

Contents

1	Introduction	1
2	Trajectory	4
2.1	Model for trajectory clustering	4
3	Trajectory Similarity	6
3.1	Warping based distance	6
3.1.1	DTW	8
3.1.2	LCSS	9
3.1.3	Pros and Cons	11
3.2	Shape based distance	11
3.2.1	Fréchet distance	11
3.2.2	Hausdorff distance	12
3.2.3	Pros and Cons	13
3.3	HC-SIM	14
4	Clustering	17
4.1	Methods	17
4.1.1	DBSCAN	19
4.1.2	K-Medoid	22
4.1.3	Affinity Propagation	23
4.1.4	Hierarchical Clustering	25
4.2	Quality Criteria	28
4.2.1	Silhouette Coefficient	29
4.2.2	Fowlkes-Mallows scores	30
5	Implementation	31
5.1	Ground Truth Data	31
5.1.1	Data	31
5.1.2	Trajectory Similarity	33
5.1.3	Analysis of the distances	34
5.1.4	Analysis of the number of cluster selection	40
5.2	Unknown Data	41
5.2.1	Data	41
5.2.2	Analysis of the distances	43

List of Tables

1	Data structure	33
---	--------------------------	----

List of Figures

1	Trajectories with varying lengths	2
2	Clustering Methods	3
3	A GPS trajectory with 7076 points from GeoLife Project dataset of Microsoft Research Asia (Qian & Lu, 2017).	6
4	Euclidean distance	7
5	Matching Methods (Su et al., 2020)	8
6	DTW distance	9
7	LCSS distance	10
8	Fréchet distance	12
9	Hausdorff distance	13
10	Three trajectories distances calculated by Hausdorff and Fréchet distance (Besse et al., 2015)	14
11	C-SIM	15
12	DBSCAN (Su et al., 2020)	19
13	DBSCAN Trajectory Clustering (Su et al., 2020)	21
14	The distance threshold r defines the neighborhood of a point. (Ertöz et al., 2003)	22
15	Hierarchical clustering models (Bian et al., 2019)	26
16	HAC clustering (Sultana, 2020)	28
17	Ward Linkage (Ignacio Gonzalez et al., 2017)	29
18	Mopsi Trajectories	32
19	Mopsi Clusters	32
20	Mopsi 11 Clusters	33
21	Overview of our process	34
22	LCSS Trajectories Clustering	35
23	LCSS 11 Clusters	35
24	DTW Trajectories Clustering	36
25	DTW 11 Clusters	36

26	Hausdorff Trajectories Clustering	37
27	Hausdorff 11 Clusters	37
28	Fréchet Trajectories Clustering	38
29	Fréchet 11 Clusters	38
30	HCSIM Trajectories Clustering	39
31	HCSIM 11 Clusters	39
32	Silhouette Score depending on cluster	40
33	Fowlkes–Mallows Score based on distance	41
34	Caltrain Trajectories Dataset	42
35	Calinski-Harabasz Index	43
36	Caltrain Clusters	45
37	Caltrain DTW Clusters	46
38	Caltrain LCSS Clusters	47
39	Caltrain Hausdorff Clusters	48
40	Caltrain Fréchet Clusters	49
41	Caltrain HCSIM Clusters	50

List of Equations

1	Euclidean distance	7
2	DTW distance	9
3	LCSS distance	10
4	Fréchet distance	12
5	Hausdorff distance	13

Abstract

Clustering is an efficient way to group data into different classes on basis of the internal and previously unknown schemes inherent of the data. With the development of the location based positioning devices, more and more moving objects are traced and their trajectories are recorded. Therefore, moving object trajectory clustering undoubtedly becomes the focus of the study in moving object data mining. To provide an overview, we survey and summarize the development and trend of moving object clustering and analyze typical moving object clustering algorithms presented in recent years. In this thesis, we firstly summarize the characteristics of trajectory. Secondly, the measures which can determine the similarity/dissimilarity between two trajectories are discussed. Thirdly, the strategies and implement processes of classical moving object clustering algorithms are analyzed. Finally, the validation criteria are analyzed for evaluating the performance and efficiency of clustering algorithms.

1 Introduction

The growing use of GPS receivers and WIFI embedded mobile devices equipped with hardware for storing data enables us to collect a very large amount of data, that has to be analyzed in order to extract any relevant information. The complexity of the extracted data makes it a difficult challenge. Knowing principal routes that people or vehicles follow during the day can provide valuable information for analysis of mobility. For instance the presence of important routes not adequately covered by the public transport service could be highlighted by a set of trajectories, which provide information on how to improve it. Trajectory clustering is an appropriate way of analyzing trajectory data, and has been applied to road network extraction, detecting taxi fraud, etc. Additionally, trajectory clustering is used to gather temporal spatial information in the trajectory data and is widespread used in many application areas, such as motion prediction (Z. Chen et al., 2010) and traffic monitoring (Atev et al., 2006)

Trajectory data is recorded in different formats depending on the type of device, object motion, or even purpose. For example, a trajectory is generated by GPS devices include sequence of points in geographical space, with a combination of coordinates and time stamp. In certain specific circumstances, other object-related properties such as direction, velocity or geographical information are added (Ying et al., 2011, 2010). This kind of multidimensional data is prevalent in many fields and applications, for example, to understand migration patterns by studying trajectories of animals, predict meteorology with hurricane data, improve athlete's performance, etc.

Given different types of analysis tasks and moving object data applications, measuring how similar two trajectories are is a common technique for most tasks and applications. Therefore, distances are a fundamental component of those tasks and applications of trajectory analysis, allowing us to determine effectively how close two trajectories are. Unlike other simple data types such as ordinal variables or geometric points where the distance description is straightforward, the distance between the trajectories must be carefully defined. It is because trajectories are basically high-dimensional data attached to both spatial and temporal attributes which need to be considered for distance measurements. As such the literature contains dozens of distance measurements for trajectory data. For example, dis-

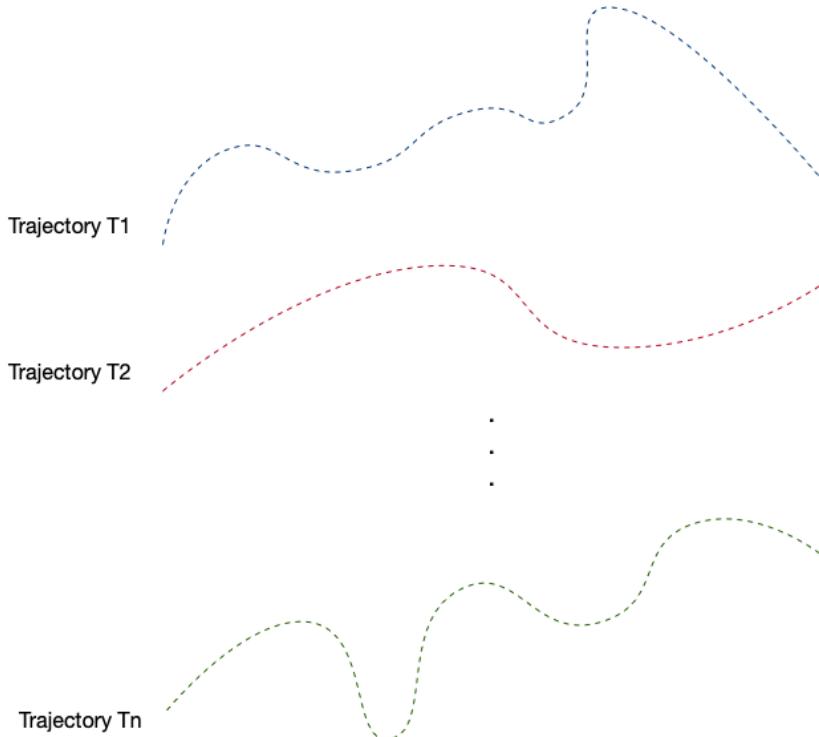


Figure 1: Trajectories with varying lengths

tance measurements measure the sequence-only distance between trajectories, such as Euclidean distance and Dynamic Time Wrapping Distance (DTW); they measure both spatial and temporal dimensions of two trajectories as well. In order to extract useful patterns from high-volume trajectory data, different methods, such as clustering and classification, are usually used. Clustering is an unsupervised learning method that combines data in groups (clusters) based on distance (Han et al., 2011; Xu & Wunsch, 2005). The aim of trajectory clustering is to categorize trajectories in cluster groups based on their similarities. The trajectories existing in each cluster have similar characteristics of movement or behavior within the same cluster and are different from the trajectories in other clusters (Berkhin, 2006; Besse et al., 2015; Yuan et al., 2017).

