University of Alberta

A Data Cleaning Framework for Trajectory Clustering

by

Agzam Idrissov

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

To my late father Yergazy Idrissov, to my loving mother Lyazzat Idrissova, to my great brother Alisher and to my dear friends
For all the endless support and inspiration

Abstract

Recent proliferation of low-cost and lightweight GPS tracking devices led to a large increase in the amounts of collected mobility data. The rapidly emerging field of location-based services requires accurate and informative knowledge mining from these large quantities of data. One such mobility knowledge mining task is trajectory clustering, where one tries to find paths that have been travelled frequently. Most existing trajectory clustering techniques do not discuss cleaning the data before applying a clustering algorithm. Since "noisy" data can have a significant effect on the clustering process, preprocessing such trajectory data will likely improve trajectory clustering results. In this thesis, we present a trajectory data cleaning framework, which consists of four steps: *Outlier Detection, Stop Detection, Interpolation and Map Matching*. We evaluate our framework using popular clustering algorithms and distance functions, and show that our proposed preprocessing (cleaning) framework indeed does improve the quality of obtained clusters.

Acknowledgements

First of all, I would like to sincerely thank my supervisor, Dr. Mario A. Nascimento, without whose guidance, encouragement and patience this thesis would never be completed. It was an honour for me to work with such a friendly, but professional at the same time, advisor.
I am also very grateful to Kazakhstan International Scholarship "Bolashak" for all the provided financial support. "Bolashak" gave me an opportunity to come to Canada and work on my degree at the University.
Finally, I would like to thank the Department of Computing Science at the University of Alberta for the enormous help throughout the whole period of my study.

Table of Contents

List of Figures

Chapter 1

Introduction

Perceiving and understanding movement patterns and behaviour has been a pervasive problem across multiple research fields [26]. Ornithologists may want to analyse birds' seasonal migration to detect any changes that were caused by certain environmental phenomena. Anthropologists may study people's movements to test their hypothesis regarding migration and population flow. Urban planning specialists can find *hot routes* - paths that are travelled most frequently to adjust further city growth accordingly. Other examples of applications of mobility data analysis include, but are not limited to, geo-marketing, transportation logistics and environmental management.

Today, with the proliferation of low-cost and portable GPS devices, tracking object movements is becoming more and more accessible. For example, animals and birds are easily equipped with lightweight GPS transmitters, whose readings are then regularly collected for further study. Aside from privacy issues, gathering human mobility data is becoming fairly accessible as well. With the development of wireless technologies, even rural areas are being covered by mobile network providers [26]. This usage of GPS-equipped devices (e.g. smartphones) allows the obtaining of large amounts of mobility data. For instance, Nokia's Lausanne dataset contains 200 people's location information for over the course of two years [36].

Since the volume of available mobility data is rapidly increasing, analysis of such large amounts of data is becoming not trivial for the domain experts. While it is relatively easy to analyse movement behaviour of a small number of individuals, performing the same task for thousands of objects will take enormous amount of effort and time. Therefore, appropriate knowledge discovery tools should be used to aggregate the data in some way and then the generated output is presented to an expert in a more clear and intuitive way. One of the main phases of the knowledge discovery in databases is *data mining* [28].

Data mining is an ensemble of statistical, machine learning and database tools that are aimed for knowledge discovery from large datasets, which makes the data more understandable for the final user [30]. According to [28], there are three most popular data mining techniques:

- *Predictive modelling*: In this technique, classification methods are used to learn the appropriate model based on a smaller set of already labelled training data. Then, when a new entry comes in, the learnt model tries to classify and label this new object. For instance, a pedestrian's future location can be estimated based on the model constructed from his/her movement history.

- *Association analysis*: The main goal of this method is to find notable relations between features in datasets. These relations are usually expressed by association rules with minimum confidence levels. One example could be a rule that states that a customer, who buys cereal will likely also buy milk in one transaction.

- *Clustering*: In clustering, the given dataset is partitioned into smaller clusters (groups), where elements are similar to each other, while being dissimilar to the objects of a different cluster. For instance, one may want to find groups of objects that were moving close to each other for a certain amount of time.

One of the popular tasks in mobility data mining is *trajectory clustering*. In the context of this thesis, a *trajectory* is defined as a sequence of temporally ordered points with spatial (e.g. longitude, latitude) and non-spatial attributes (e.g. object ID). Therefore, trajectory clustering is the process of grouping similar trajectories together.

In general, trajectory clustering can refer to several types of analysis. Some applications require finding clusters of trajectories where objects moved close to each

other at the same time, i.e. both spatial and temporal collocation is needed. Other works focus on trajectories that made similar turns in one moment of time, i.e. spatial collocation is neglected [26]. The focus of this thesis will be on, arguably, the most common trajectory clustering subtask - finding groups of trajectories where objects travelled along the same path during the whole duration of a sampling, i.e. temporal collocation is not considered. This trajectory clustering subtask is often employed in a large amount of applications in various research fields. In [1], this kind of trajectory cluster analysis was performed to find high concentrations of sulphate and nitrate in the atmosphere above the Belfast area. The authors of [8] use trajectory clustering to improve the accuracy of their automatic pedestrian counting video system. One biomedical application is presented in [29], where Haniu et al. perform trajectory clustering analysis to study postnatal protein development. Andrienko et al. find trajectory clusters in evacuation traces after an explosion to study people's behaviour in critical situations [6].
The problem of data quality arises when the existing trajectory clustering algorithms are used on the real world datasets. Usually, trajectory data is collected by attaching a GPS device to an object. Ideally, the accuracy of GPS positioning is 5-10 meters [57]. However, in the real world this accuracy is significantly decreased due to the following factors [34, 61]:

- limited satellite visibility
- satellite or receiver problems
- atmospheric and ionospheric conditions
- multi-path signal reflection
- signal blocking

The last two error sources are especially significant in urban areas where a signal reflects from several buildings or even becomes totally obstructed. These errors can substantially distort the recorded GPS positions [61].
When data quality decreases, clustering quality will also be affected. In [41], the

authors conducted a study where they have shown that improving data quality will also increase the clustering accuracy. In particular, they added Gaussian noise to GPS coordinates of a dataset and visually compared the difference between the resulted clusters of points. They discovered that adding this noise made clusters break down into smaller clusters, while, visually, these clusters should have been the members of one large cluster.

In this work, however, our task is a bit different - instead of investigating the effects of noise on performing clustering of *points*, we attempt to improve the *trajectory* clustering quality by processing a dataset before clustering. Following this, we propose a trajectory cleaning framework for mobility datasets, which can be applied to any given trajectory dataset prior to trajectory clustering. Our framework consists of four main steps:

- *Outlier Detection.* As shown in [41] outliers can significantly decrease the clustering quality. Therefore, during this step, outliers are detected and removed using statistical methods.

- *Stop detection.* Since our goal is to find frequently travelled paths, we are interested in clustering the actual movements of objects. Hence, the gatherings of points where an object spent some time without any movement is not relevant to our task. Therefore, this step finds these stops and removes them.

- *Interpolation.* During GPS sampling it is often the case when a GPS receiver looses connectivity with the satellites. In such cases, parts of a trajectory remain missing. The *Interpolation* step fills those and the empty segments that resulted after the *Stop Detection* step with needed points.

- *Map Matching.* This step is optional and can be applied when a high accuracy map of a location where GPS points were sampled is present. During this step points are matched to actual roads and routes on the obtained map to improve overall data quality.

We then evaluate our framework by computing inter-cluster similarities for two cases: clusterings that were obtained before applying our cleaning framework and after. The conducted experiments show that our trajectory cleaning framework is able to significantly improve the quality of the resulted clusters with respect to the chosen clustering quality measure on the real world datasets.

The rest of the paper is organized as follows. Chapter 2 discusses the related work that has been done prior. Chapter 3 describes the proposed trajectory cleaning framework itself and each of the 4 steps. Then, the evaluation of our framework is presented in Chapter 4. Finally, Chapter 5 concludes the thesis and sets directions for future work.

Chapter 2

Related Work

In this chapter we will discuss the existing literature related to our research. We will start with other previously proposed trajectory preprocessing frameworks. Then, existing proposed trajectory clustering algorithms and distance functions are discussed, because some of them are used during experimental evaluation of the proposed framework. Finally, we will touch on trajectory-specific clustering algorithms.

2.1 Trajectory Preprocessing

In general, GPS data preprocessing methods can be divided into two categories. One group of algorithms uses only collected attributes of a GPS reading (such as longitude, latitude, speed and heading) to detect noise points. Other methods use additional satellite information (e.g. the number of the visible satellites) to preprocess the GPS readings and make a decision whether each point can be considered reliable. In the following, we will briefly survey existing algorithms of both categories.

2.1.1 Additional Information GPS Preprocessing

One of the earlier works in this field is presented in [14], where authors use satellite-receiver pseudorange and signal frequency information for outlier detection. Another group of methods [46, 61, 56] use *positional dilution of precision*, a parameter, which represents how good the satellite visibility was at the time of sampling. For instance, if this parameter is smaller than 5, the point is considered unreliable and dismissed from further processing. Another algorithm that uses additional information is presented in [16], where authors also apply map matching method to clean the dataset. However, for its outlier detection step, that algorithm requires the number of visible satellites in a processed dataset, which is not always available. In our work, we will focus on algorithms that do not require any additional satellite-receiver information, since most GPS datasets do not always have this information readily available.

2.1.2 GPS Preprocessing without Additional Information

During the stop detection step of our framework we will extend the algorithm proposed in [4]. In that trajectory preprocessing framework, the authors propose to use the concepts of *stops* and *moves* for the semantic enrichment of trajectories. *Stops* are places on a trajectory where an object spent some minimum amount of time. *Moves* are parts of a trajectory between stops. The framework consists of three steps: Data Cleaning, Adding Semantics and Transformation steps. In the Data Cleaning step, several restrictions are applied to a raw dataset to eliminate some of the noise:

- the speed between two points should be less than a predefined threshold
- points are sorted in the order of timestamps
- points with the same timestamps are removed
- a trajectory should have some minimum number of points

The Adding Semantics step detects stops and moves, while the Transformation step adjusts the output format of previous steps for the chosen data mining algorithm.
Our framework differs from the above approach in two key aspects. First, in the above method, outliers are detected using a fixed speed threshold. Such approach will fail to consider cases when average speeds are different (e.g. a person may be walking or driving a car). We address this limitation by using a statistical approach

to determine appropriate speed thresholds to detect outliers. Second, we use interpolation to fill the missing segments of a trajectory that were resulted after the stop removal. In addition, if an accurate road network map is present, we apply map matching algorithm to improve the accuracy of the GPS point coordinates.
A slightly different approach is taken in [54], where the authors also have a three-step preprocessing framework: data filtering and smoothing, trips and activities detection, and mode detection. During the first step, a dataset is filtered by assuming the maximum speed and error buffer and then smoothed by using the Gaussian Kernel. Then, stops and moves are detected using the minimum time spent in one location. Lastly, travel modes (e.g. pedestrian, bicycle, car, train) are detected using fuzzy membership functions. The authors also mention including map matching algorithm in the data filtering step, but there is no indication of conducting experiments that include that in the paper. In addition, during the stop detection phase, authors use fixed speed and distance thresholds which are not tied to a travel mode. This might cause problems when the average speed varies. For instance, one might define a car's movement a stop, when the average speed is less than 5 km/h. However, for a pedestrian it is still a movement, not a stop. A fixed threshold speed will detect only one type of a stop, but not both. In addition, in our work we attempt to evaluate how data cleaning actually affects trajectory clustering quality.

2.2 Distance Functions

Many trajectory clustering approaches rely on specifically designed distance functions [62, 15, 59]. A *distance function* is a metric that is used to determine the similarity between two trajectories in a set of trajectories. If the distance function value for two given objects is small, then these objects are considered similar. Conversely, a large distance function value suggests that the objects are dissimilar.
Distance functions usually have the following properties [2]:

- the distance cannot be negative
- if two objects are equal, the distance between them is zero

- if the distance from object A to object B is x, then the distance from B to A is also equal to x (symmetry)
- the distance from object A to object C via object B is less or equal to the distance from A to C directly (triangle inequality)

Typically, most distance functions for trajectory clustering are adopted from the time series domain [65], because a trajectory is, in essence, a time series, where each observation is represented by a set of spatial coordinates. Next, we briefly survey the existing literature regarding trajectory distance functions. We use notations adopted from [42] and [65], which compare performances of several popular distance measures.

Suppose two trajectories A and B are represented by sets of two dimensional points, with coordinates x and y and denoted as $(a_i^x; a_i^y)$ and $(b_j^x; b_j^y)$, respectively. In the following, we will define a distance D between these two trajectories on based the chosen distance function.

2.2.1 Euclidean Distance

The Euclidean distance between two points is defined as the length of the line segment that connects these two points. Therefore, the Euclidean distance between two trajectories is simply the average distance between respective points of both trajectories [21]:

$$D_{Euclidean}(A, B) = \frac{1}{N} \sum_{i=1}^{N} \left[(a_i^x - b_i^x)^2 + (a_i^y - b_i^y)^2 \right]^{\frac{1}{2}}$$

To determine the Euclidean distance between two trajectories, they need to be of the same length N, i.e. the number of points in both trajectories needs to be equal. This limits the applicability of this distance function for most real world trajectory datasets.

2.2.2 PCA Distance

In [10], the authors propose the *Principal Component Analysis* (PCA) on trajectories. First, the dimensionalities of all the trajectories are reduced to a set of one dimensional vectors. Then, this set of vectors is merged into one data table, which represents the whole set of trajectories. After that, eigenvector decomposition of the covariance matrix is applied to extract the PCA components. Then, the transformation matrix Φ is used to obtain the Principal Component (PC) coefficients which represent each trajectory:

$$Y = \Phi_M^T \left[X - \bar{X}\right]$$

where X is the matrix that contains all the trajectories, $\bar{X}$ is the dataset mean vector and Y is the resulted matrix with PC coefficients. Finally, the PCA distance between two trajectories is equal to the Euclidean distance between the respective PC coefficients of the two trajectories.
Piece-wise analysis of PCA distance allows handling of noise in trajectory data. However, like in Euclidean distance, only trajectories with equal sizes can be processed with this algorithm.

