York, NY, USA, 2014. ACM.

Zeinalipour-Yazti, Demetrios, Laoudias, Christos, Costa, Constandinos, Vlachos, Michail, Andreou, Maria I., and Gunopulos, Dimitrios. Crowdsourced trace similarity with smartphones. In *EEE Transactions on Knowledge and Data Engineering (TKDE '13), IEEE Computer Society*, pp. 1240–1253, Los Alamitos, CA, USA, 2013. IEEE.