In general, two main approaches are used for clustering complex data such as trajectories. First, use trajectory-specific clustering algorithms and second, use generic clustering algorithms that support trajectory-specific distance functions (DFs). One of the most suitable clustering algorithms for trajectories is density-based clustering (Kriegel et al., 2011) which can extract clusters with arbitrary shape and is also tolerant against outliers (Ester et al., 1996). Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is one of the most popular

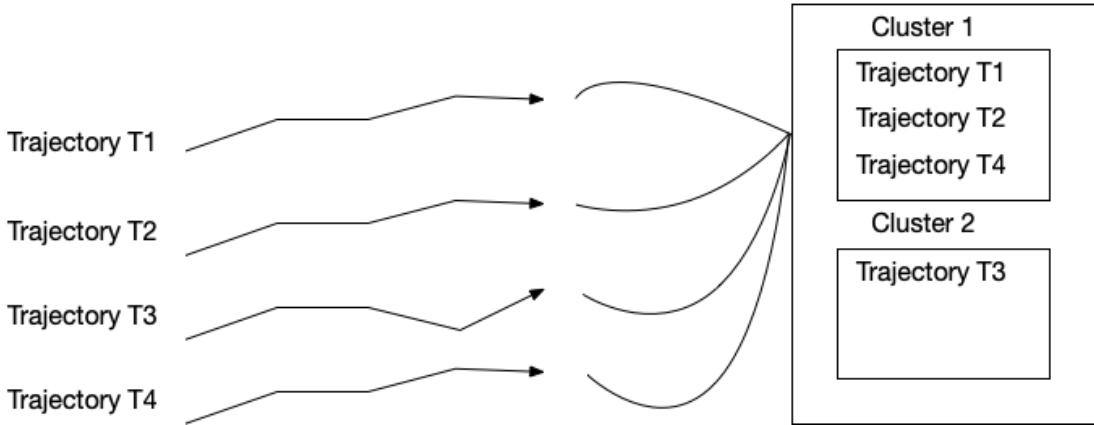


Figure 2: Clustering Methods

methods among this family, which is widely employed in trajectory clustering (Zhao & Shi, 2019; Cheng et al., 2018; J. Chen et al., 2011; Lee et al., 2007). Measurement of similarity is the central focus of the clustering problem; thus, similarity (inverse distance) should be calculated prior to grouping. Distance definition in spatial trajectories is much more complicated than point data. Trajectories are sequences of points in several dimensions that are not of the same length. Thus, in order to compare two trajectories, a comprehensive approach is needed to fully determine their distance. Depending on the analysis purpose and also the data type, different distance functions are presented. The concept of similarity is domain specific, so different distance functions are defined in order to address different aspects of similarity such as spatial, spatio-temporal, and temporal. Spatial similarity is based on spatial parameters like movement path and its shape whereas spatio-temporal similarity uses the temporal component as well. For instance, to extract the movement patterns in trajectories like detecting the transportation mode, besides the trajectory's geometry, temporal component should also be considered. Among all defined Distance Functions so far, Euclidean, Fréchet, Hausdorff, DTW, LCSS, EDR, and ERP distances are the basic functions in similarity measurements from which so many other functions are generated (Abbaspour et al., 2017; Aghabozorgi et al., 2015; Wang et al., 2013).

2 Trajectory

A trajectory is a sequence of time-stamped point records describing the motion history of any kind of moving objects, such as people, vehicles, animals, and natural phenomenon. For example, tracking devices with Global Position System (GPS) create a trajectory by tracking object movement as $Trajectory = (A, B, \dots, T_n)$, which is consecutive spatial space sequence of points, and T_i indicates a combination of coordinates and timestamps such as $T_i = (x_i, y_i, t_i)$.

Theoretically, a trajectory is considered to be a continuous record, i.e., a continuous function of time mathematically, since the object movement is continuous in nature. In practice, however, the continuous location record for a moving object is usually not available since the positioning technology (e.g., GPS devices, road-side sensors) can only collect the current position of the moving object in a periodic manner. Due to the intrinsic limitations of data acquisition and storage devices such inherently continuous phenomena are acquired and stored (thus, represented) in a discrete way. This subsection starts with approximations of object trajectories. Intuitively, the more data about the whereabouts of a moving object is available, the more accurate its true trajectory can be determined.

2.1 Model for trajectory clustering

The increasing use of GPS and Wi-Fi mobile devices equipped with data storage hardware allows for the collection of enormous data, for example, of the relevant data to be used in order to find the best way from A to B, detect abnormal behavior, optimize the flow of traffic in a city, foresee the next location or destination of a city. In fact, this aims to build from the data the different characteristics which characterize the different daily movement of road vehicles. We take clustering methods for trajectories into consideration. Clustering techniques are intended to consolidate similar trajectories into groups which differ. The complexity of the trajectory makes this a difficult task because objects in a given area can move along many different paths.

Several methods can be used to cluster trajectories. We will concentrate on trajectory clustering based on distance but have examined other specific methodologies.

Trajectories are usually treated as multidimensional (2D or 3D in most cases) time series data as in Figure 3; Gaffney & Smyth (1999), Vasquez & Fraichard (2004), Hu et al. (2006) successfully apply trajectory clustering methods on video-stream trajectories. The continuous definition of trajectory also applies by Gariel et al. (2011) to re-sample trajectories to achieve equal-length time series. These new trajectories are then analyzed for the main components to obtain and finally cluster the primary components. This approach is used for routes in aircraft. All these methods take both the spatial as well as the temporal aspect of the trajectory into account, not adapting to vehicle trajectories limited to a road network of very irregular times. Our main objective is to detect the main track and traffic flow that can be used to study the various behaviours.

In this context, the aim of this work is to build a collection of paths, which model car drivers' behaviors in a data-driven manner. These models are learned from a collection of automotive locations. We focus here on grouping trajectories with similar paths. This clustering is based on comparing trajectory objects and requires a new definition of the distance between the studied objects.

A large amount of work has been done to give new definitions of trajectory distance. Several methods have been used to cluster data set of trajectories. Clustering methods using Euclidean distance lead to inaccurate results mainly because trajectories have different number of points. Hence, several methods based on warping distance have been defined. Typical examples include the distance measures based on DTW, Edit distance and Longest Common Subsequence (LCSS), which reorganize the time index of trajectories to obtain a perfect match between them. Another approach is to focus on the geometry of the trajectories, in particular their shape. Shape distances like Hausdorff and Fréchet distances can be adapted to trajectories. In next section, we will study and compare several distances on trajectory.



Figure 3: A GPS trajectory with 7076 points from GeoLife Project dataset of Microsoft Research Asia (Qian & Lu, 2017).

3 Trajectory Similarity

There are many ways to define how close two objects are. A trajectory distance generalization that consideres moving objects and it measures on average how close two objects were during some time period. $d(A, B)$ denotes the distance between two trajectories A and B . The larger the value is, the less similar the two trajectories are. Many functions can be used to qualify this dissimilarity beyond the concept of mathematical distance. Distances can be classified into two categories: those whose compare the shape of trajectory (Shape-based distance) and those take into account the temporal dimension (Warping based distance).

3.1 Warping based distance

Euclidean distance is the most commonly adopted distance measures that pair-wisely computes the distance between corresponding points of two trajectories. Besides being relatively straightforward and intuitive, Euclidean distance and its variants have several other advantages. The complexity of computing value is linear; in addition, they are easy to implement, indexable withany access method, and parameter free. Euclidean distance was proposed as a distance measure between time series and once considered as one of the most widely used distance functions since the 1960s (Keogh & Pazzani, 2000; Faloutsos et al., 1994; Pfeifer & Deutrch, 1980; Priestley, 1980).

For two trajectories A and B , the Euclidean distance $d_{Euclidean}(A, B)$ with the

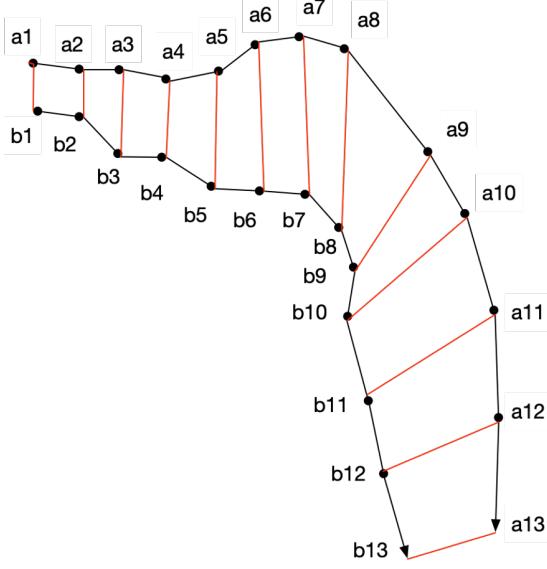


Figure 4: Euclidean distance

same size n is define as follows:

$$d_{Euclidean}(A, B) = \frac{\sum_{i=1}^n d(a_i, b_i)}{n} \quad (1)$$

Where a_i and b_i are the i th sample point of A and B respectively. The time complexity is $O(n)$.

Euclidean distance measure is straightforward; however, it requires the comparing trajectories to be the same size, which is not common in the actual situation; otherwise it will fail to decide the match pairs of the trajectories to be compared. The aim of warping distance is to solve this problem. To improve the accuracy of similarity measurement, the dynamic time warping algorithm (DTW), longest common subsequence algorithm (LCSS), EDR and ERP were introduced. These distances are defined the same way, but they use different cost functions. Firstly, these distance measures find all the sample point match pairs among the two compared trajectories A and B . A sample point match pair, $pair(a_i, b_j)$, is formed by two sample points where $a_i \in A$ and $b_j \in B$. There are several sample point matching strategies such as minimal Euclidean distance or minimal transformation cost. Then these distance measures accumulate the distance for matched pairs or count the number of match pairs to get the final distance results. Thus,

the sample point matching strategy is the key for every discrete sequence-only distance measure. The sample point matching strategies can be divided into the following two types:

Complete match: For two compared trajectories A and B , complete match strategy requires that every sample point of A and B should be in a match pair, as shown in Figure 5(a). Thus, the number of matches is $\max(\text{size}(A), \text{size}(B))$.