2.2.3 DTW Distance

Dynamic Time Warping distance (DTW) [12] attempts to find the optimal alignment between two given time series in a non-linear fashion [38]. First, a DTW distance matrix is created, where the Euclidean distances between points of compared trajectories are stored. Then, a *warping path* is defined as a set of elements of a DTW distance matrix that satisfies the following conditions:

- boundary condition: the first element of a warping path should always be equal to the top left element of the DTW distance matrix. Similarly, the last element of a warping path should be the same as the last (bottom right) element of the DTW matrix

- continuity condition: the next element of a warping path is always chosen from the adjacent cells of the matrix

- monotonicity condition: the indexes of points in a warping path should always increase or stay the same

Note, that a warping path can be found using dynamic programming approach, so one does not need to traverse the whole distance matrix.
Several warping paths may exist between two trajectories. Now, suppose w_0, w_1, ..., w_i, ..., w_K are the elements of the warping path W that has the minimum distance sum among all the warping paths. Then, the DTW distance between two trajectories is defined as follows.

$$D_{DTW} = \frac{\sum_{K}^{i=1} w_i}{K}$$

Unlike the above distances, DTW distance does not require trajectories to have the same number of points. This is due to the "warping" nature of this distance measure. However, because DTW considers each point of a trajectory, DTW has been shown to be quite sensitive to noise [42].

2.2.4 LCSS Distance

Vlachos et al. [59] propose another similarity measure based on the well-known Longest Common Subsequence problem (LCSS). The key idea of this approach is similar to DTW: try to align two time series by allowing them to stretch. However, unlike DTW algorithm, LCSS calculates the number of matches (LCSS score) rather than the actual distance between respective points. If there is no match between the compared points, these points will not contribute to the overall score. This allows LCSS to alleviate the effect of outliers to the final distance. Given trajectories A and B with sizes N and M respectively, the LCSS score between them is defined as [59]:

$$LCSS(A,B) = \begin{cases} 0, & \textit{if } A \textit{ or } B \textit{ is empty} \\ 1 + LCSS_{\sigma,\epsilon}(First(N-1,A), First(M-1,B)), & \\ & \textit{if } ||a_N - b_M|| < \epsilon \textit{ and } |N-M| \leq \sigma \\ max(LCSS_{\sigma,\epsilon}(First(N-1,A),B), & \\ \quad LCSS_{\sigma,\epsilon}(A, First(M-1,B))), \textit{otherwise} & \end{cases}$$

where $First(F,T)$ is a set of all points F except the last point of a trajectory T. LCSS uses two thresholds: a distance threshold ϵ and a temporal threshold σ. Two points are considered a match if their distance is within ϵ and if the time difference between them does not exceed σ.

Then, the distance between trajectories A and B is defined as:

$$D_{LCSS} = 1 - \frac{LCSS_{\sigma,\epsilon}(A, B)}{max(N, M)}$$

In this work we use both DTW and LCSS distances to show that when these distance functions are used, proper trajectory preprocessing can improve clustering quality. While DTW is known for its sensitivity to noisy data [42], our experiments show that we can also improve clustering accuracy by applying our framework prior to using LCSS as a distance function for clustering. We will discuss clustering algorithms in the next section.

2.3 Clustering

Clustering (often called *cluster analysis*) is considered an important unsupervised learning problem in data mining. Given a set of unlabelled data, the goal of clustering is to identify the inner structure of the given dataset without labels available [28]. Then, a *cluster* is a set of objects, where objects are similar to each other within a cluster and dissimilar to the objects of another cluster.

Next, we will briefly discuss existing clustering algorithms, which can be divided into four main groups [51]:

- partitioning methods
- hierarchical methods
- density-based methods
- other methods

2.3.1 Partitioning Methods

Partitioning clustering methods split the whole dataset into smaller sets according to the chosen partitioning criteria [51]. For instance, in *k-means* [40], given the number of clusters k, the algorithm will divide the whole dataset into k clusters.

The partitioning in *k-means* is achieved by finding the centroid for each cluster, such that its distance to the rest of the points inside the cluster would be as small as possible. The algorithm starts by randomly dividing the whole dataset into k partitions. Then, for each partition, the centroid is calculated and all the objects are assigned to a partition with the closest centroid. After that, new centroids are calculated and the process iterates until all the centroids remain unchanged.

While *k-means* is simple, it suffers from several serious drawbacks. First, the number of clusters should be known *a priori*. Second, *k-means* tends to produce clusters of circular shapes, although there could be actual clusters with various shapes. Lastly, the algorithm is quite sensitive to noise [51]. A variation of *k-means*, an algorithm called *k-median*, tries to alleviate this problem. Instead of using the computed centroid point itself (this point may not exist in the data because the location of this point is computed), *k-median* uses an actual data point which is closest to the computed centroid. The *C-means* [13] fuzzy algorithm is another representative of the partitioning clustering methods. Fuzziness probability shows to what degree certain event occurs. For instance, instead of strict statements, like "true" or "false", fuzziness allows the use of statements such as "true to a certain degree", "slightly cold" and "moderately high". In clustering that uses traditional binary logic, each data point will either belong or not belong to a cluster. In contrast, in fuzzy clustering each point is assigned a degree of membership to each cluster. That is, one point can belong to several clusters with varying degrees of membership.

C-means also starts with dividing data into c partitions. Then, for each partition, the centroid is calculated based on some initial guess. Next, the degree of membership of an each point is calculated for all clusters. After that, new centroids for each cluster are calculated based on the given distance function and the degrees of membership. This process continues until the maximum change in degrees of membership fails to exceed the given stop condition threshold [13].

2.3.2 Hierarchical Methods

Partitioning clustering methods fail to detect clusters with varying densities, because global parameters (such as k in *k-means*) that would find those clusters with different densities cannot be determined *a priori* [20]. *Hierarchical clustering* alleviates this problem by representing clusters in the form of a tree (often called a *dendrogram*). There are two types of hierarchical algorithms: agglomerative and divisive [11]. In *agglomerative clustering* each object is initially a separate cluster. These clusters are then recursively merged until the whole dataset becomes a single cluster. Conversely, in *divisive clustering*, the process starts with the whole dataset being one cluster and then divides it into smaller clusters based on a cluster closeness threshold. Both agglomerative and divisive approaches allow generating a dendrogram. The cluster closeness is then determined based on the following distance functions for clusters [51]:

- single link: the minimum pairwise distance between the elements of one cluster
- complete link: the maximum pairwise distance between the elements of one cluster
- average link: average pairwise distance between the elements of one cluster

In [64], the authors propose an iterative and dynamic hierarchical clustering algorithm, *Balanced Iterative Reducing and Clustering using Hierarchies* (BIRCH). One of the problems with the previous clustering algorithms is that they try to cluster the whole dataset at once, which sometimes may not fit into the main memory. BIRCH tries to alleviate this problem by incrementally building a tree of (clustering) features and then accessing those features depending on the memory capacity. In essence, each clustering feature contains a set of pointers to its descendants in the tree, where this set of descendants represents a sub cluster of a node. Then, the size of the whole tree is adjusted using a threshold on the diameter of a leaf cluster. The diameter of a cluster is defined by the distance (e.g. Euclidean) between the two mutually furthest points that belong to this cluster.

BIRCH starts with scanning all data points and creates a tree of clustering features. Then, the algorithm scans all the leaf nodes (nodes on the bottom of a tree) and generates a smaller tree by merging clusters based on the given leaf cluster diameter threshold. Then, the whole tree is given as an input to any existing clustering algorithm for further clustering. Finally, the postprocessing step which removes outliers is applied to the resulted tree. During this step, for each generated cluster, a centroid seed is calculated; points that are too far from that seed are considered outliers and removed.

Since BIRCH uses centroids to redistribute points within each cluster, it fails to detect clusters with non-uniform shapes [27]. This problem is addressed by CURE [27], which uses both centroid and graph-based redistribution of points. First, CURE randomly scatters representative points to cover non-spherical shapes of clusters. Then, a cluster is represented by using c number of representative points. This way, the single link distance between two clusters is defined by the minimum distance between any two representatives of those clusters. However, this algorithm suffers from one significant disadvantage: as in partitioning methods, the number of clusters should be specified in advance in order for the methods to stop during bottom up joining of clusters.

In general, hierarchical clustering has several key advantages compared to partitioning methods, such as not needing to specify the number of clusters in advance (except for CURE as discussed above) and convenient interpretation of clustering results. The main disadvantage of these methods is that they cannot change clustering after splitting or merging of clusters is made. In other words, the clusters are formed based on the local decisions, which may not be globally reasonable. Thus, the results are sensitive to the order in which data is processed [51].

2.3.3 Density-based Clustering

Density-based clustering algorithms attempt to use the density of data points in certain areas to find clusters. Arguably, the most well-known algorithm in this field is DBSCAN [23]. Given two parameters, the neighbourhood distance threshold Eps and the minimum required number of points within neighbourhood $MinPts$, DBSCAN clusters each point based on the following types of points:

- *Core point*: a point P that has at least $minPts$ points within its Eps -neighbourhood. Eps-neighbourhood of a point is defined as the maximal set points that lie within Eps distance from that point.

```
Algorithm: DBSCAN
Input: $SetOfPoints$, $Eps$, $MinPts$
Output: $SetOfClusters$
$SetOfPoints$ is UNCLASSIFIED
$ClusterID$ := nextID(NOISE)
for $i = 1 \rightarrow SetOfPoints$ do
    $Point$ := $SetOfPoints$.get($i$)
    if $Point.ClId$ = UNCLASSIFIED then
        if ExpandCluster($SetOfPoints$, $Point$, $ClusterId$, $Eps$, $MinPts$) then
            $ClusterId$ := nextId($ClusterId$)
        end if
    end if
end for
```

Figure 2.1: DBSCAN clustering algorithm pseudocode

- *Directly density-reachable points*: all the points within core point's Eps-neighbourhood (non-core points are called *border points*)
- *Density-reachable points*: points that can reach each other via some sequence of other directly density reachable points
- *Density-connected points*: two points that are both density reachable to a common point

Following the above definitions, a cluster in DBSCAN is then formed from the maximal set of mutually density-connected points.
The algorithm starts by randomly choosing a point and checking its Eps - neighbourhood. If this point satisfies the core-point condition, then the cluster based on all density-connected points is formed. Then, all the points within the Eps-neighbourhood of density-reachable points are added to the cluster as long as their density is large enough ($ExpandCluster$ function in the pseudo code below). If the point is a border point and does not have any density-reachable points, then the next random unclustered point is chosen for further processing. Points that are neither core points, nor border points are considered noise. A pseudo code of the basic version of DBSCAN is presented in Figure 2.1.
In this work, we will use DBSCAN three times. First, a modification of this algorithm is used during the stop detection step of our framework. Stops are detected by clustering dense areas, where an object spent a significant amount of time. Second, we will use DBSCAN when evaluating our framework and inves-

tigate whether trajectory cleaning actually produces better clustering. Lastly, one of the most well-known trajectory clustering algorithms, TRACLUS [37], uses a variant of DBSCAN to perform line segment clustering; to evaluate our framework further, we will use TRACLUS as well.

2.3.4 Other methods

An evolutionary improvement over DBSCAN is proposed in [7]. One problem with DBSCAN was that it is not designed to detect clusters with varying densities. *Ordering Points To Identify the Clustering Structure* (OPTICS) addresses this problem by using the hierarchical approach in conjunction with density-based clustering. While the same Eps and $MinPts$ parameters are used, OPTICS creates cluster hierarchies with different densities by varying the Eps parameter. In essence, a dendrogram of density-connected clusters is created for a more intuitive and convenient interpretation of the structure of the given dataset.
Even though OPTICS is considered a more advanced clustering algorithm, in this work we will use DBSCAN for its prevalence among trajectory clustering approaches [37, 39, 9]. Note, however that our framework can be easily applied prior to any clustering algorithm of choice.

2.4 Trajectory Clustering Algorithms

One of the common tasks in analysing motion data is finding frequently traversed routes - trajectory clusters. One of the first steps in this field is presented in [24], where authors present a probabilistic modelling approach to trajectory clustering. First, trajectories are represented by a mixture of regression models. Then, the *Expectation-Maximization* algorithm (also referred to as *EM-Step*) is used to estimate the needed weights for the created model. During the *E-step*, for each trajectory, a matrix Z with degrees of membership for each cluster is computed. Then, given membership matrix Z, the *M-step* calculates the maximum likelihood parameter estimates for the created model. Finally, clusters are extracted from the given data based on the learnt model. A similar algorithm is proposed in [3], where authors use the Hidden Markov models instead of the mixture regression models. However, one limitation of this category of algorithms is that large amounts of noise cause a decrease of the accuracy of the EM-step [63].

Visual approaches constitute another category of trajectory clustering methods. Andrienko et al. [5] propose a visually-aided algorithm that uses human expertise to iteratively improve clustering results. Schreck et al. [53] mix *Self-Organizing Kohonen Maps* clustering with visually interactive human expert judgement to produce better clusters. However, in this thesis, our focus is on clustering methods that do not rely on any expert interaction.

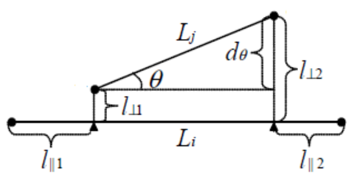
A geometric approach is proposed in [44], where the authors divide trajectories into segments with temporally close endpoints. Then, those segments are clustered using OPTICS and the average Euclidean distance function. Because this algorithm considers only contemporary line-segments, it will not detect frequently travelled routes that are separated in time. Our goal, however, is to find paths that were travelled during the whole sampling period.

One of the most well-known trajectory clustering approaches is presented in TRACLUS [37]. Like in the above method ([44]), the main idea of this algorithm is to perform trajectory partitioning and to cluster the resulted subtrajectories.

First, each trajectory is approximated using a subset of characteristic points - points where the trajectory's direction changes significantly. This is achieved using the *Minimum Description Length* principle, where these characteristic points are chosen in such way, that the resulted line segments would best describe the original trajectory and their number would be as small as possible. Then, the line segments between the consecutive characteristic points participate in the clustering process. The distance function for the clustering process considers the mutual angle, the length of line segments and the distance between endpoints of those line segments [37] as shown in Figure 2.2.