Partial match: For two compared trajectories A and B , partial match strategy does not require every sample point of A and B should be in a match pair, as shown in Figure 5(b). Thus, some sample points will not be matched to any sample points.

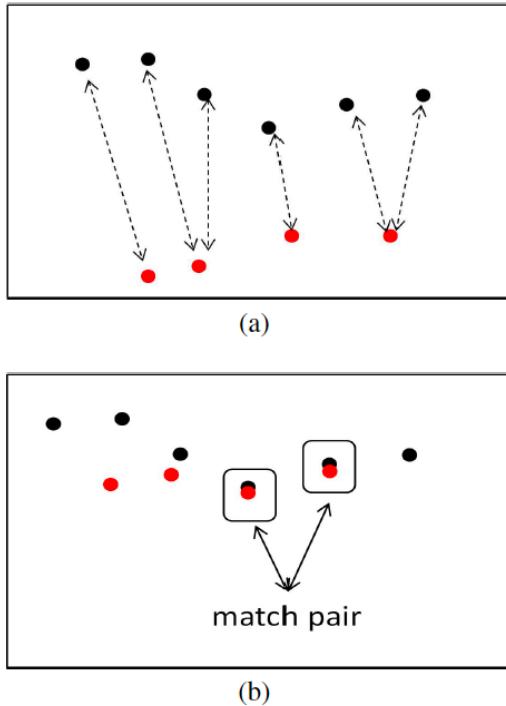


Figure 5: Matching Methods (Su et al., 2020)

3.1.1 DTW

Dynamic time warping (DTW) is an algorithm for measuring the distance between two sequences. DTW has existed for over a hundred years. Initially, DTW was introduced to compute the distance of time series (Myers et al., 1980). In 1980s, DTW was introduced to measure trajectory distance (Kruskal, 1983) and

has become one of the most popular trajectory distance measure since then. DTW distance is defined in a recursive manner and can be easily applied in dynamic programming. It searches through all possible points' alignment between two trajectories for the one with minimal distance.

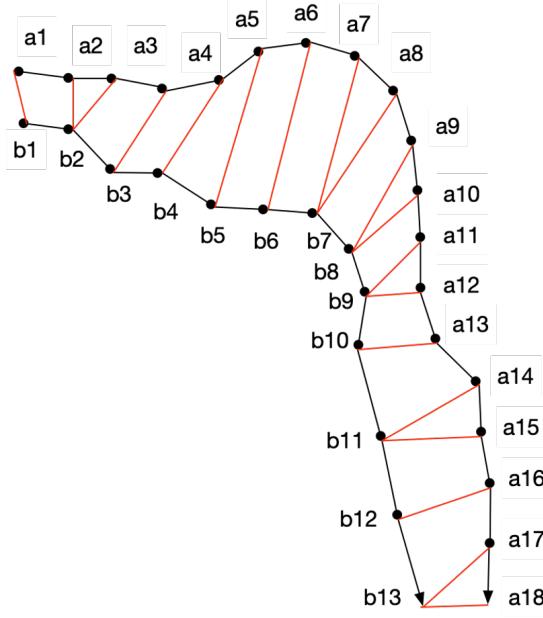


Figure 6: DTW distance

Specifically, the DTW $d_{DTW}(A, B)$ between two trajectories A and B with lengths of n and m is defined as:

$$d_{DTW}(A, B) = \begin{cases} 0, & \text{if } n = 0 \text{ and } m = 0 \\ \infty, & \text{if } n = 0 \text{ or } m = 0 \\ d(\text{Head}(A), \text{Head}(B)) + \min\{d_{DTW}(A, \text{Rest}(B)), \\ d_{DTW}(\text{Rest}(A), B), d_{DTW}(\text{Rest}(A), \text{Rest}(B))\}, & \text{otherwise} \end{cases} \quad (2)$$

The time complexity of DTW is $O(mn)$.

3.1.2 LCSS

Longest common subsequence (LCSS) and edit distance-based distance measures are the mainly used distance metrics of partial match measures. LCSS is a traditional similarity metric for strings, which is to find the longest subsequence common to two compared strings. The value of LCSS similarity between sequences

S_1 and S_2 stands for the size of the longest common subsequence of S_1 and S_2 . Trajectory can be treated as a sequence of sample points, so LCSS was used to measure the similarity between trajectories. The value of LCSS similarity between trajectories A and B stands for the size of the longest common subsequence of A and B . However, it can hardly find any two sample points with the exact same location information. Thus, in measuring the similarity of trajectories A and B , LCSS treats a_i ($a_i \in A$) and b_j ($b_j \in B$) to be the same as long as the distance between a_i and b_j , which is less than a threshold ϵ . Thus, some sample points of A and B cannot be in any match pairs. Then LCSS distance simply counts the number of match pairs between A and B .

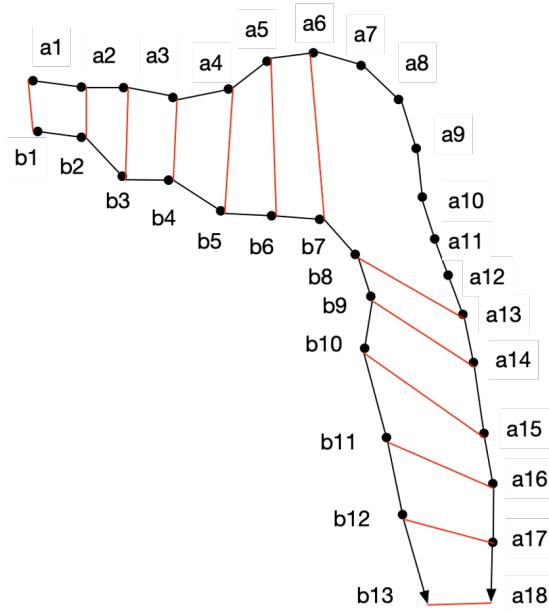


Figure 7: LCSS distance

Since the sample points far apart do not contribute to the value of LCSS distance, these sample points do not have match points. In contrast to Euclidean distance, LCSS is robust to noise. LCSS distance between trajectories is defined as:

$$d_{LCSS}(A, B) = \text{size}(A) + \text{size}(B) - 2s_{LCSS}(A, B) \quad (3)$$

3.1.3 Pros and Cons

The primary advantage of these distances is that they allow for the comparison of trajectories of varying lengths.

The following are the two main limitations of warping-based distance:

- Warping methods rely on a one-to-one comparison of sequences. As a result, it is frequently necessary to select a specific series that will be used as a reference, onto which all other sequences will be mapped. The index of two sequences being compared should be well balanced in order to best capture variability, such as detecting if there were accelerations and decelerations during time series measurement. As a result, the reference sequence must be carefully chosen.
- The performance of traditional warping-based methods is hampered by the large amount of noise inherent in road traffic data, which is not the case when examining time series.

3.2 Shape based distance

These distances try to catch geometric features of the trajectories, in particular, their shape instead of matching the trajectory sample points directly. This means trajectories are clustered independently of their location and of a global change of scale. The Hausdorff distance and Fréchet distance are the most well-known distance.

3.2.1 Fréchet distance

The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. The Fréchet distance between the two curves is the length of the shortest leash sufficient for both to traverse their separate paths. Let A and B be two trajectories which can be represented as two continuous functions f_a and f_b over time t , where the starting time and end time of time period t are denoted by $t.start$ and $t.end$

respectively. The Fréchet distance between A and B is defined as the infimum over all reparameterizations $f_a(t)$ and $f_b(t)$.

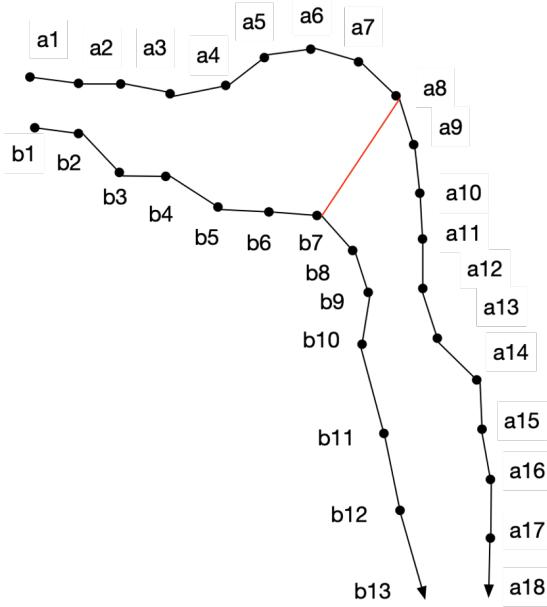


Figure 8: Fréchet distance

The Fréchet distance algorithm is shown below:

$$d_{Frechet}(A, B) = \inf \max_{t \in [t.start, t.end]} \{d(f_a(t), f_b(t))\} \quad (4)$$

Since its value is the longest distance between two trajectories at the same time, a noisy point is always far away from a trajectory, causing Fréchet distance to be very sensitive to noise.

The time complexity of Fréchet distance is $O(mn)$.

3.2.2 Hausdorff distance

The Hausdorff distance is a metric. It measures the distance between two sets of metric spaces. Informally, two sets are close in the Hausdorff distance if every point of either set is close to some point of the other set. The Hausdorff distance is the longest distance you can be forced to travel by an adversary who chooses a point in one of two sets, from where you then must travel to other set. In other words, it is the greatest of all the distances from a point in one set to the closest point in the other set.