Then, the distance function between two line segments L_i and L_j is defined as the weighted sum of perpendicular, parallel and angular distances:

$$dist(L_i, L_j) = w_\perp \cdot d_\perp(L_i, L_j) + w_\parallel \cdot d_\parallel(L_i, L_j) + w_\theta \cdot d_\theta(L_i, L_j)$$

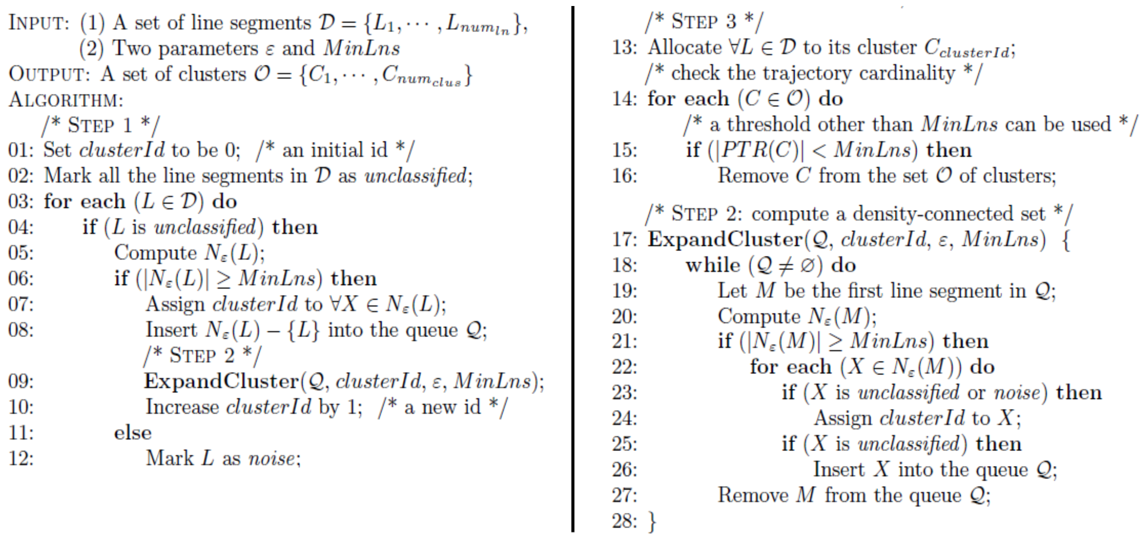


$$d_\perp = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}}$$

$$d_\parallel = \mathrm{MIN}(l_{\parallel 1}, l_{\parallel 2})$$

$$d_\theta = \|L_j\| \times \sin(\theta)$$

Figure 2.2: Distance function components in TRACLUS, taken from [37]



```
INPUT: (1) A set of line segments $\mathcal{D} = \{L_1, \cdots, L_{num_{ln}}\}$,
       (2) Two parameters $\varepsilon$ and $MinLns$
OUTPUT: A set of clusters $\mathcal{O} = \{C_1, \cdots, C_{num_{clus}}\}$
ALGORITHM:
  /* STEP 1 */
  Set $clusterId$ to be 0;  /* an initial id */
  Mark all the line segments in $\mathcal{D}$ as $unclassified$;
  for each ($L \in \mathcal{D}$) do
      if ($L$ is $unclassified$) then
          Compute $N_\varepsilon(L)$;
          if ($|N_\varepsilon(L)| \geq MinLns$) then
              Assign $clusterId$ to $\forall X \in N_\varepsilon(L)$;
              Insert $N_\varepsilon(L) - \{L\}$ into the queue $\mathcal{Q}$;
              /* STEP 2 */
              ExpandCluster($\mathcal{Q}$, $clusterId$, $\varepsilon$, $MinLns$);
              Increase $clusterId$ by 1;  /* a new id */
          else
              Mark $L$ as $noise$;
  /* STEP 3 */
  Allocate $\forall L \in \mathcal{D}$ to its cluster $C_{clusterId}$;
  /* check the trajectory cardinality */
  for each ($C \in \mathcal{O}$) do
      /* a threshold other than $MinLns$ can be used */
      if ($|PTR(C)| < MinLns$) then
          Remove $C$ from the set $\mathcal{O}$ of clusters;

  /* STEP 2: compute a density-connected set */
  ExpandCluster($\mathcal{Q}$, $clusterId$, $\varepsilon$, $MinLns$) {
      while ($\mathcal{Q} \neq \emptyset$) do
          Let $M$ be the first line segment in $\mathcal{Q}$;
          Compute $N_\varepsilon(M)$;
          if ($|N_\varepsilon(M)| \geq MinLns$) then
              for each ($X \in N_\varepsilon(M)$) do
                  if ($X$ is $unclassified$ or $noise$) then
                      Assign $clusterId$ to $X$;
                  if ($X$ is $unclassified$) then
                      Insert $X$ into the queue $\mathcal{Q}$;
          Remove $M$ from the queue $\mathcal{Q}$;
  }
```

Figure 2.3: Line segment clustering algorithm in TRACLUS, taken from [37]

Finally, the authors use a modified version of DBSCAN to cluster all line segments using the defined distance function (the pseudocode for this line segment clustering algorithm is presented in Figure 2.3). The difference is that TRACLUS does not consider trajectories that have an insufficient number of the line segments (denoted as PTR in Figure 2.3). Lastly, for each cluster of line segments an average direction vector called *a representative trajectory* is computed.

Another contribution of TRACLUS is that the authors use a clustering quality measure to evaluate the produced clusters. We will use this measure (discussed in Section 4.1) and the algorithm itself later during the experiments for the quantitative evaluation of our framework.

Chapter 3

Trajectory Data Cleaning Framework

In this chapter the proposed trajectory data cleaning framework is presented. First, gatherings of points where an object spent a considerable amount of time (stops) are detected and removed in the *Stop Detection* step. Second, in the *Interpolation* step, missing parts of each trajectory are completed with generated and evenly spread points. Then, during the *Outlier Detection* step noise points and outliers are removed using a statistical approach. Finally, when an accurate geographical map is present, a map matching algorithm can be applied *a posteriori* to further improve data quality. The description of the used map matching procedure is given in Section 3.4.

3.1 Stop Detection

The main motivation behind the elimination of stops is that, in most cases, analysts are interested in the actual *movement* of an object. However, in the real world, objects often make short stops during travel. Cars stop at intersections, animals stop to eat grass or drink water, pedestrians stop to talk to other people - these kinds of stops do not contribute to the actual movement. In fact, many popular distance functions that are used for mobility data analysis are quite sensitive to them. For instance, as shown in Figure 3.1, since DTW compares each point in a trajectory, visually similar trajectories will have a higher distance function value between them because of those stops. While trajectory A (without a stop) and trajectory B (with

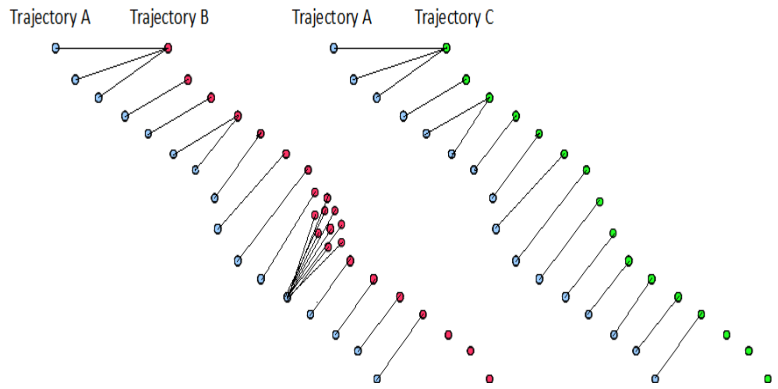


Figure 3.1: Processing of trajectories with and without stop by using DTW

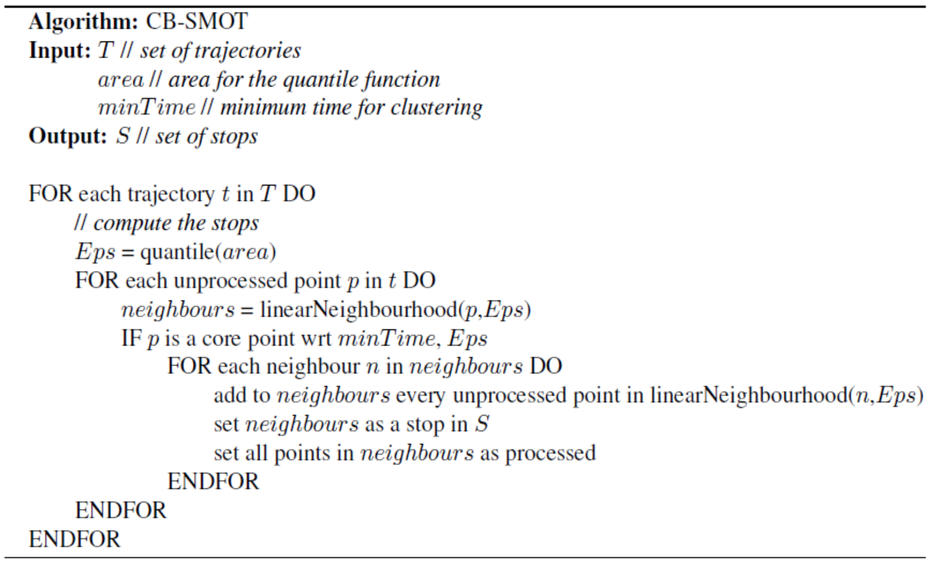
a stop) are similar, the DTW distance between trajectory A and trajectory B will be larger compared to the distance between trajectory A and trajectory C (without a stop).

Our stop detection method is based on Palma et al.'s work [48] on discovering interesting places on a trajectory. In this work, the authors present an algorithm called *CB-SMOT* that finds interesting places (stops) on a trajectory using density-based clustering. We will use the idea behind this stop detection algorithm as one of the steps in our framework to find these stops and eliminate them.

In the original work the authors require a user to specify two parameters: $MinTime$ - minimum amount of time spent at one location and Eps - the distance threshold, which is used to represent how close points should lie to each other to be considered a stop. We will show that it is possible to detect stops without the need to specify Eps.

CB-SMOT stop detection algorithm uses the DBSCAN clustering method to find places where the point density is high enough to be considered a *stop*. A *stop* is defined as a place on a trajectory where an object spent some minimum amount of time. Stops are, in essence, clusters that were formed from all the density-reachable points from the respective core points.

The difference between CB-SMOT and DBSCAN's point clustering scheme is in their core point conditions. While the latter uses the minimum number of points ($MinPts$ parameter) to define cluster density, in the former, the stops are based on the minimum time spent in one location ($MinTime$). This way, *a core point* P



```
Algorithm: CB-SMOT
Input: $T$ // set of trajectories
       $area$ // area for the quantile function
       $minTime$ // minimum time for clustering
Output: $S$ // set of stops

FOR each trajectory $t$ in $T$ DO
    // compute the stops
    $Eps$ = quantile($area$)
    FOR each unprocessed point $p$ in $t$ DO
        $neighbours$ = linearNeighbourhood($p$,$Eps$)
        IF $p$ is a core point wrt $minTime$, $Eps$
            FOR each neighbour $n$ in $neighbours$ DO
                add to $neighbours$ every unprocessed point in linearNeighbourhood($n$,$Eps$)
                set $neighbours$ as a stop in $S$
                set all points in $neighbours$ as processed
            ENDFOR
    ENDFOR
ENDFOR
```

Figure 3.2: CB-SMOT algorithm pseudocode [48]

with respect to Eps and $MinTime$ in CB-SMOT is defined as a point that satisfies the following condition:

$$T_{latest} - T_{earliest} \leq MinTime$$

where T_{latest} and $T_{earliest}$ are two points with, respectively, the latest and the earliest timestamps that lie within the Eps-neighbourhood of the candidate core point P. This condition means that not all high density regions will be considered as stops, but only those where the object spent a relatively large (with respect to $MinTime$ parameter) amount of time.

The authors also present a heuristic to estimate the Eps parameter by using a quantile function over the standard deviation of distances between consecutive points of a trajectory. However, a user still needs to enter a parameter called *area* that represents the approximate proportion of stop points per trajectory. The relevant pseudo code of CB-SMOT is shown in Figure 3.2.

In our work, we modified the algorithm to automate the Eps parameter selection. In CB-SMOT, a user needs to specify the $area$ parameter, which represents a proportion of points that can possibly generate stops and are used for the further Eps parameter estimation. However, we found out that using the mean of consecutive distances between points for Eps parameter is sufficient for finding these stops. Our

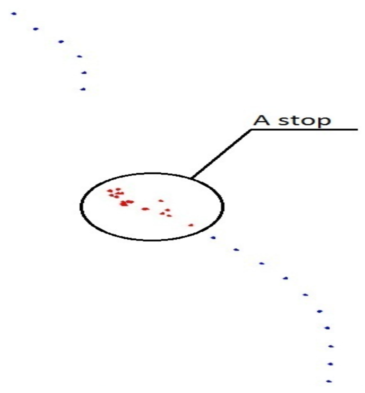


Figure 3.3: A stop, detected by an estimated Eps parameter

experiments show that this parameter estimation technique can be successfully applied to detect all the major stops on a trajectory. Figure 3.3 illustrates an example of a stop that was found using our improved version of the original CB-SMOT algorithm. While the same stop could be found by the original CB-SMOT algorithm, our modification of this method does not require the $area$ parameter.

After all the stops on a trajectory are detected, we remove them and fill the resulted gap in the next step of our framework - *Interpolation*, which is described in the next section. This way, even though a stop is removed, the information that an object travelled at a particular location is preserved.

3.2 Interpolation

Limited satellite visibility or receiver issues can lead to situations when a GPS receiver is unable to record object's current position for extended amounts of time, which results in a loss of sampling points of a trajectory. Moreover, an additional loss of sampling points could be introduced when the stops are removed from a trajectory. As a result, trajectories often contain *missing segments* that do not have any sampling points. A *missing segment* is a part of a trajectory that, according to a given GPS sampling rate and object's movement direction should be there, but is missing [32]. An example of such missing segment is shown in Figure 3.4.

Like in the case with stops, the presence of missing segments can also affect the clustering quality. For instance, both DTW and LCSS will be affected, because

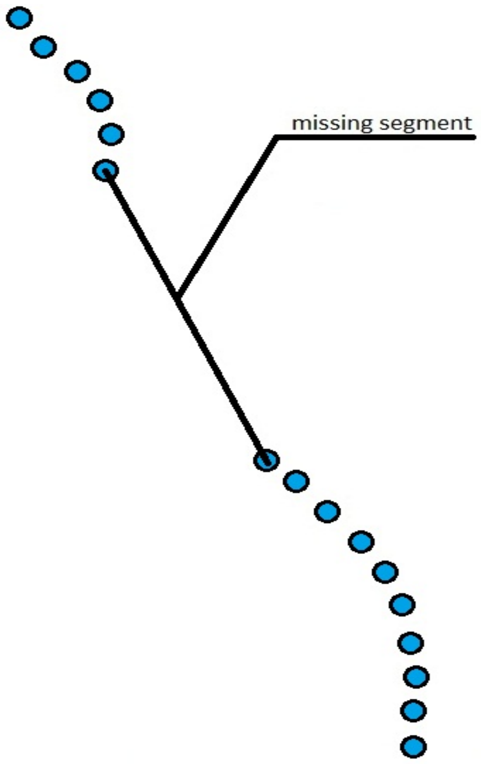


Figure 3.4: A missing segment on a trajectory

they compare trajectories in a piece-wise fashion. The absence of a considerable part of a trajectory will surely affect how the distances to other trajectories are calculated.