The Hausdorff distance algorithm is shown below:

$$d_{Hausdorff}(A, B) = \max\{\sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b)\} \quad (5)$$

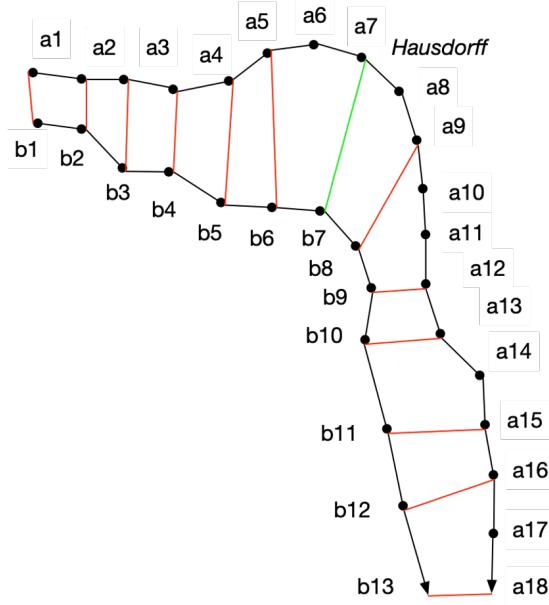


Figure 9: Hausdorff distance

The time complexity of Hausdorff distance is $O(n^2)$.

3.2.3 Pros and Cons

Both the Fréchet and Hausdorff distances are metrics, which means that they satisfy triangle inequality. This is a necessary attribute of the distance utilized in clustering algorithms like dbscan or K-medoid if we want the clustering process to be efficient. They've been frequently employed in a variety of fields that require shape comparison. However, they may not be able to compare entire trajectories. For example, the Fréchet and Hausdorff distances both return the maximum distance between two trajectories at specific points.

As in Figure 10, we can see that T^1 and T^2 are the most comparable of the three trajectories, yet they are the furthest apart in Fréchet due to the maximum distance of six at the end of the trajectory. Meanwhile, the Hausdorff distances for the three trajectories are nearly identical, implying poor precision. These

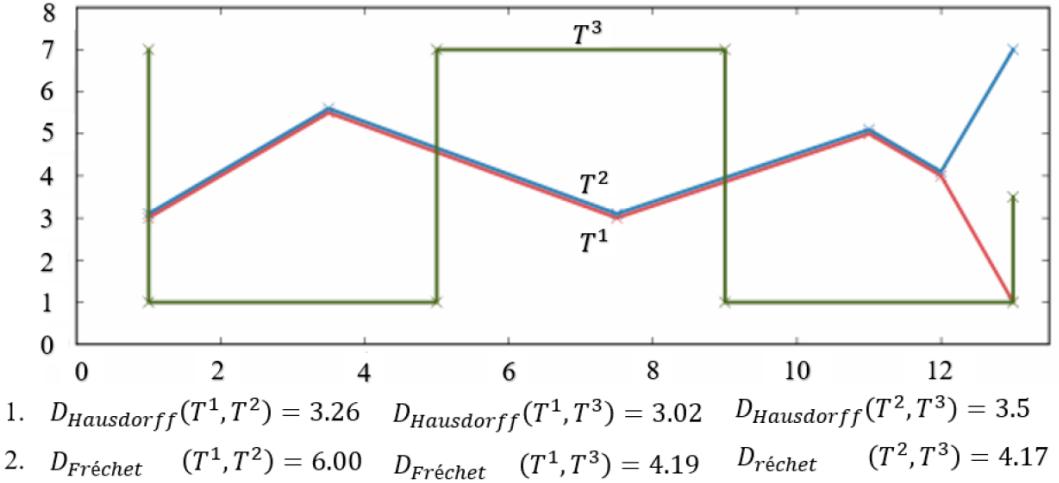


Figure 10: Three trajectories distances calculated by Hausdorff and Fréchet distance (Besse et al., 2015)

two approaches are effective in applications involving shape comparisons, such as image comparison.

3.3 HC-SIM

Fränti & Mariescu-Istodor (2019) proposed a new shape-based distance, HC-SIM (Hierarchical Cell Similarity), a parameter independent variant of C-SIM measure, and we will compare it with other shape-based distances. HC-SIM are proposed as it is the least affected by changes in sampling rate and performs fairly well under noise and point shifting.

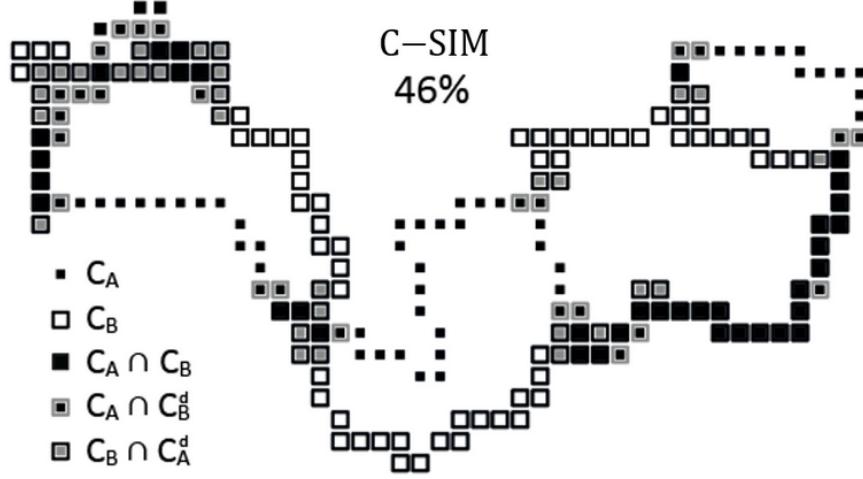


Figure 11: C-SIM

C-SIM algorithm was proposed by Mariescu-Istodor & Fränti (2017) to compute similarity between two trajectories. It first retrieves the cell representation and then calculates the similarity measure using the cells. Intersection algorithm computes the intersection between two sets efficiently in linear time using the hashing technique as below:

C-SIM: Computing Similarity Between Two Trajectories

Input: Trajectories A and B

Ouput: Similarity

- $C_1, C_1^d \leftarrow Points - to - Cells(A)$
- $C_2, C_2^d \leftarrow Points - to - Cells(B)$
- $cab \leftarrow Intersection(A, B)$
- $cab^d \leftarrow Intersection(A, B^d)$
- $cba^d \leftarrow Intersection(B, A^d)$
- $similarity \leftarrow (cab + cab^d + cba^d) / (|C_1| + |C_2| - cab)$

Points-to-Cells: Finding the set of Cells that approximate a given trajectory. Mariescu-Istodor & Fränti (2017) used a hashing method to keep track of cells

already existing in the representation. With 25×25 -meter-sized cells, a 100×100 km square results in a grid of size $4,000 \times 4,000$ cells. A trajectory consists of pairs of Easting and Northing values that passes through the cells. Gaps can appear in the cell representation when user is traveling faster than the cell length divided by sampling interval, or when user moves and the device fails to update the location or for some other reasons. The cells were generated in the order that the trajectory points were recorded; if two consecutively generated cells are not adjacent, the gap is filled by using linear interpolation with equation:

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1 \quad (6)$$

where (x_1, y_1) and (x_2, y_2) are the easting and northing values of the two cells inside a 100km square.

Intersection: Jacard index was used to measure amount of similarity. It is calculated as the size of the intersection divided by the size of the union of two sets (Mariescu-Istodor & Fränti, 2017):

$$J(C_A, C_B) = \frac{|C_A \cap C_B|}{|C_A \cup C_B|} \quad (7)$$

Each of two trajectories are dilated separately, compute the two intersections with original of the other trajectory as follows:

$$S(C_A, C_B) = \frac{|C_A \cap C_B| + |C_A \cap C_B^d| + |C_B \cap C_A^d|}{|C_A| + |C_B| - |C_A \cup C_B|} \quad (8)$$

The total time complexity of C-SIM is $O(N_A + N_B + |C_A| + |C_B|)$ where N_A and N_B are the number of points in the two trajectories.

In this thesis, we calculate HC-SIM distance through dissimilarity measure:

$$\text{HC-SIM}(A, B) = \frac{1}{L} \sum_{i=1}^L \text{C-SIM}(A, B) \quad (9)$$

$$d_{\text{HC-SIM}}(A, B) = 1 - \text{HC-SIM}(A, B) \quad (10)$$

4 Clustering

Clustering is the most common unsupervised learning method in pattern recognition. It is basically the task of grouping objects such that similar objects reside in the same group together. Clustering algorithms have gained a lot of attention among researchers and therefore, a large number of clustering algorithms have evolved. In order to be successful with clustering, first we need to understand the nature of the objects we need to cluster. We need to examine their properties and understand how these objects are inputted to an algorithm. For instance, one may be interested to know whether the data that needs to be clustered is inputted to the algorithm incrementally. This is an interesting problem when data is collected as clusters are evolving. On the other hand, the dataset can be present as a whole prior to the execution of the clustering algorithm. Additionally, the properties of the objects that need to be clustered are very important. These properties can give rise to a number of important questions such as if the objects are vector-based or if the objects are metric based. Moreover, one needs to know if objects can be transformed from one form to another so that further analysis can be done on them. One of the most important aspects of clustering is a way of comparing the objects which is widely known as a distance function.

In this section, we will explore clustering algorithms and make an attempt to show which clustering algorithm is reasonable for trajectory clustering. We first start off with some prerequisites that many clustering algorithms require and show that these requirements are met by the trajectory distance function. Next, we move onto a traditional hierarchical clustering (HAC) algorithm and we will show how HAC attacks the problem. HAC is simple to understand and it may present us with some insight as to what we should expect from a general clustering algorithm.

4.1 Methods

Clustering algorithms are generally designed to be able to target a large class of objects. However, some algorithms have specific assumptions on the type of objects that are to be clustered. The most general assumption is a mathematical definition for the objects so they can be stored in memory or on a storage device

using a data structure. Perhaps the second most widely accepted requirement is a distance function so the algorithm would be able to say how similar two objects are relative to another. Clustering algorithms use a vector-based representation of the objects so each object can be represented as a point in a k-dimensional space and typically the Euclidean distance is used as the distance function between the points. Observe that such assumptions can be very strong in real world problems. We know that we have a mathematical representation of our trajectories. We have also discussed the distance function between the trajectories and we also know that the trajectory distance function follows the metric space rules: it is symmetrical, and it follows the triangulation property. HAC (Hierarchical Agglomerative Clustering) method only requires a distance function between the objects.

The choice of the clustering method is restricted by the characteristics of the trajectory object. Indeed, trajectories have different lengths which prevents an easy definition of a mean trajectory object. The k-means method cannot be used on our trajectory set, nor spectral clustering methods. k-medoid can be used but an efficient algorithm, like partitioning around medoids, or dbscan method, require a valid metrics. Indeed, these algorithms are based on nearest neighbor and require the distance used to satisfy the triangular inequality. Most of the studied distances, LCSS, DTW, are not metrics. In this way, dbscan or partitioning around medoids algorithms will not be used. Moreover, dbscan depends on two extra parameters that are hard to estimate in this case. To perform the clustering of the trajectories, we will focus on two methodologies: hierarchical cluster analysis (HCA) and affinity propagation (AP). As a matter of fact, HCA and AP can use distance/similarity which does not satisfy the triangle inequality. We point out that the choice of the clustering method is restricted to the trajectory object we deal with. Actually, trajectories have different lengths. HCA and AP are both methods which only require the distance/similarity matrix, and thus can cluster objects of different lengths. Both these methods will be used to evaluate our distance.

4.1.1 DBSCAN

The Densely Clustering models measure how closely points are packed together after given the centroids. Inspired by this idea, Density-based spatial clustering of applications with noise (DBSCAN), which has been widely applied to trajectory clustering is proposed in 1996. The key idea is that for any data vector to belong to a cluster, there must be at least a given number of data vectors within a specified radius. In other words, the density of the neighborhood around the data vector must exceed a given threshold. In general, the shape of the neighborhood depends on the choice of distance function.

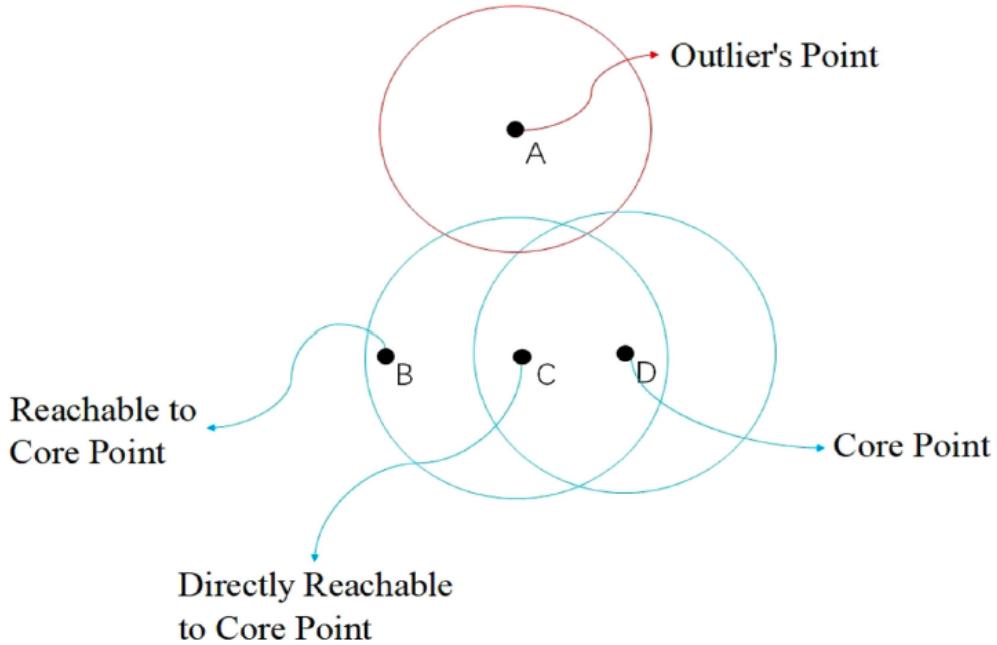


Figure 12: DBSCAN (Su et al., 2020)

The DBSCAN algorithm (Ester et al., 1996; Kriegel et al., 2011) takes two global inputs: the radius r and the minimum number of data vectors k . Given these two values, the local density of a data vector is defined as the number of data vectors k_i that lie within the radius r from the data vector x_i . If the local density is greater than the minimum number of points k , then x_i is called a core point. The concepts of density-reachability and density-connectivity are the basis for the notion of cluster. Ester et al. (1996) define these as follows: A data vector p is directly density-reachable from another data vector q if the distance between them is shorter than r , and q is a core point. A data vector p is density-reachable from a

vector q if they are joined by a sequence of data vectors $x_1, \dots, x_n, x_1 = q, x_n = p$ such that the data vector x_{i+1} is directly density-reachable from the data vector x_i . A data vector p is density-connected to the vector q if there is an intermediate data vector o from which both p and q are density-reachable. Data vectors that are not core points but are part of a cluster but are called border points. Finally, data vectors that do not belong to a cluster are called noise points.

The original DBSCAN algorithm gained widespread popularity because it was very efficient for large data sets. According to Kotsiantis & Pintelas (2004), DBSCAN out-performs hierarchical and partitioning algorithms. However, this seems to depend on the dimensionality of the data: Kriegel et al. (2011) gives the time complexity of DBSCAN as $O(N^2)$ in the worst case, although they also claim that the average runtime time complexity can be brought down to $O(N \log N)$ if the dimensionality of the data is not too high and R*-trees are used for indexing the data. For comparison, the worst case time complexity of $k - means$ is $O(INK)$, where I is the number of iterations, N is the number of data vectors, and K is the number of clusters. Hierarchical agglomerative clustering can be performed in $O(\tau N^2)$ where τ is a small number that refers to the number of nearest neighbors that need to be updated after merging clusters (Franti et al., 2006).

DBSCAN does not require the number of clusters to be given as a parameter because of the way the clusters are formed based on the connectivity of data vectors to each other. The algorithm can generally be used for any data set where a distance function between two data vectors can be defined. Like with non-parametric methods in general, an important advantage is that it can discover clusters of varying sizes and shapes. DBSCAN can also handle outliers because of its concept of noise.

High-dimensional data can also make clustering more challenging distances between data vectors become more uniform, effectively making them seem more similar (Ertöz et al., 2003). Furthermore, choosing the parameters for the distance and density thresholds requires domain expertise, and the definitions of the algorithm such as density-connectivity and border points are not very simple. A simplified version of the DBSCAN algorithm was proposed by Campello et al. (2013) which gets rid of the concept of border points.

These techniques do not require the number of clusters to be given as an input parameter and also do not make any assumptions about the density within the clusters. The basic idea of using DBSCAN for trajectory clustering is that considering a similarity function (such as Euclidean or DTW), each trajectory existing in a cluster should have at least $\text{min}T_r$ number of trajectories in its neighbourhood within a radius of Eps .

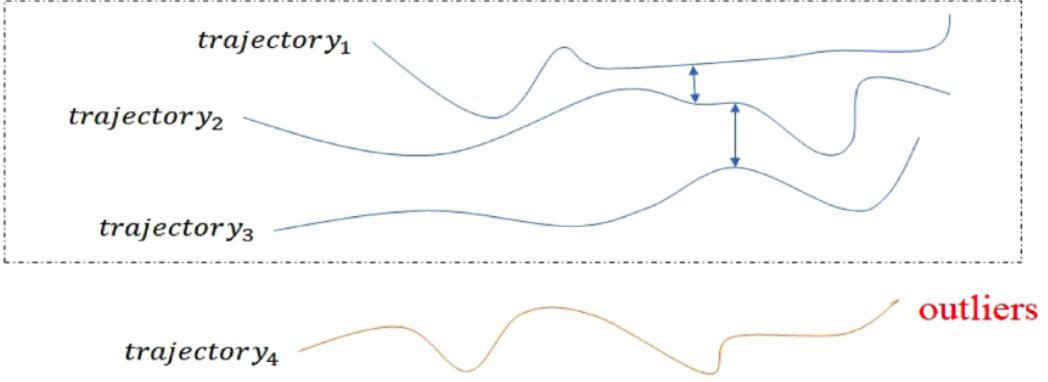


Figure 13: DBSCAN Trajectory Clustering (Su et al., 2020)

This is basic idea of using DBSCAN in trajectory clustering task according to Moayedi et al. (2019) with trajectory dataset TD :

- Eps-neighborhood of trajectory $T_i \in TD$ is defined by $N_{Eps(T_i)} = \{T_j \in TD | dis(T_i, T_j) \leq Eps\}$. If Eps-neighborhood of T_o has at least $\text{min}Tr$ trajectories, T_o is called a core trajectory.
- T_i is directly density-reachable to T_o with respect to Eps, $\text{min}Tr$, and DF if $T_i \in N_{Eps}(T_o)$ and $|N_{Eps}(T_o)| \geq \text{min}Tr$
- For a give Eps, $\text{min}Tr$ and DF, trajectory T_p is density-reachable from a trajectory T_q if there exists a series of trajectories $A, \dots, T_n, A = T_q, T_n = T_p$ in the condition that T_k is directly density-reachable to T_{k+1} .
- Trajectories T_i and T_j are density-connected for a given Eps, $\text{min}Tr$ and DF, if there is a trajectory T_0 such that both T_i and T_j are density-reachable from it.
- A cluster C is a non-empty subset of TD , for every T_p, T_q : if $T_p \in C, T_q$ is density-reachable from T_p then $T_q \in C$ and T_p is density-connected to T_q . A trajectory which is not part of any clusters is defined as an outlier.