Let p_i and p_{i+1} be consecutive points on a trajectory with timestamps t_{p_i} and $t_{p_{i+1}}$, respectively. Also, let ϕ and ψ be the *interpolation* and *trajectory breaking* thresholds, respectively as well. Then, if $\psi > t_{p_{i+1}} - t_{p_i} > \phi$, the interpolation procedure is used to fill the missing segment with points. If, however, $t_{p_{i+1}} - t_{p_i} > \psi$, we will not interpolate this missing segment. Instead, we will break down the given trajectory into two separate subtrajectories, so that each endpoint of the missing segment would belong to a separate trajectory. The motivation behind this is that if the signal is missing for long durations of time compared to the usual sampling rate, it is often the case that the object is probably inside a building or some other place where there is no signal reception, i.e. the object reached its destination. Thus, we divide this trajectory into two parts according to the trajectory breaking threshold, which corresponds to the average amount of time without receiving any GPS signal

[32].

Suppose there is such a missing segment on a trajectory q. Then, let $p_0, p_1, ...p_i, ...p_{|q|}$ be the set of points of this trajectory and let points p_a and p_b be the endpoints of this missing segment such that $0 < a < b < |q|$. Using the Euclidean distance between p_a and p_b (denoted as $Dist(p_a, p_b)$), we can calculate the number of subsegments N that is needed to evenly distribute newly generated points across the whole missing segment [32]:

$$N = \left\lceil \frac{2kDist(p_a, p_b)}{\sum_{j=a-k}^{a-1} Dist(p_j, p_{j+1}) + \sum_{j=b}^{b+k-1} Dist(p_j, p_{j+1})} \right\rceil$$

where k represents how many points before p_a and after p_b are considered to calculate the needed number of subsegments N. The distance between newly generated points p_i and p_{i+1} (where $a < i < b$) is given as follows:

$$Dist(p_i, p_{i+1}) = \frac{Dist(p_a, p_b)}{N}$$

After the distance between generated consecutive points is found, we create $N - 1$ points starting from p_a towards p_b according to this distance. This way, the new points will be evenly spread across the missing segment.

The last step of the interpolation is to generate timestamps and speed information based on the calculated number of subsegments. Let t_{p_a} and t_{p_b} be the timestamps of points p_a and p_b. Then, the points generated between both endpoints will have the following timestamps:

$$t_{p_{a+i}} = t_{p_a} + i \cdot \frac{t_{p_b} - t_{p_a}}{N}$$

where $0 < i < N$ and $p_{a+i} < p_b$.

Note that this interpolation algorithm is applied not only when the missing segment is due to a lack of signal, but also when the missing segment is created after the stop detection and removal.

3.3 Outlier Detection

Atmospheric and ionospheric disturbances, signal blocking or limited satellite visibility can cause the imprecise recording of an object's positions by a GPS device [34, 61]. In the context of this thesis, such incorrectly positioned points are referred to as *outliers*. An example of such an outlier is shown in Figure 3.5. For instance, if the average speed during the whole trajectory duration was low and one observation shows that the speed at this moment of time was unusually high, then one can call this point an outlier.

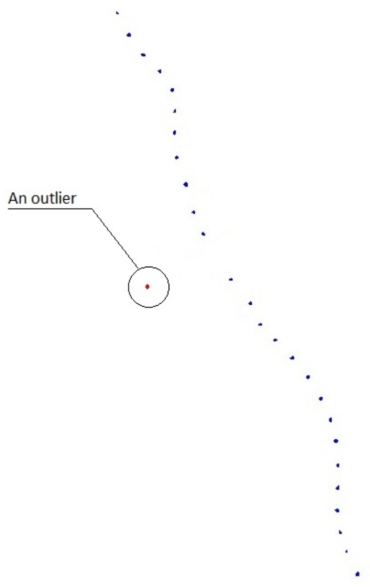


Figure 3.5: An example of an outlier

One of the problems of the existing distance functions (e.g. DTW) is that many of them cannot deal with data imprecisions [18]. The *Outlier Detection* step of the proposed framework is aimed to detect points that do not conform to the general behaviour of a trajectory and to eliminate them.

For the outlier detection in a trajectory, we use a statistical technique called *Three Sigma Rule*. According to Pukelsheim [49], outliers can be effectively identified using the mean and the standard deviation. In this work, we applied this rule on the calculated speeds to detect points that have unusually high speeds for the current trajectory. The idea behind this is to detect points that accelerated in an improbable manner compared to their average behaviour.

Let $p_0, p_1, ...p_i, ...p_{|t|}$ be the set of points on a trajectory t. Also, let s_{p_i} be the speed of a point p_i. Then, the mean of point speeds is defined as:

$$\mu = \frac{\sum_{i=0}^{|t|} s_{p_i}}{|t|}$$

The standard deviation of speeds is then defined as follows:

$$\sigma = \sqrt{\frac{\sum_{i=0}^{|t|} s_{p_i}^2}{|t|} - \mu}$$

After the mean and the standard deviation are found, we apply the *Three Sigma Rule* on the calculated speeds. Each point on a trajectory that satisfies the following condition is considered an outlier:

$$s_{p_i} > \mu + \alpha \cdot \sigma \tag{3.3.1}$$

Note, that the α parameter here is usually set to 3 according to the *Three Sigma Rule*. However, this parameter setting works only if the normal speed distribution is assumed. While this holds for most real world datasets that contain *either* pedestrian *or* vehicle movement [58, 19], when a dataset has *both* pedestrian and vehicle movement mixed, this speed distribution can be exponential. In fact, since both datasets used for our experiments observe both pedestrian and vehicle travel, they both exhibit the exponential distribution of speeds as shown in Figure 3.6. In Chapter 4, we experimentally determine the recommended value for α for the exponential distribution of speeds. The second assumption is that there are no several consecutive outliers. This assumption is also usually true in trajectory datasets, since the presence of two or more consecutive possible speed outliers usually means that the object actually travelled to those positions, i.e. those points, probably, are not outliers. In Chapter 4, this approach is evaluated by manually inserting outliers to the dataset and testing whether this technique is able to identify them.

3.4 Map Matching

When an accurate road network map is presented, our framework can be enhanced further by using the *map matching* technique. Given a set of GPS points and a set

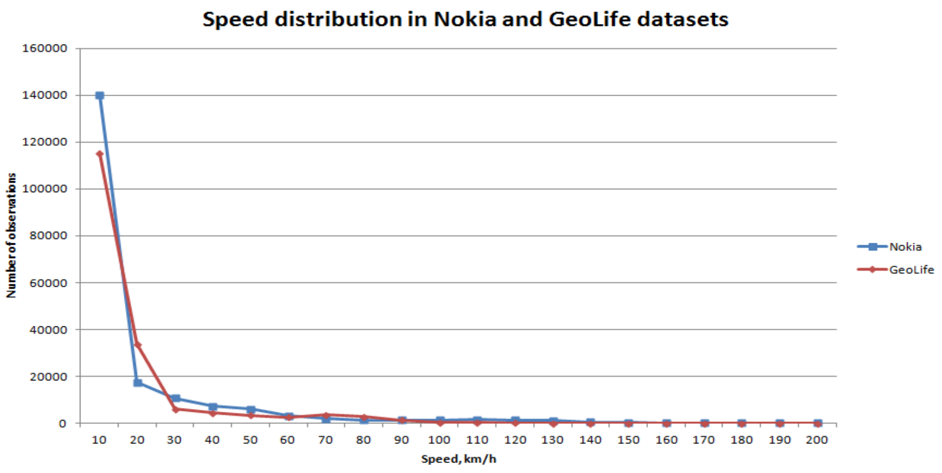


Figure 3.6: Speed distributions for Nokia and GeoLife datasets

of roads, *map matching* tries to align these points to the road segment by which an object actually travelled. Since road networks are usually more accurate than GPS readings, map matching allows to improve the accuracy of the observed point's accuracy.

In this work, we use an approach proposed in [50], which is based on the fuzzy logic matching. Unlike in binary logic, fuzzy logic uses multiple valued logic system. Instead of "true" or "false", fuzzy logic operates with values such as "true to some extent", "small", "high", "average" or real number values between zero and one [45].

In [50], fuzzy logic is used to quantify the set of rules given by an expert to determine the likelihood of matching a given point to a given road segment. For instance, let P be a point that needs to be matched, r be the candidate road segment, HD - the heading difference between P and r, PD - the perpendicular distance between P and r. Then, the algorithm proceeds as follows. First, four main input variables are "fuzzyfied", i.e. assigned a membership function as shown in Figure 3.7: the speed of the object v, the perpendicular distance PD, the heading difference HD and the horizontal dilution of precision ($HDOP$). The $HDOP$ parameter represents the confidence level of a reading based on the relative positions of satellites and a receiver [35]. Note, that the $HDOP$ parameter is optional. If this parameter is unavailable in a dataset, one may simply ignore the rules that are associated with

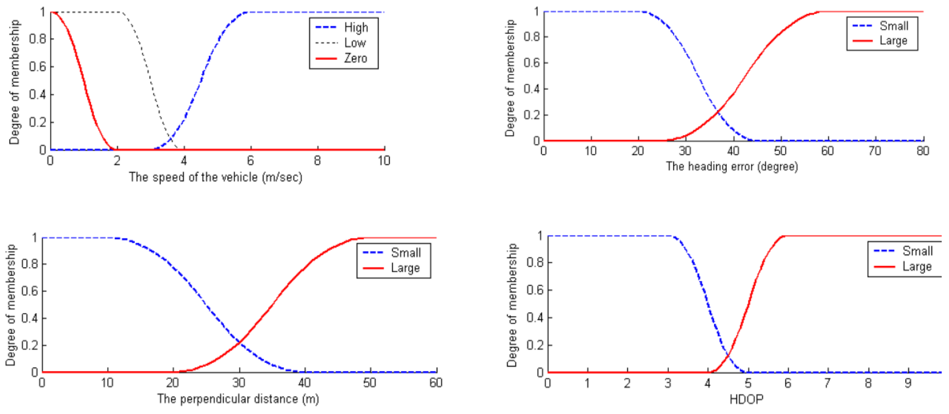


Figure 3.7: Fuzzyfication of the input parameters, taken from [50]

this parameter.

Then, for each rule in the set of defined rules, the likelihood L of matching P to r is quantified. The following set of rules is used in this phase of the algorithm [50]:

- If (v is HIGH) and (HD is SMALL) then (L is AVERAGE)
- If (v is HIGH) and (HD is LARGE) then (L is LOW)
- If ($HDOP$ is GOOD) and (PD is SMALL) then (L is AVERAGE)
- If ($HDOP$ is GOOD) and (PD is LONG) then (L is LOW)
- If (HD is SMALL) and (PD is SHORT) then (L is HIGH)
- If (HD is LARGE) and (PD is LONG) then (L is LOW)

Then, for each rule above, a degree of membership ω_i computed based on the respective membership functions. After that, the final matching likelihood Z, which shows how likely is that P matches r, is calculated as follows:

$$Z = \frac{\sum_{i=0}^{n} \omega_i \cdot z_i}{\sum_{i=0}^{n} \omega_i}$$

where n is the number of rules and z_i is a fuzzy constant the value of which depends on L:

$$z_i = \begin{cases} 10 & \text{if } L_i \text{ is LOW} \\ 50 & \text{if } L_i \text{ is AVERAGE} \\ 100 & \text{if } L_i \text{ is HIGH} \end{cases}$$

Finally, a point is matched to a road segment with the highest matching likelihood. In the above, we described only the first step of the approach proposed in [50], which uses only 4 parameters and 6 rules associated with them. Since the next steps of that map matching framework require routing and road direction information which are not always available, we will use only the first step of that approach in our framework. However, our experiments, which are presented in Chapter 4, show that applying just the first step of this method greatly improves data clustering quality.

Chapter 4

Experiments and Results

4.1 Experimental Setup

To evaluate our proposal, we implemented our framework using the Java programming language and applied it to Nokia's Lausanne Dataset [36] and Microsoft's GeoLife Trajectories Dataset [66, 67, 68]. The former dataset contains GPS trajectories of more than 200 users over the course of two years with a sampling rate of 10 seconds. The latter dataset was collected from 178 users over the course of four years with sampling rate varying between one to five seconds. This sampling rate applies to 91% of the trajectories.

For each dataset, a set of trajectories was produced as follows. Each user's initial raw trajectory was divided into smaller trajectories according to the trajectory breaking threshold (set to 180 seconds for both datasets), which was described in Section 3.2. Trajectories with the number of points less than ten were removed from the further processing, since they do not contribute to the clustering process. Due to virtual memory constraints during the generation of the distance function matrices, 4000 trajectories (users *002, 005, 007, 009, 010*) and 1000 trajectories (user *000*) with around 400000 points in total were obtained from Nokia's Lausanne and Microsoft's GeoLife datasets, respectively. We will further refer to these sets of produced trajectories as *raw* trajectory sets.

We then applied our framework to each raw trajectory set, which produced the respective *cleaned* trajectory sets. Next, both raw and cleaned trajectory sets were given as an input to the following trajectory clustering algorithms: DBSCAN clustering using DTW distance function, DBSCAN using LCSS, and TRACLUS. Finally, for each clustering algorithm we compared how our data cleaning framework affected the quality of the resulting clustering.

Since there is no "ground truth" available, the problem of quantitative evaluation of clustering itself becomes non trivial. To our knowledge, there is no widely acknowledged framework to evaluate the accuracy of resulted clusterings without using any external knowledge. In this work, we used a measure proposed in [37], called *QMeasure*, along with visual validation. In addition, we tried different orders of the steps of our framework to determine which order has the most impact on the clustering quality.

QMeasure is an internal clustering evaluation measure defined in [37], which represents the sum of the squared pairwise distances between the elements of one cluster (*TotalSSE*), while penalizing for incorrectly identified noise points (*NoisePenalty*):

$$QMeasure = TotalSSE + NoisePenalty = \\ = \sum_{i=1}^{|C|} \left(\frac{1}{2|C_i|} \sum_{x \in C_i} \sum_{y \in C_i} Dist(x,y)^2 \right) + \frac{1}{2|F|} \sum_{w \in F} \sum_{z \in F} Dist(w,z)^2$$

where C_i is a cluster in a set of clusters C, F is a set of noise trajectories and $Dist(x, y)$ is a distance between trajectories x and y, which can be either the DTW, LCSS or TRACLUS distance. Note that smaller *QMeasure* values imply "better" clustering. Since *QMeasure* is a purely internal clustering evaluation measure, visual validation was performed along with quantitative quality measurement.

4.2 Results

4.2.1 Clustering Results

For each clustering technique we tried several values of the required input parameters. The ranges of both Eps and $minPts$ ($minLns$ for TRACLUS) parameters for all clustering experiments were chosen based on the visual validation of the produced clusters. This parameter estimation technique is common in trajectory clustering approaches that use DBSCAN [60], [17]. Indeed, the parameters that were set to the values outside of the shown range did not result in meaningful clusterings; clusters became either too small (couple of lines), or too large (the whole set of trajectories is identified as one cluster).

Our experiments show that out of the three discussed clustering techniques, the DTW distance is shown to be the most sensitive to noise and stops. As Figures 4.1, 4.2, 4.3 and 4.4 suggest, the *QMeasure* decreases after applying the proposed

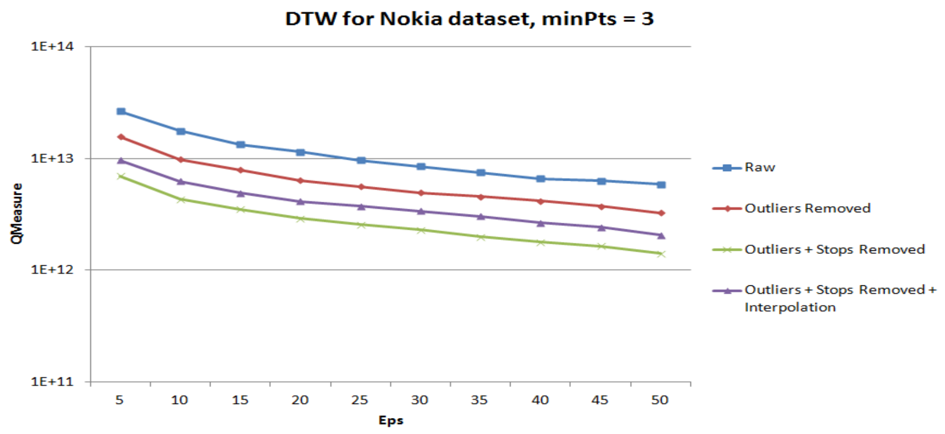


Figure 4.1: *QMeasure* for DTW with fixed *minPts* parameter for the Nokia dataset

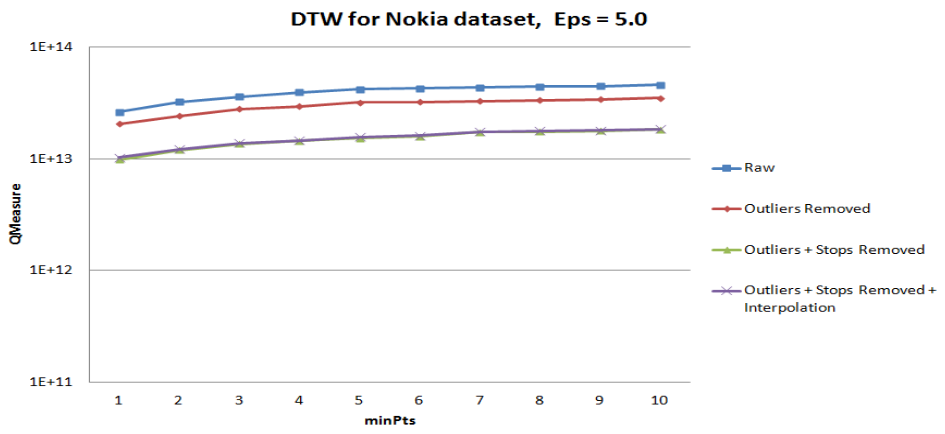


Figure 4.2: *QMeasure* for DTW with fixed *Eps* parameter for the Nokia dataset

cleaning framework. The overall decline of the *QMeasure* for Nokia dataset in Figure 4.1 means that with the increasing *Eps*, more trajectories are included into clusters and the number of noise trajectories decreases. Because the noise penalty (NP) is significantly higher than the sum of squared pairwise distances (TotalSSE), the *QMeasure* decreases with the larger *Eps* values. Conversely, in Figure 4.2, the larger is the value of *minPts*, the larger is the number of noise trajectories. Since the number of noise trajectories increments, the NP increases as well, which results in the ascending values of the *QMeasure*.

Both produced plots suggest that out of three steps of the proposed framework, the *Stop Detection* step had the most impact on clustering quality for the Nokia dataset. On average, when the clustering was performed using DTW distance function, after both *Outlier Detection* and *Stop Detection* steps, the *QMeasure* decreased by 62% compared to the clustering resulted from the raw data, whereas applying only the *Outlier Detection* step reduced the *QMeasure* by roughly 24%. This improvement was computed as follows:

$$Improvement = \frac{QMeasure_{cleaned} - QMeasure_{raw}}{QMeasure_{raw}} \cdot 100\%$$

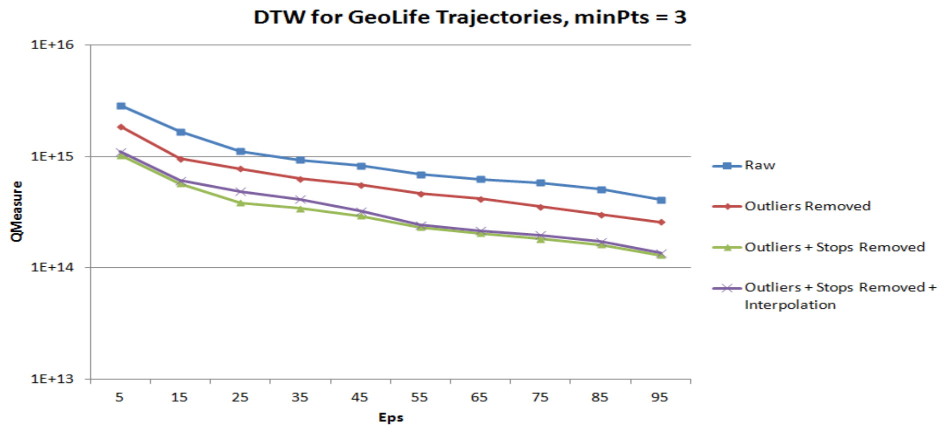


Figure 4.3: *QMeasure* for DTW with fixed *minPts* parameter for the GeoLife dataset

Applying our framework to the GeoLife dataset yielded results illustrated in Figures 4.3 and 4.4. The improvement in the *QMeasure* for this dataset after the *Outlier Detection* step was around 35%, whereas the combined *Outlier* and *Stop Detection* steps yielded a 64 % improvement when using DTW as a distance function. These results imply that, as expected, DTW is quite sensitive to outliers and stops. Hence, our framework was able to significantly improve the clustering results for this distance function.

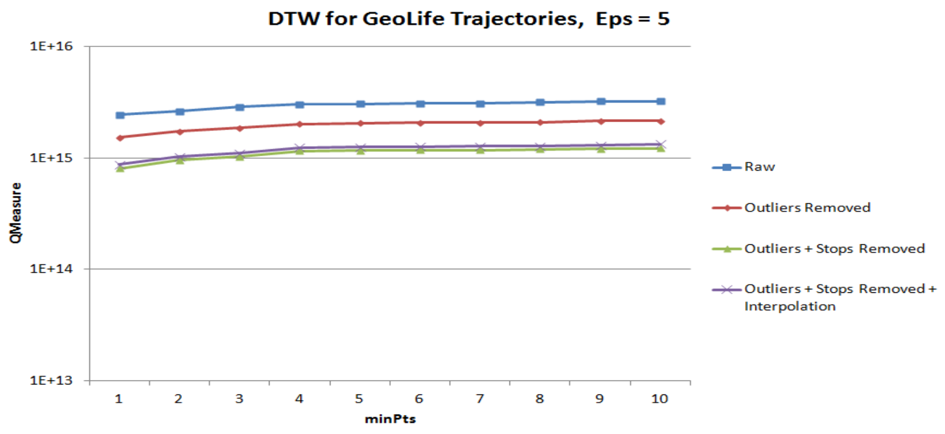


Figure 4.4: *QMeasure* for DTW with fixed *Eps* parameter for GeoLife dataset

Unfortunately, the *Interpolation* step did not result in the improvement of the clustering quality. Most of the times the improvement after this step was too small or, sometimes, caused actually an increase in the *QMeasure*. One of the possible reasons for this is that the used interpolation technique connected the endpoints of a missing segment by means of a straight line segment, i.e. did not take a trajectory's shape into account. For future work, we would like to investigate whether considering the shapes of the previous and the next trajectory segments actually improves clustering quality.

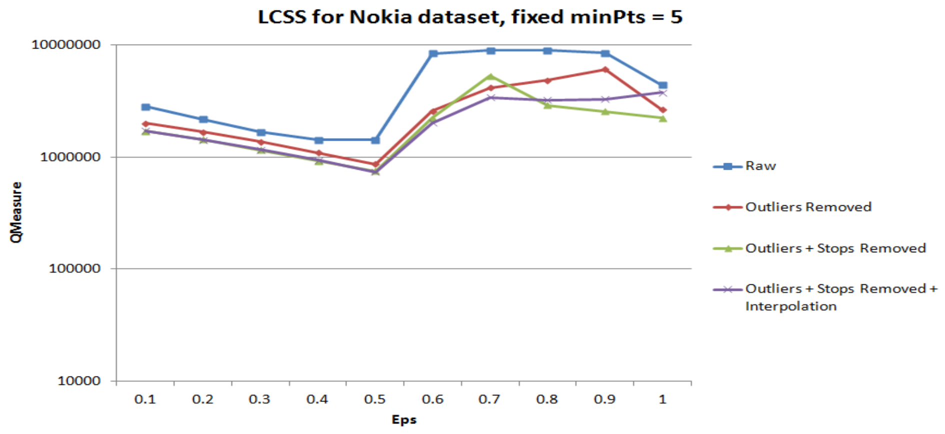


Figure 4.5: *QMeasure* for LCSS with fixed *minPts* parameter for the Nokia dataset

In Figures 4.5 and 4.6 clustering evaluation results are presented for LCSS distance function when applied to Nokia dataset. The decrease of the *QMeasure* on the visually meaningful settings ($Eps = 0.5$, $minPts = 5$) after the *Outlier Detection*

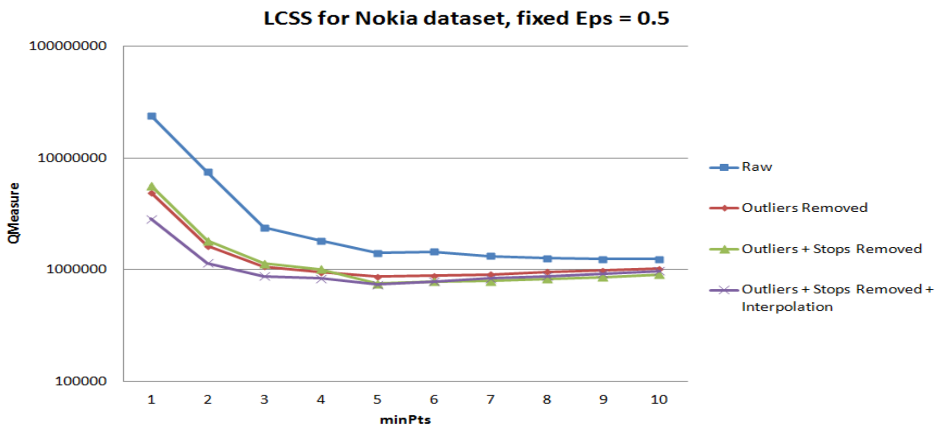


Figure 4.6: *QMeasure* for LCSS with fixed *Eps* parameter for the Nokia dataset

step was around 38% and approximately 45% after applying both *Stop Detection* and *Outlier Detection* steps.

We believe that the spike in Figure 4.5 resulted from the fact that LCSS uses a score instead of the actual distance (as in DTW) to determine the similarity between two trajectories (see Section 2.2.4). In essence, LCSS counts the number of points from two trajectories that have the smaller distance between them than a predefined distance threshold. Because of this score, as the Eps becomes larger, at one moment, noise trajectories rapidly become more dissimilar from the rest of the clusters. However, during both visually reasonable and unreasonable clusterings, the values of the *QMeasure* for the processed dataset were smaller compared to the raw dataset.

When using LCSS for clustering the GeoLife dataset, our framework was not able to improve the resulting clustering quality in a significant degree (Figures 4.7 and 4.8). The average improvement in the *QMeasure* after applying the framework was only around 1.5%. One of the reasons behind this is that LCSS was able to produce tighter clusters with the smaller number of the noise trajectories on the visually adequate parameters compared to DTW. In Figure 4.8, all the curves are smooth and plain, which shows that the clusters did not change much. The spikes in Figure 4.7 have the same nature as in Figure 4.5 - noise trajectories lie further apart from any cluster member trajectories in terms of LCSS distance. Nonetheless, on the visually adequate parameters ($Eps = 0.3$, $minPts = 5$) the improvement of the *QMeasure*

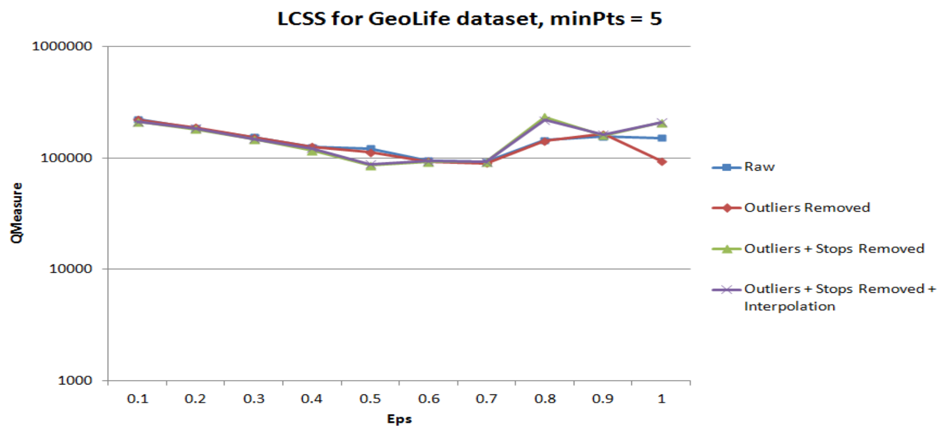


Figure 4.7: *QMeasure* for LCSS with fixed *minPts* parameter for the GeoLife dataset

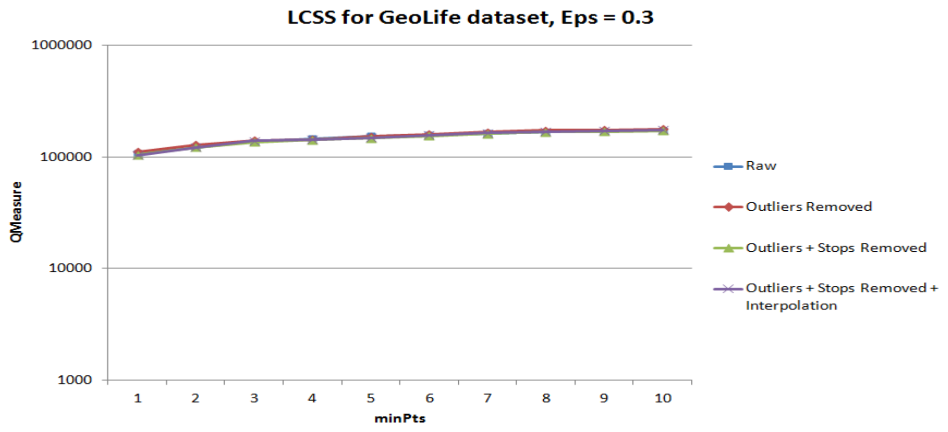


Figure 4.8: *QMeasure* for LCSS with fixed *Eps* parameter for the GeoLife dataset

was close to 4%. This shows that our framework will result in a better clustering when needed and will not interfere with a clustering if it is already "good".