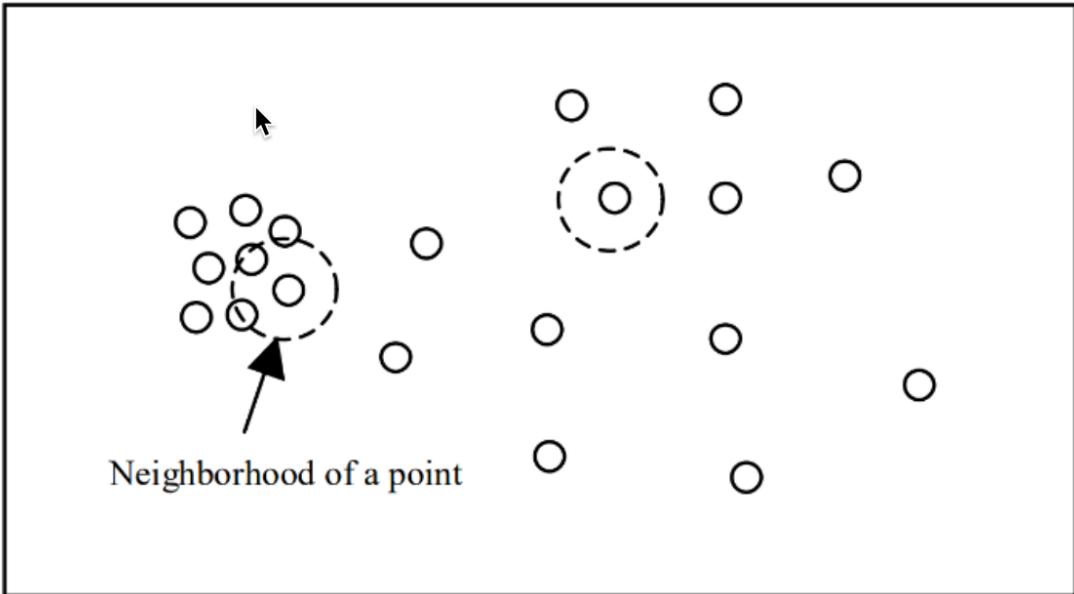


Figure 14: The distance threshold r defines the neighborhood of a point. (Ertöz et al., 2003)

A disadvantage of DBSCAN is that it struggles with data sets that contain clusters of varying densities (Ertöz et al., 2003). An example of this is shown in Figure 14. If the chosen distance threshold is too small, all the points belonging to the sparse cluster will be considered noise. If the threshold is too large, all points will be in the same cluster. A possible solution might be to run the algorithm multiple times and removing the already found clusters from the data set after each run.

4.1.2 K-Medoid

K-medoids algorithm groups the data based on their distance to each other. Medoid is the representative of a cluster that has the maximum sum of similarity to others in the cluster that has the minimum distance between this object and the medoid.

Medoid is the representative of a cluster that has the maximum sum of similarity to others in the cluster [9]. It should be one of the objects in the set, which is different from Median [43]. Median is the value separates the higher half from the lower half in the set, it could be the average of two middle numbers if set size is even number, otherwise it is the only middle one. As is introduced in [37], K-medoids is more robust compared with the K-means algorithm because it reduces

the influence from outlier and noise; but the time complexity is $O(k(n - k)^2)$, k is the number of clusters, n is the number of objects, so it is high complexity. In [39], the author gives us a method to speed up the K-medoids algorithm, so that the time complexity could be reduced to $O(n^2)$.

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

Partition Around Medoid: Get clusters by K-medoids clustering

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

Output: K clusters

Algorithm:

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

4.1.3 Affinity Propagation

Affinity propagation (AP) is a centroid based clustering algorithm similar to k Means or K medoids, which does not require the estimation of the number of clusters before running the algorithm. Affinity propagation finds “exemplars” i.e. members of the input set that are representative of clusters. Both similarities and preferences are often represented through a single matrix, where the values on the main diagonal represent preferences. Matrix representation is good for

dense datasets. Where connections between points are sparse, it is more practical not to store the whole $n \times n$ matrix in memory, but instead keep a list of sparse matrix. Behind the scene, ‘exchanging messages between points’ is the same thing as manipulating matrices, and it’s only a matter of perspective and implementation. The main drawbacks of K-Means and similar algorithms are having to select the number of clusters and choosing the initial set of points. Affinity Propagation, instead, takes as input measures of similarity between pairs of data points, and simultaneously considers all data points as potential exemplars. Real-valued messages are exchanged between data points until a high-quality set of exemplars and corresponding clusters gradually emerges.

As an input, the algorithm requires us to provide two sets of data: Similarities between data points, representing how well-suited a point is to be another one’s exemplar. If there’s no similarity between two points, as in they cannot belong to the same cluster, this similarity can be omitted or set to -Infinity depending on implementation. Preferences, representing each data point’s suitability to be an exemplar. We may have some a priori information which points could be favored for this role, and so we can represent it through preferences.

The algorithm then runs through a number of iterations, until it converges. Each iteration has two message-passing steps:

Calculating responsibilities: Responsibility $r(i, k)$ reflects the accumulated evidence for how well-suited point k is to serve as the exemplar for point i , taking into account other potential exemplars for point i . Responsibility is sent from data point i to candidate exemplar point k .

Calculating availabilities: Availability $a(i, k)$ reflects the accumulated evidence for how appropriate it would be for point i to choose point k as its exemplar, taking into account the support from other points that point k should be an exemplar. Availability is sent from candidate exemplar point k to point i .

In order to calculate responsibilities, the algorithm uses original similarities and availabilities calculated in the previous iteration (initially, all availabilities are set to zero). Responsibilities are set to the input similarity between point i and point k as its exemplar, minus the largest of the similarity and availability sum between point i and other candidate exemplars. The logic behind calculating how

suitable a point is for an exemplar is that it is favored more if the initial a priori preference was higher, but the responsibility gets lower when there is a similar point that considers itself a good candidate, so there is a ‘competition’ between the two until one is decided in some iteration. Calculating availabilities, then, uses calculated responsibilities as evidence whether each candidate would make a good exemplar. Availability $a(i, k)$ is set to the self-responsibility $r(k, k)$ plus the sum of the positive responsibilities that candidate exemplar k receives from other points. Finally, we can have different stopping criteria to terminate the procedure, such as when changes in values fall below some threshold, or the maximum number of iterations is reached. At any point through Affinity Propagation procedure, summing Responsibility (r) and Availability (a) matrices gives us the clustering information we need: for point i , the k with maximum $r(i, k) + a(i, k)$ represents point i ’s exemplar. Or, if we just need the set of exemplars, we can scan the main diagonal. If $r(i, i) + a(i, i) > 0$, point i is an exemplar. We’ve seen that with K-Means and similar algorithms, deciding the number of clusters can be tricky. With AP, we don’t have to explicitly specify it, but it may still need some tuning if we obtain either more or less clusters than we may find optimal. Luckily, just by adjusting the preferences we can lower or raise the number of clusters. Setting preferences to a higher value will lead to more clusters, as each point is ‘more certain’ of its suitability to be an exemplar and is therefore harder to ‘beat’ and include it under some other point’s ‘domination’. Conversely, setting lower preferences will result in having less clusters; as if they’re saying “no, no, please, you’re a better exemplar, I’ll join your cluster”. As a general rule, we may set all preferences to the median similarity for a medium to large number of clusters, or to the lowest similarity for a moderate number of clusters. However, a couple of runs with adjusting preferences may be needed to get the result that exactly suits our needs.

4.1.4 Hierarchical Clustering

Some clustering algorithms such as k-means clustering (MacQueen et al., 1967) require a constant value (k as input) for the number of clusters prior to execution. The main issue with such algorithms is the unknown nature of data. In a real-world problem, one normally does not expect to have a precise knowledge of how the data is spread out across the space in advance. Therefore, it is a very strong

assumption to expect the number of clusters in advance. In contrast, hierarchical clustering algorithms [13] try to tackle these issues by introducing a different view of how clusters can be constructed. Such algorithms usually require a measure of dissimilarity among clusters.