Finally, according to our experiments, as Figures 4.9 and 4.10 suggest, TRACLUS trajectory clustering framework actually handles outliers and missing segments in an equal degree to our *Outlier Detection* and *Interpolation* steps. At the same time, our *Stop Detection* step was able to improve the *QMeasure* roughly by 23% on average on the Nokia dataset.

As when clustering with DTW, as the *minLns* parameter increments, since clusters contain less trajectories and the number of noise trajectories increases, the *QMeasure* ascends (Figure 4.9). On the contrary, as the Eps parameter increases (Figure 4.10), the *QMeasure* diminishes because of the decline of the noise penalty sum.

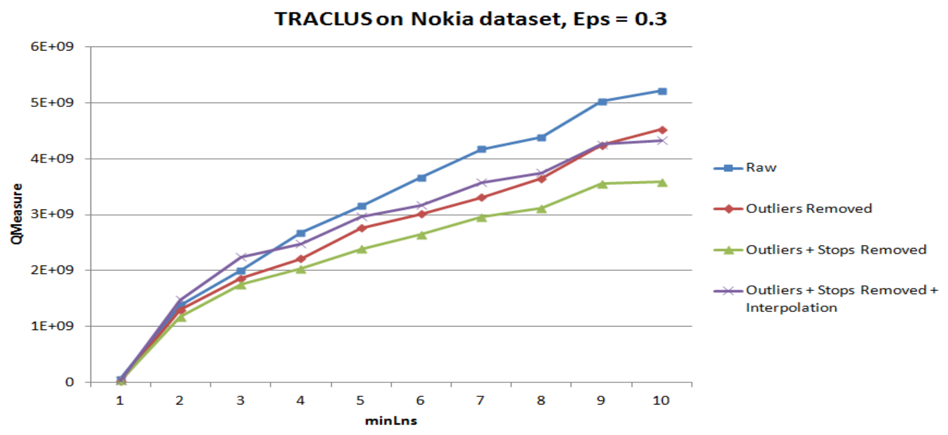


Figure 4.9: *QMeasure* for TRACLUS with fixed *Eps* parameter for the Nokia dataset

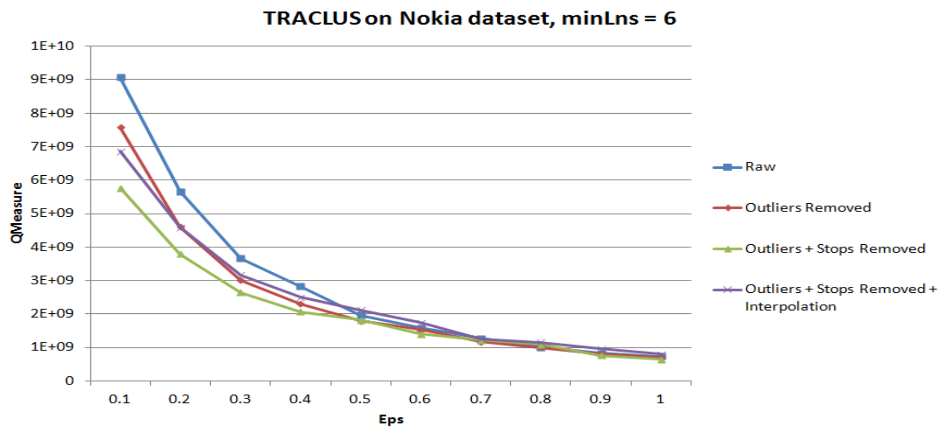


Figure 4.10: *QMeasure* for TRACLUS with fixed *minLns* parameter for the Nokia dataset

The above applies to the GeoLife dataset as well (Figures 4.11 and Figures 4.12), where the *QMeasure* decreased by 9% after applying our framework.

One of the reasons of why our trajectory cleaning framework improved clustering quality after TRACLUS to a lesser degree compared to DTW and LCSS is that TRACLUS uses a crude trajectory approximation technique. In TRACLUS, in order to find clusters of subtrajectories, trajectories are approximated using a set of characteristic points. Figure 4.13 presents an example of such rough trajectory approximation. Lines resulted from TRACLUS's trajectory approximation effectively oversimplify trajectories by not taking the curves of the trajectories into account. While this is useful when finding frequently travelled vectors of the observed ob-

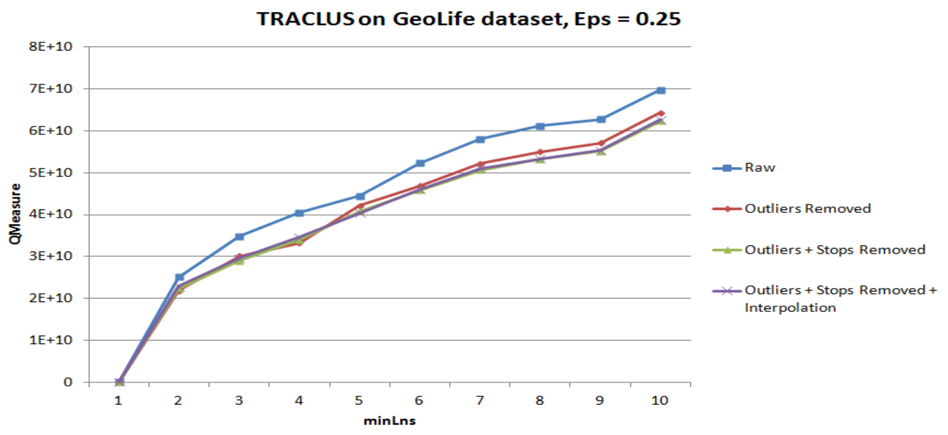


Figure 4.11: *QMeasure* for TRACLUS with fixed *Eps* parameter for the GeoLife dataset

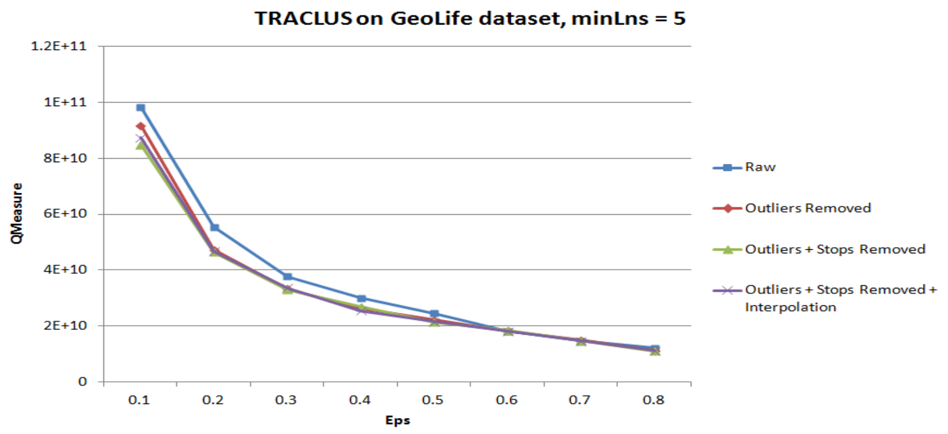


Figure 4.12: *QMeasure* for TRACLUS with fixed *minLns* parameter for the GeoLife dataset

jects, finding the exact positions of objects that belong to each movement cluster becomes much more complicated, if not impossible.

In addition, the impact of changing the order in which steps of the proposed framework are executed was evaluated. In particular, two main modes were compared to each other:

- *Mode 1:* Outlier Detection, Stop Detection, Interpolation (in this order)
- *Mode 2:* Stop Detection, Interpolation, Outlier Detection (in this order)

The number of modes is limited because the *Interpolation step* should be executed after *Stop Detection*, to fill in the missing segments resulted from *Stop Detection*.

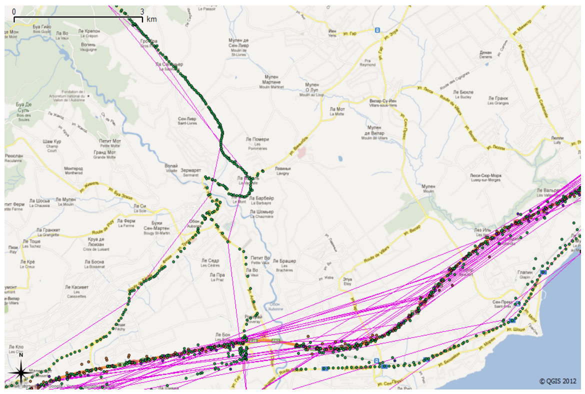


Figure 4.13: Examples of crude trajectory approximation by TRACLUS

While *Mode 1* was tested earlier (Figures 4.1 and 4.2), we tested *Mode 2* with the same parameters and compared the *QMeasure* for both outputs.

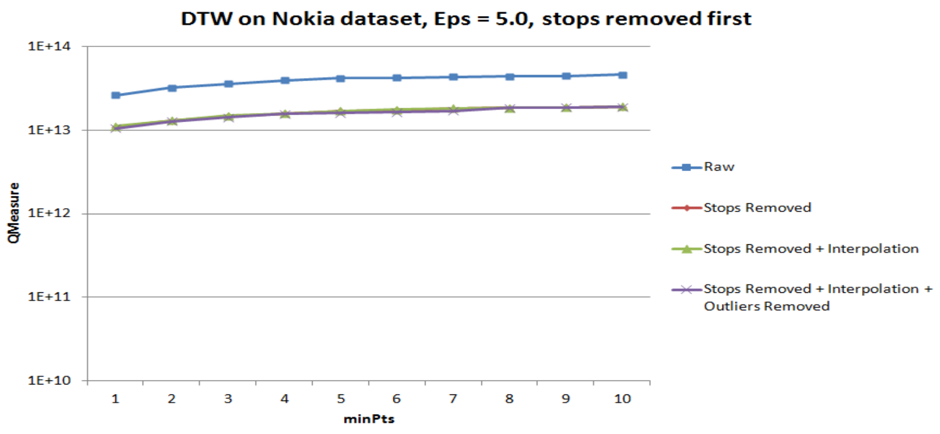


Figure 4.14: *QMeasure* for DTW with fixed *Eps* parameter for the Nokia dataset (Mode 2)

Results for *Mode 2*, which are presented in Figures 4.14, 4.15, 4.16 and 4.17 reveal that this mode yielded approximately 3% less improvement in the *QMeasure* than *Mode 1* for Nokia dataset and approximately 2% less improvement for GeoLife dataset. In addition, applying the *Interpolation step* after the *Outlier Detection* step helped to recover incorrectly identified outliers, that resulted from too rapid acceleration. Therefore, to achieve the best performance, one needs to execute steps of the framework in the following order:

1. Outlier Detection

2. Stop Detection

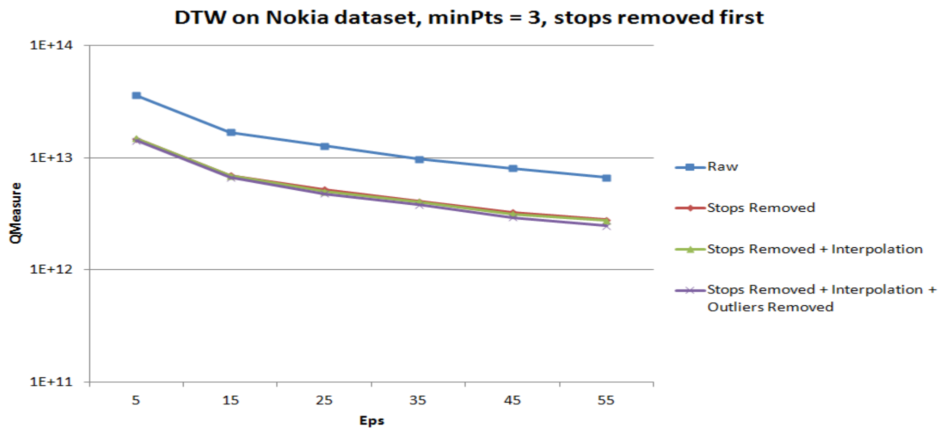


Figure 4.15: *QMeasure* for DTW with fixed *minPts* parameter for the Nokia dataset (Mode 2)

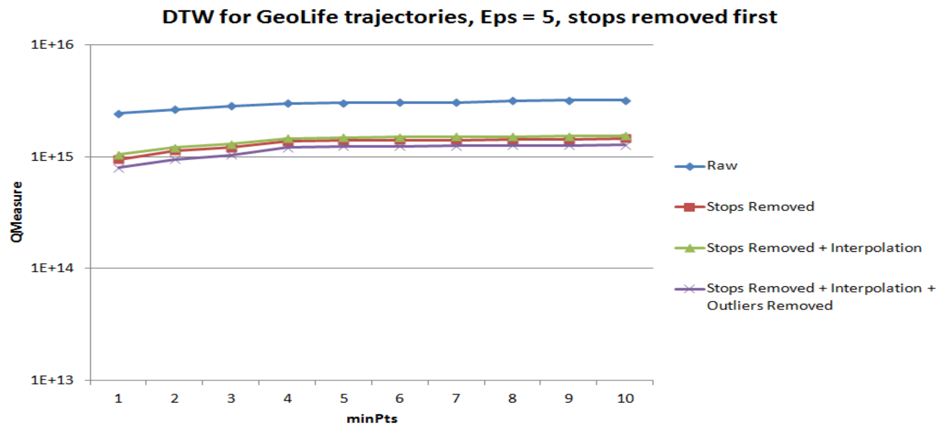


Figure 4.16: *QMeasure* for DTW with fixed *Eps* parameter for the GeoLife dataset (Mode 2)

3. Interpolation

Finally, we determined how the *Map Matching* step affects the clustering. First, an example of how map matching "tightens" potential trajectory clusters is illustrated in Figure 4.18. Since points are matched to a road segment that most likely corresponds to those points in the real world, trajectories that were matched to this road segment will be more similar to each other. Consequently, with the increasing similarity between truly close trajectories, the resulting clustering quality improves as well.

Results for clustering with DTW distance function and DBSCAN support this claim. On average, the *Map Matching* step improved the *QMeasure* approximately by

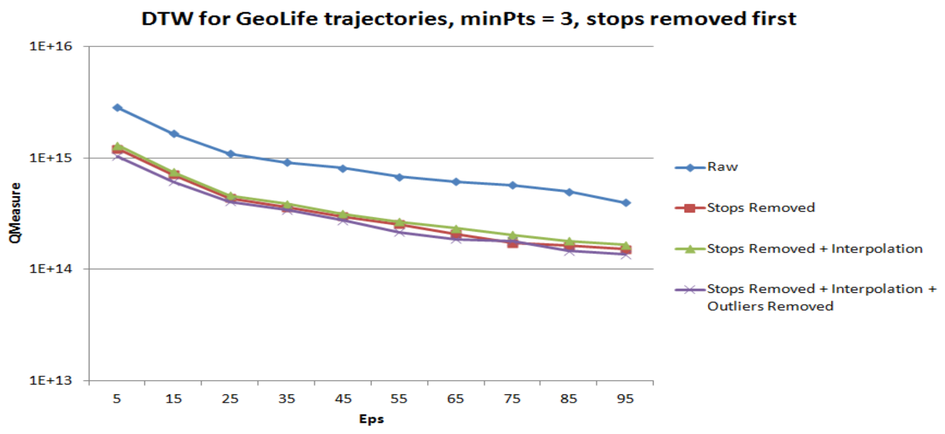


Figure 4.17: *QMeasure* for DTW with fixed *minPts* parameter for the GeoLife dataset (Mode 2)

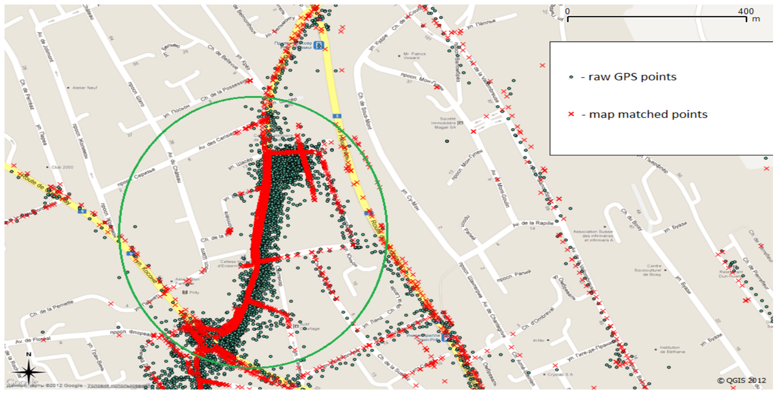


Figure 4.18: GPS points before and after the *Map Matching* step

98% on Nokia dataset (Figures 4.19 and 4.20) and approximately 100% on GeoLife dataset (Figures 4.21 and 4.22). Noteworthy, the curves resulted after the *Map Matching* step seem much smoother compared to the other steps of the framework. This is because the *Map Matching* step does not remove points like other steps, but merely improves the accuracy of their coordinates with respect to the given road network. Note, however, that the *Map Matching* step relies on the accuracy of the used road network - the more accurate is the road network, the more accurate is the matching of points to road segments.

4.2.2 Outlier Detection Evaluation

Next, we present the evaluation results for the *Outlier Detection* step separately. Since there is no training trajectory data with labelled outliers available, we tested

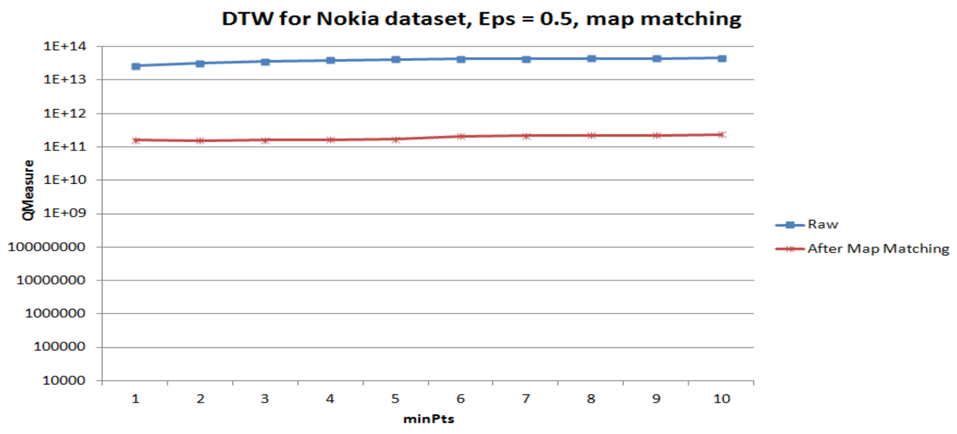


Figure 4.19: *QMeasure* for DTW after the *Map Matching* step for the Nokia dataset, fixed *Eps*

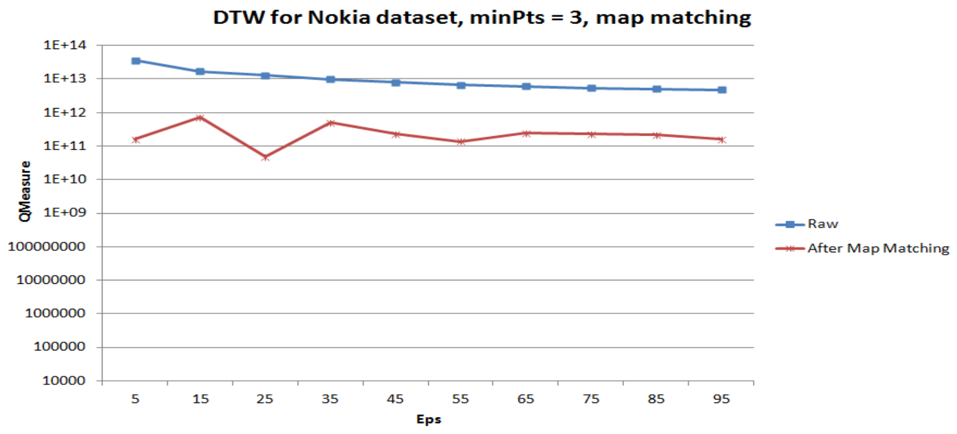


Figure 4.20: *QMeasure* for DTW after the *Map Matching step* for the Nokia dataset, fixed *minPts*

this statistical outlier detection technique by manually inserting outliers into the real world data. First, 10 trajectories with 917 points were taken from the dataset. Then, we manually labelled existing outliers and manually inserted new outliers so that the number of outliers would constitute the 5% of the total number of points. This 5% added outlier ratio is typically used for outlier detections studies that involve simulations [31, 55]. As in the example shown in Figure 4.23, all the outliers were inserted at reasonable distances from their actual trajectories. These distances were chosen such that a human expert would identify those points as outliers.

Then, according to the labels of manually inserted outliers, points were classified into four categories:

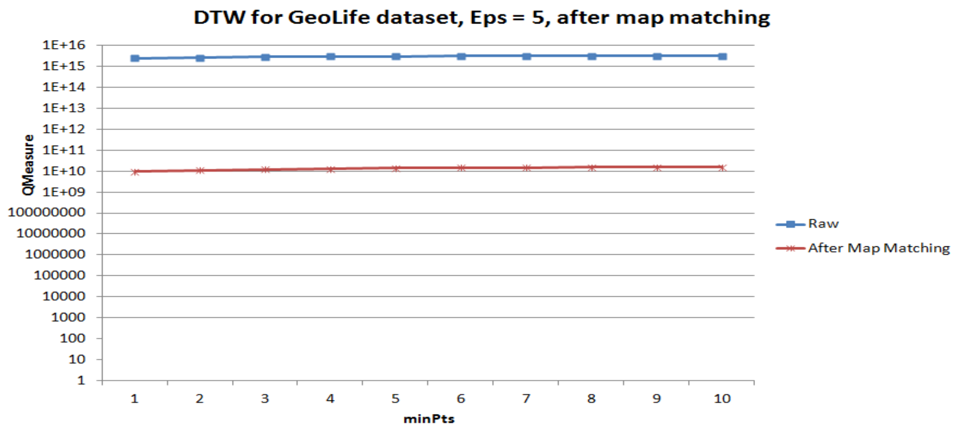


Figure 4.21: *QMeasure* for DTW after the *Map Matching* step for the GeoLife dataset, fixed *Eps*

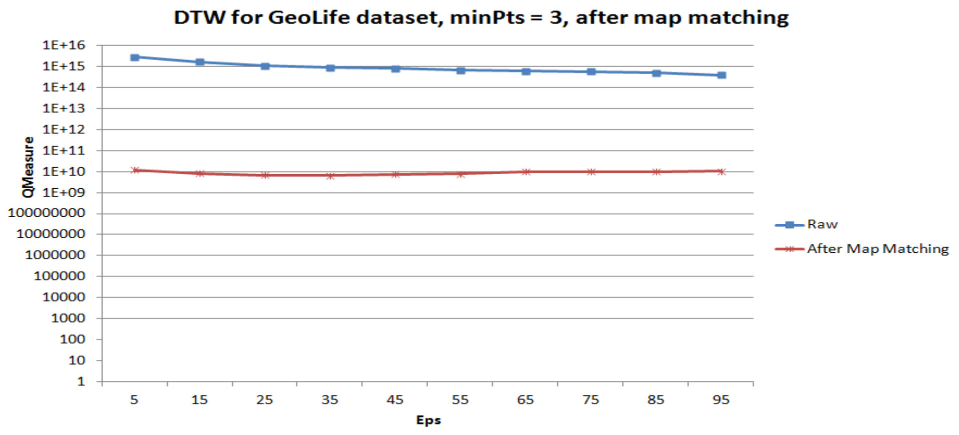


Figure 4.22: *QMeasure* for DTW after the *Map Matching step* for the GeoLife dataset, fixed *minPts*

- *True Positives(TP)* is the number of correctly identified outlier points
- *True Negatives(TN)* is the number of points that were correctly classified as non-outliers
- *False Positives(FP)* is the number of non-outliers that were incorrectly classified as outliers
- *False Negatives(FN)* is the number of points that were initially labelled as outliers, but incorrectly identified as non-outliers

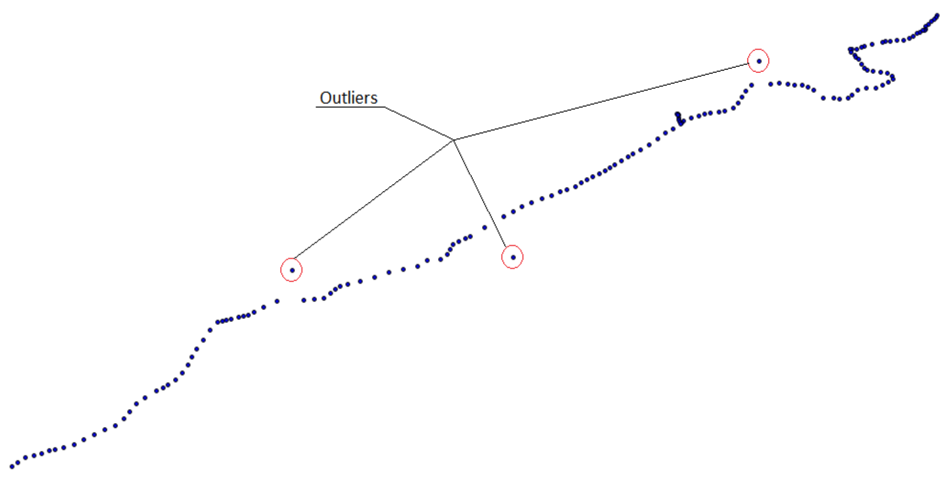


Figure 4.23: Manually inserted outliers

As discussed in Section 3.3, we detect outliers with unusually high speeds based on the following condition:

$$s_{p_i} > \mu + \alpha \cdot \sigma$$

where s_{p_i} is the speed of a point, μ and σ are the arithmetic mean and the standard deviation of a set of all speeds of a trajectory, respectively and α is a parameter that determines how many standard deviations from the mean should a point's speed be to be considered an outlier.

Then, we tried a range of values for α to determine such value for this parameter that would yield the highest *F-measure* [52], which is classification quality metric from information retrieval domain. The *F-measure* is usually used when the number of instances of a class is rare [43]. Since, in our case, the number of instances of an outlier class is rare (5%), we decided to choose the *F-measure* as the performance evaluation metric for the outlier detection. Note, that the *F-measure* varies between 0 and 1, where 0 means that all the points were classified incorrectly, while a *F-measure* of 1 means the opposite - the algorithm found all the outliers and did not classify any non-outliers as outliers.

The *F-measure* is the harmonic mean of the precision and the recall, where precision represents the fraction of outliers among all points classified as outliers and recall shows what fraction of points that were classified as outliers are actually out-

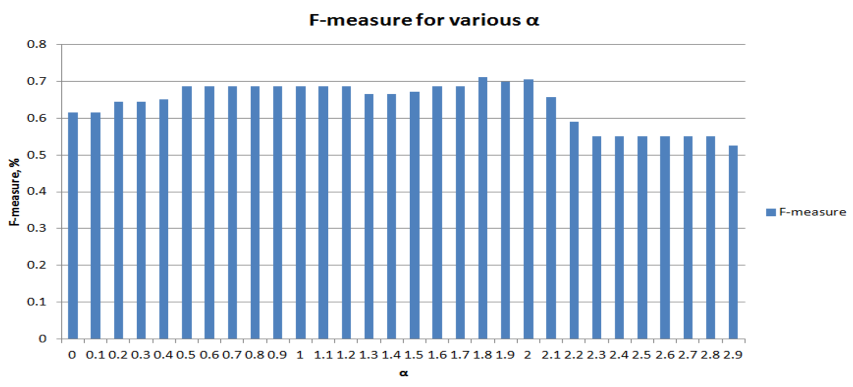


Figure 4.24: The *F-measure* for various α settings for outlier detection

liers. Then, the *F-measure* is calculated as follows [47]:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

As shown in Figure 4.24, in our experiments, the largest *F-measure* was equal to 0.71 and was achieved when $\alpha = 1.8$. To our knowledge, there are no studies that conduct such outlier detection experiments on trajectory datasets. However, a study on comparison of the general outlier detection techniques show that our values of the *F-measure* are comparable to the *F-measures* of the general outlier detection methods (0.4 - 0.82) [22].

4.3 Experiments and Results Conclusion

In this chapter, we presented the evaluation results for the proposed trajectory cleaning framework. Our framework has shown to improve the clustering quality when the DTW and LCSS distance functions are used with the DBSCAN. On the visually optimal settings, applying the framework led to the decrease of the clustering quality measure *QMeasure*. When using TRACLUS, the improvement of the clustering quality was less compared to DTW and LCSS. However, this lack of improvement is explained by that algorithm's simplifying trajectory approximation.

We also show, that when an accurate road network map is available, the quality of clusters could be greatly enhanced using the map matching technique. In both used datasets, the improvement in the *QMeasure* was around 100% when using this technique.

Finally, we evaluate the *Outlier Detection* step by manually inserting outliers and measuring the classification quality of the statistical outlier detection algorithm that we used in this step. Our results show that this classification quality is comparable to the most accurate algorithms in the general outlier detection domain.