As the name suggests, hierarchical clustering algorithms produce a hierarchical representation of the data. Such hierarchical data representations are normally stored in a tree data structure. Each node of the tree is considered to be a cluster. The root of the tree is at the highest level and it contains all the elements. The root of the tree can be viewed as a single cluster. Each internal node of the tree contains children and each child can be interpreted as a single cluster. Assuming that the entire dataset is given in advance, there are two basic strategies:

- Agglomerative (bottom-up)
- Divisive (top-down)

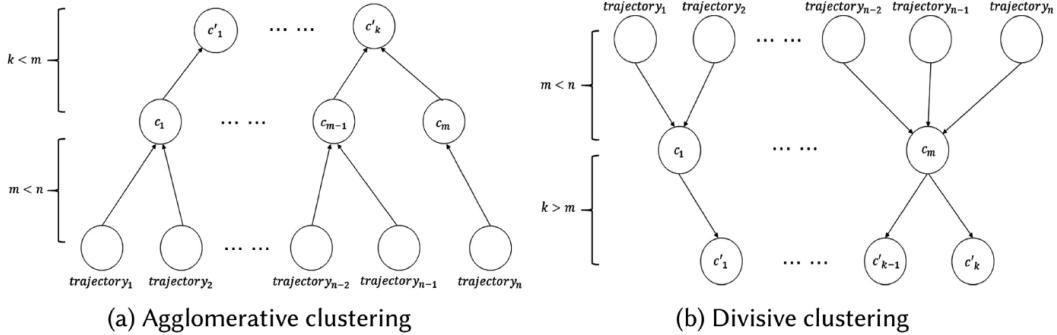


Figure 15: Hierarchical clustering models (Bian et al., 2019)

The agglomerative approach starts at the bottom of the tree. Each object is inserted to a leaf node representing a single cluster containing a single object. The algorithm proceeds by merging similar objects by constructing internal nodes containing several nodes from the previous level. This is done recursively until there is a single node which defines the root of the tree.

The divisive approach works in opposite direction. Unlike the agglomerative approach, the divisive approach starts off by creating the root of the tree. The algorithm continues by dividing the entire dataset into two subsets of similar objects. This approach is also recursive. The recursion continues until the division process reaches a subset with a single node.

In this section, we focus on HAC (Hierarchical Agglomerative Clustering) as an example of the hierarchical clustering algorithm. The main idea is to give some insight of how such algorithms are designed. This is also beneficial because the reader will have an idea of what these tree data structures may look like. Assuming that we are given a dataset with n objects to cluster. HAC starts by creating a cluster for each object and then it performs $n - 1$ merges. At each step, the two most similar clusters are merged together. These merges form new internal nodes of the tree containing more objects. Notice that when two nodes are being merged together, the algorithm needs to have a dissimilarity measurement between clusters. This means that the distance function is not enough. However, the cluster dissimilarity measure is derived from the provided distance function. HAC clustering may use one of the following ways to compute the merge cost:

- Single Linkage
- Complete Linkage
- Average Linkage
- Ward Linkage

The dissimilarity measurement is a derivation of distance function. Let G and H be two clusters, the single linkage (SL) measurement is the least distance between any two objects, $g \in G$ and $h \in H$:

$$d_{SL}(G, H) = \min_{\substack{g \in G \\ h \in H}} d(g, h) \quad (11)$$

The complete linkage (CL) is the opposite of single linkage: complete linkage is the maximum distance between any two objects, $g \in G$ and $h \in H$:

$$d_{CL}(G, H) = \max_{\substack{g \in G \\ h \in H}} d(g, h) \quad (12)$$

The average linkage (AL) is the average between groups:

$$d_{AL}(G, H) = \frac{1}{|G| \times |H|} \sum_{g \in G} \sum_{h \in H} d(g, h) \quad (13)$$

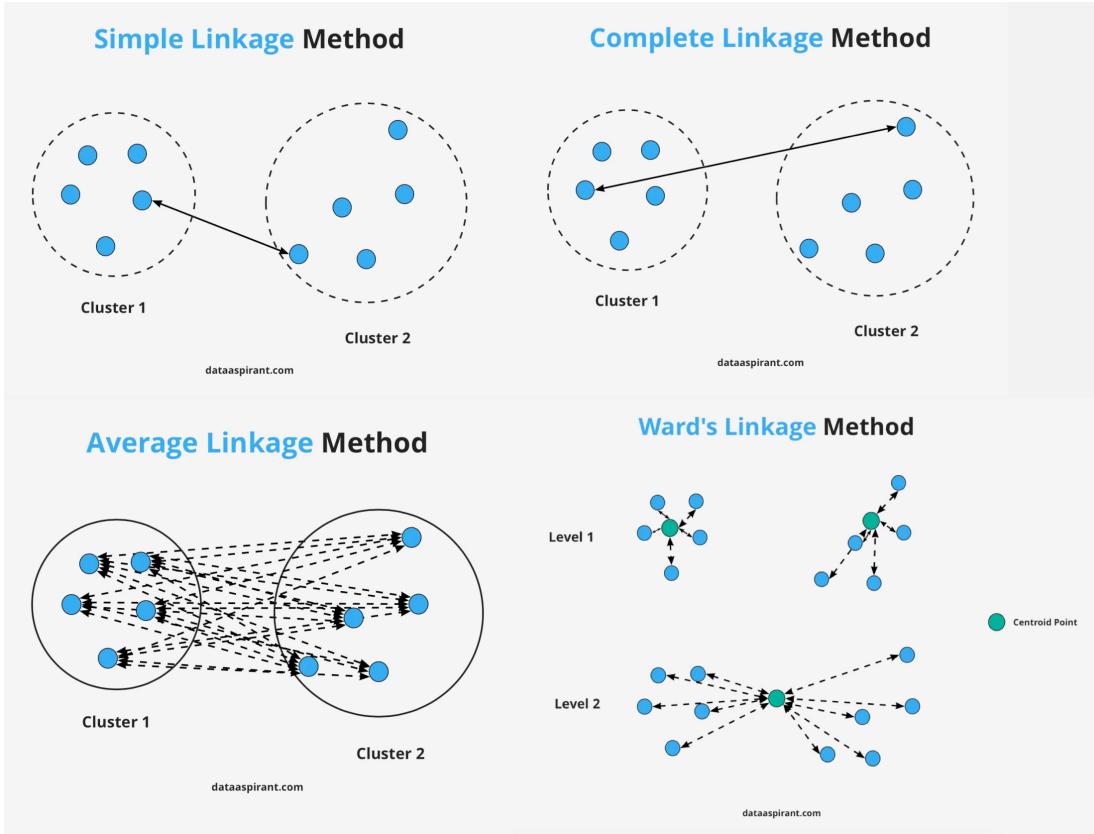


Figure 16: HAC clustering (Sultana, 2020)

Finally, in the ward linkage (WL) for each cluster a error function is defined. This error function is the average distance of each datapoint in a cluster to the center of gravity in the cluster:

$$d_{WL}(G, H) = \frac{n_G n_H}{n_G + n_H} \|\vec{m}_G - \vec{m}_H\|^2 \quad (14)$$

The one we choose to use is called Ward Linkage. Unlike the others, instead of measuring the distance directly, it analyzes the variance of clusters. Ward's is said to be the most suitable method for quantitative variables.

Where \vec{m}_j is the center of cluster j , and n_j is the number of points in it.

4.2 Quality Criteria

A clustering algorithm aims to gather objects into homogeneous groups that are far one from another. In this section, we focus on those scores to evaluate clusters

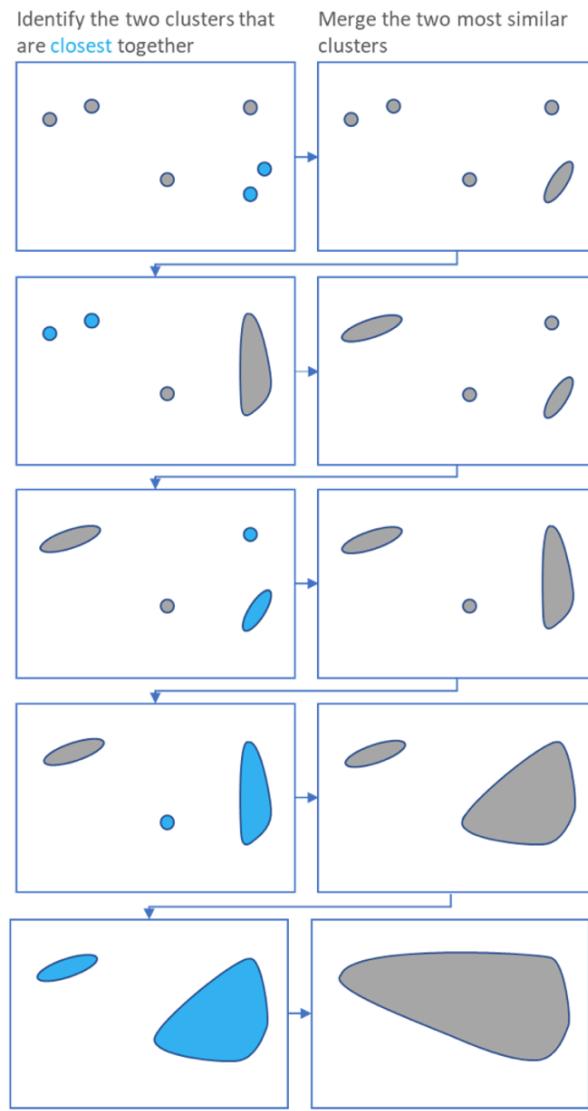


Figure 17: Ward Linkage (Ignacio Gonzalez et al., 2017)

when ground truth labels are known or unknown.