Chapter 5

Conclusions and Future Work

Trajectory clustering is a popular data mining task which consists of finding paths that were travelled frequently. This pattern mining technique finds its application in a wide variety of areas as urban planning, climatology, medicine and economic forecasting [25]. However, low trajectory data quality can significantly decrease the accuracy of the existing clustering algorithms. Therefore, in this work, we have proposed a data cleaning framework for trajectory clustering, which consists of four main steps: *Outlier Detection, Stop Detection, Interpolation* and *Map Matching*. By applying this framework to a trajectory dataset, one can:

- detect and remove outliers
- identify and remove gatherings of points without movement that do not contribute to the trajectory clustering process (stops)
- fill in gaps in a trajectory that were resulted from a missing signal
- in case when an accurate road network map is present, improve points' accuracy by matching them to appropriate road segments

The presented framework has several features that are different from other trajectory data cleaning frameworks that were proposed before. First, some of the previous approaches required to have an additional GPS information (e.g. the number of visible satellites), while our framework suffices to have only longitude, latitude and time stamp readings. Second, even though other frameworks use some of the techniques similar to ours, none of them combine all these steps together. Finally, to our knowledge, only our framework was evaluated using popular clustering techniques. We evaluated our framework using two trajectory datasets (Nokia MDC and GeoLife Trajectories), two distance functions (DTW and LCSS) and two clustering algorithms (DBSCAN and TRACLUS), the latter of which is designed specifically for trajectory clustering. As a quality measure for the comparison between the resulted clusterings we used the *QMeasure*, which represents the difference between the sums of squared pairwise distances of clusters' members and noise. Our experiments show, that applying our framework consistently improves the resulting clustering quality. In addition, the effects of using different combinations of framework steps were investigated.

For future work, we would like to investigate how to achieve the following:

- improve the *Interpolation* step. One research direction could be to complete the missing segments using the similar subtrajectories of nearby trajectories. For instance, if we know that three persons already passed the location where the fourth person did not receive a signal, we could use this information to interpolate the resulted missing segment. Another idea would be to take into account a trajectory's shape that follows and precedes the missing segment.

- improve the trajectory splitting mechanism using a trajectory's both geometric and temporal features to extract subtrajectories

- take into account the intersecting trajectories during the *Stop Detection*

- conduct experiments with other distance functions and clustering algorithms

Bibliography

[1] S. S. Abdalmogith and R. M. Harrison. The Use of Trajectory Cluster Analysis to Examine the Long-Range Transport of Secondary Inorganic Aerosol in the UK. *Atmospheric Environment Journal*, 39:6686–6695, 2005.

[2] E. Akleman and J. Chen. Generalized Distance Functions. In *Proceedings of the International Conference on Shape Modeling*, pages 72–79, 1999.

[3] J. Alon, S. Sclaroff, G. Kollios, and V. Pavlovic. Discovering Clusters in Motion Time-Series Data. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, volume 1, pages 375–381, 2003.

[4] L .O. Alvares, G. Oliveira, C. A. Heuser, and V. Bogorny. A framework for trajectory data preprocessing for data mining. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, pages 698–702, 2009.

[5] G. Andrienko, N. Andrienko, S. Rinzivillo, M. Nanni, D. Pedreschi, and F. Giannotti. Interactive visual clustering of large collections of trajectories. In *IEEE Symposium on Visual Analytics Science and Technology*, pages 3 –10, 2009.

[6] G. L. Andrienko and N. V. Andrienko. Interactive Cluster Analysis of Diverse Types of Spatiotemporal Data. *SigKDD Explorations Journal*, 11:19–28, 2009.

[7] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander. OPTICS: Ordering Points To Identify the Clustering Structure. In *Proceedings of the International Conference on Management of Data*, pages 49–60, 1999.

[8] G. Antonini and J. P. Thiran. Trajectories Clustering in ICA Space: an Application to Automatic Counting of Pedestrians in Video Sequences. In *Proceedings of the Conference on Advanced Concepts for Intelligent Vision Systems*, 2004.

[9] S. Athavale and N. Sao. Data mining on moving object trajectories. *International Journal of Computer Science Engineering and Technology*, 2:1040–1042, 2012.

[10] F. I. Bashir, A. A. Khokhar, and D. Schonfeld. Object Trajectory-Based Activity Classification and Recognition Using Hidden Markov Models. *IEEE Transactions on Image Processing Journal*, 16:1912–1919, 2007.

[11] P. Berkhin. Survey of Clustering Data Mining Techniques. 2002. Accrue Software, San Jose, California - Technical Report, available at http://tsrenderer.googlecode.com/svn-history/r37/trunk/doc/papers/10.1.1.71.1599.pdf.

[12] D. J. Berndt and J. Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In *Proceedings of AAAI Workshop on Knowledge Discovery in Databases*, pages 359 – 370, 1994.

[13] J. C. Bezdek, R. Ehrlich, and W. Full. FCM: The Fuzzy C-means Clustering Algorithm. *Journal of Computers and Geosciences*, 10:191–203, 1984.

[14] G. Blewitt. An Automatic Editing Algorithm for GPS data. *Geophysical Research Letters Journal*, 17:199–202, 1990.

[15] D. Buzan, S. Sclaroff, and G. Kollios. Extraction and Clustering of Motion Trajectories in Video. In *Proceedings of the International Conference on Pattern Recognition*, volume 2, pages 521–524, 2004.

[16] I. Cavar, H. Markovic, and H. Gold. GPS Vehicles Tracks Data Cleansing Methodology. In *Proceedings of the International Conference on Traffic Science*. Available at: http://venera.fpz.hr/publications/ICTS%202006.pdf.

[17] C. Chang and B. Zhou. Multi-granularity Visualization of Trajectory Clusters Using Sub-trajectory Clustering. In *Proceedings of the IEEE International Conference on Data Mining*, pages 577–582, 2009.

[18] L. Chen, M. T. Ozsu, and V. Oria. Robust and Fast Similarity Search for Moving Object Trajectories. In *Proceedings of the International Conference on Management of Data*, pages 491–502, 2005.

[19] W. Daamen and S. P. Hoogendoorn. Free speed distributions based on empirical data in different traffic conditions. In *Proceedings of the International Conference of Pedestrian and Evacuation Dynamics*, pages 13–25. 2007.

[20] I. Davidson. Understanding K-Means Non-hierarchical Clustering. 2002. Technical Report 02-2. Albany: State University of New York. Available at http://www.cs.albany.edu/ davidson/courses/CSI635/UnderstandingK-MeansClustering.pdf.

[21] M. M. Deza and E. Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009.

[22] H. J. Escalante. A Comparison of Outlier Detection Algorithms for Machine Learning. In *Proceedings of the International Conference on Programming and Software*, pages 228–237, 2005.

[23] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on KDD*, pages 226–231, 1996.

[24] S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 63 – 72, 1999.

[25] F. Giannotti. Mobility, data mining and privacy understanding human movement patterns from trajectory data. In *Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management*, volume 1, pages 4–5, 2011.

[26] F. Giannotti and D. Pedreschi. *Mobility, Data Mining and Privacy - Geographic Knowledge Discovery*. Springer, 2008.

[27] S. Guha, R. Rastogi, and K. Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proceedings of the International Conference on Management of Data*, pages 73–84, 1998.

[28] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. San Francisco, California: Morgan Kaufmann Publishers, 2000.

[29] H. Haniu, N. Komori, N. Takemori, A. Singh, J. D. Ash, and H. Matsumoto. Proteomic Trajectory Mapping of Biological Transformation: Application to Developmental Mouse Retina. *Proteomics Journal*, 6:3251–3261, 2006.

[30] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin. The Elements of Statistical Learning: Data Mining, Inference and Prediction. *Mathematical Intelligencer Journal*, 27:83–85, 2005.

[31] P. S. Horn, L. Feng, Y. Li, and A. J. Pesce. Effect of Outliers and Nonhealthy Individuals on Reference Interval Estimation. *Clinical Chemistry Journal*, 47:2137–2145, 2001.

[32] A. Idrissov and M. A. Nascimento. A Trajectory Cleaning Framework for Trajectory Clustering. In *Nokia Mobile Data Challenge Workshop*, 2012. Available at: research.nokia.com/files/public/mdc-final225-idrissov.pdf.

[33] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Englewood Hills, New Jersey: Prentice-Hall, 1988.

[34] J. Jun, R. Guensler, and J. Ogle. Smoothing Methods to Minimize Impact of Global Positioning System Random Error on Travel Distance, Speed, and Acceleration Profile Estimates. *Transportation Research Record Journal*, 1972:141–150, 2006.

[35] R.B. Langley. Dilution of precision. *GPS World*, 10:52–59, 1999.

[36] J. K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornetr, T. Do, O. Dousse, J. Eberle, and M. Miettinen. The mobile data challenge: Big data for mobile computing research. In *Mobile Data Challenge by Nokia Workshop, in conjunction with International Conference on Pervasive Computing*, 2012.

[37] J. Lee, J. Han, and K. Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 593–604, 2007.

[38] T. W. Liao. Clustering of Time Series Data - a Survey. *Pattern Recognition Journal*, 38:1857–1874, 2005.

[39] L. Xu Liu, J. T. Song, B. Guan, Z. X. Wu, and K. J. He. Tra-DBScan: A Algorithm of Clustering Trajectories. *Journal of Applied Mechanics and Materials*, 121–126:4875–4879, 2012.

[40] S. P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–136, 1982.

[41] A. Moreira, M. Y. Santos, M. Wachowicz, and D. Orellana. The impact of data quality in the context of pedestrian movement analysis. In M. Painho, M. Y. Santos, H. Pundt, W. Cartwright, G. Gartner, L. Meng, and M. P. Peterson, editors, *Geospatial Thinking*, volume 0 of *Lecture Notes in Geoinformation and Cartography*, pages 61–78. Springer Berlin Heidelberg, 2010.

[42] B. Morris and M. M. Trivedi. Learning Trajectory Patterns by Clustering: Experimental Studies and Comparative Evaluation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 312–319, 2009.

[43] Y. Nan, K. M. Chai, W. S. Lee, and H. L. Chieu. Optimizing f-measure: A tale of two approaches. *ICML*, 2012. Available at: http://www.comp.nus.edu.sg/ leews/publications/fscore.pdf.

[44] M. Nanni and D. Pedreschi. Time-focused Clustering of Trajectories of Moving Objects. *Journal of Intelligent Information Systems*, 27:267–289, 2006.

[45] V. Novak, I. Perfilieva, and J. Mockor. Mathematical Principles of Fuzzy Logic. *The Kluwer International Series in Engineering and Computer Science*, 517, 1999.

[46] J. Ogle, R. Guensler, W. Bachman, M. Koutsak, and J. Wolf. Accuracy of Global Positioning System for Determining Driver Performance Parameters. *Transportation Research Record Journal*, 1818:12–24, 2002.

[47] D. L. Olson and D. Delen. *Advanced Data Mining Techniques*. Springer Verlag, 2008.

[48] A. T. Palma, V. Bogorny, B. Kuijpers, and L. O. Alvares. A clustering-based approach for discovering interesting places in trajectories. In *Proceedings of the ACM Symposium on Applied computing*, pages 863–868, 2008.

[49] F. Pukelsheim. The three sigma rule. *American Statistician Journal*, 48:88–91, 1992.

[50] M. Quddus, R. Noland, and W. Ochieng. A High Accuracy Fuzzy Logic Based Map Matching Algorithm for Road Transport. *Journal of Intelligent Transportation Systems*, 10:103–115, 2006.

[51] B. Rama, P. Jayashree, and S. Jiwani. A survey on clustering: Current status and challenging issues. *International Journal on Computer Science and Engineering*, 1:2976–2980, 2010.

[52] C. J. Van Rijsbergen. Foundation of Evaluation. *Journal of Documentation*, 30:365–373, 1974.

[53] T. Schreck, J. Bernard, T. Tekusova, and J. Kohlhammer. Visual Cluster Analysis of Trajectory Data with Interactive Kohonen Maps. In *IEEE Symposium on Visual Analytics Science and Technology*, pages 3–10, 2008.

[54] N. Schuessler and K. W. Axhausen. Processing GPS Raw Data Without Additional Information. paper presented at the 88th Annual Meeting of the Transportation Research Board, 2009.

[55] W. Stacklies and H. Redestig. Handling of Data Containing Outliers. 2007. Available at http://rss.acs.unt.edu/Rdoc/library/pcaMethods/ doc/outliers.pdf.

[56] P. Stopher, Q. Jiang, and C. Fitzgerald. Processing gps data from travel surveys. paper presented at the 28th Australasian Transport Research Forum, 2005.

[57] P. R. Stopher and C. C. Stecher. *Travel Survey Methods: Quality and Future Directions*. Oxford: Elsevier, 2006.

[58] R. E. Turochy and R. Sivanandan. Effectiveness of Unmanned Radar as a Speed Control Technique in Freeway Work Zones. Presented at the 77th Annual Meeting, Transportation Research Board, Washington, D.C., 1998.

[59] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings of the International Conference on Data Engineering*, pages 673 –684, 2002.

[60] Y. Wang, Q. Han, and H. Pan. A clustering scheme for trajectories in road networks. In Y. Wu, editor, *Proceedings of the 2009 3rd International Conference on Teaching and Computational Science*, volume 117 of *Advances in Intelligent and Soft Computing*, pages 11–18. Springer Berlin Heidelberg, 2012.

[61] J. Wolf, S. Hallmark, M. Oliveira, R. Guensler, and W. Sarasua. Accuracy Issues with Route Choice Data Collection by Using Global Positioning System. *Transportation Research Record Journal*, 1660:66–74, 1999.

[62] Y. Yanagisawa and T. Satoh. Clustering Multidimensional Trajectories based on Shape and Velocity. In *Proceedings of the International Conference on Data Engineering*, 2006.

[63] J. Yin and Q. Yang. Integrating Hidden Markov Models and Spectral Analysis for Sensory Time Series Clustering. In *Proceedings of the IEEE International Conference on Data Mining*, pages 506–513, 2005.

[64] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the International Conference on Management of Data*, pages 103–114, 1996.

[65] Z. Zhang, K. Huang, and T. Tan. Comparison of Similarity Measures for Trajectory Clustering in Outdoor Surveillance Scenes. In *Proceedings of the International Conference on Pattern Recognition*, volume 3, pages 1135–1138, 2006.

[66] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Ma. Understanding Mobility Based on GPS Data. In *Proceedings of the International Conference on Ubiquitous Computing/Handheld and Ubiquitous Computing*, pages 312–321, 2008.

[67] Y. Zheng, X. Xie, and W. Ma. GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory. *IEEE Data(base) Engineering Bulletin*, 33:32–39, 2010.

[68] Y. Zheng, L. Zhang, X. Xie, and W. Ma. Mining Interesting Locations and Travel Sequences from GPS Trajectories. In *Proceedings of the International Conference on World Wide Web*, pages 791–800, 2009.