4.2.1 Silhouette Coefficient

If the ground truth labels are not known, evaluation must be performed using the model itself. The Silhouette Coefficient (Rousseeuw, 1987) is an example of such an evaluation, where a higher Silhouette Coefficient score relates to a model with better defined clusters. The Silhouette Coefficient is defined for each sample and is composed of two scores:

- **a:** The mean distance between a sample and all other points in the same class.
- **b:** The mean distance between a sample and all other points in the *next nearest cluster*.

The Silhouette Coefficient s for a single sample is then given as:

$$s = \frac{b - a}{\max(a, b)} \quad (15)$$

Silhouette coefficients (as these values are referred to as) near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster.

4.2.2 Fowlkes-Mallows scores

The Fowlkes-Mallows index (Fowlkes & Mallows, 1983) can be used when the ground truth class assignments of the samples is known. The Fowlkes-Mallows score FMI is defined as the geometric mean of the pairwise precision and recall:

$$FMI = \frac{TP}{\sqrt{(TP + FP)(TP + FN)}} \quad (16)$$

Where TP is the number of **True Positive** (i.e. the number of pair of points that belong to the same clusters in both the true labels and the predicted labels), FP is the number of **False Positive** (i.e. the number of pair of points that belong to the same clusters in the true labels and not in the predicted labels) and FN is the number of **False Negative** (i.e the number of pair of points that belongs in the same clusters in the predicted labels and not in the true labels).

Since the index is directly proportional to the number of true positives, a higher index means greater similarity between the two clusterings used to determine the index. One basic way to test the validity of this index is to compare two

clusterings that are unrelated to each other. Fowlkes & Mallows (1983) showed that on using two unrelated clusterings, the value of this index approaches zero as the number of total data points chosen for clustering increase; whereas the value for the Rand index (Steinley, 2004) for the same data quickly approaches making Fowlkes–Mallows index a much more accurate representation for unrelated data. This index also performs well if noise is added to an existing dataset and their similarity compared. Fowlkes & Mallows (1983) showed that the value of the index decreases as the component of the noise increases. The index also showed similarity even when the noisy dataset had a different number of clusters than the clusters of the original dataset. Thus making it a reliable tool for measuring similarity between two clusters.

The score ranges from 0 to 1. A high value indicates a good similarity between two clusters.

5 Implementation

In this section, we evaluate and compare 5 distances LCSS, DTW, Hausdorff, Fréchet and HCSIM. All these distances have been implemented in Python.

We also used Python for the implementation of the chosen clustering algorithms, the *scipy* library for *hierarchical clustering analysis*.

5.1 Ground Truth Data

5.1.1 Data

The data we used are GPS data collected from Mopsi. Mopsi is a website that helps users to find where their friends are and what is around them (Mariescu-Istodor, 2013). There are trajectories recorded by users. Those trajectories are displayed on the map with detailed information, such as speed, traveled distance and user’s transportation mode (walking, running, cycling, skiing) which is automatically inferred by the method in Waga et al. (2012). Mopsi allows the user to search trajectories in different ways. It also provides recommendations and tools

for managing data collection.

Mopsi data has two types: geo-tagged photos and trajectories. Geo-tagged photos contain information about their location and recorded time. The trajectory is a set of GPS point stored at a fixed interval. In this thesis, we use those trajectories as the data source. There were 444 trajectories in Joensuu, Finland extracted from Mopsi data as shown in Figure 18.



Figure 18: Mopsi Trajectories

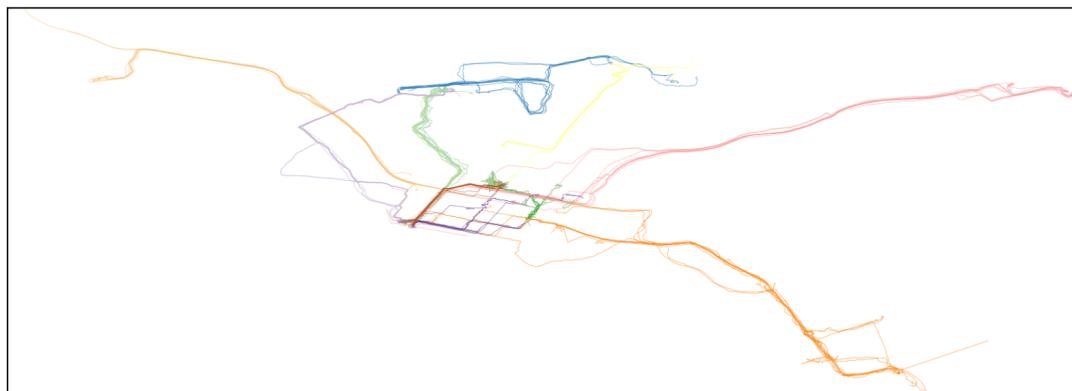


Figure 19: Mopsi Clusters

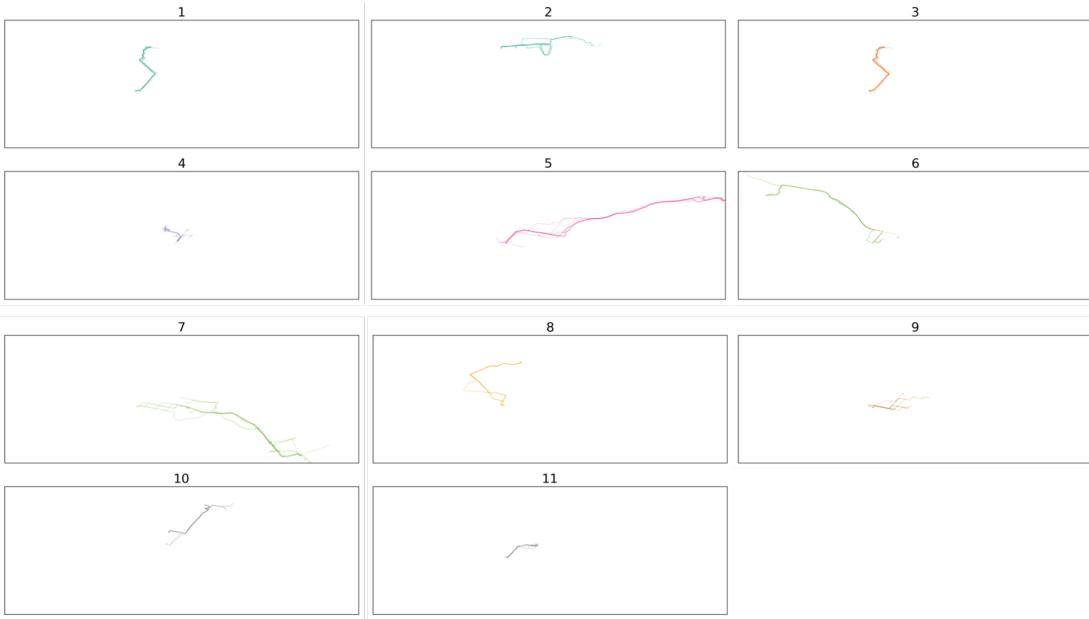


Figure 20: Mopsi 11 Clusters

The data structure is in Table 1.

Column	Type	Description	Example
Latitude	Double	Point latitude	62.926880 ($62^{\circ} 55' 36.7674''$)
Longitude	Double	Point longitude	23.184691 ($23^{\circ} 11' 4.8876''$)
Timestamp	String	Point timestamp	1559983789 seconds
Altitude	Double	Point altitude	-1.0 meter

Table 1: Data structure

5.1.2 Trajectory Similarity

To cluster “similar” trajectories together, it is essential to formulate a similarity measure between two trajectories. The process computed trajectory similarity was illustrated as follows:

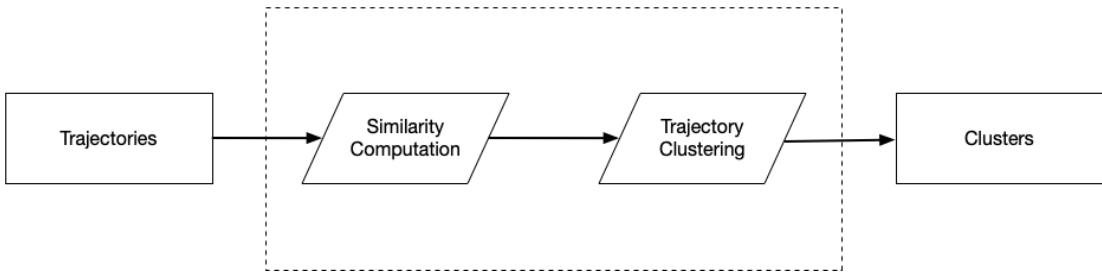


Figure 21: Overview of our process

Figure 21 shows our clustering process that consists of two components. We first identified clues among trajectories then uses them to measure to measure pair-wise similarity among trajectories. With a set of trajectories with pair-wise similarity measures, we labeled trajectories to capture similarity relationship among trajectory pairs. After that based on the label we are able to obtain clusters of trajectories that have high similarity.

5.1.3 Analysis of the distances

Hierarchical Clustering models help to understand trajectory by multiple features, so this treetype construction is proper to implement. As shown in Figure 15, two hierarchical types are also known as “bottom-up” and “top-down” approaches.

In Agglomerative frameworks, trajectories are grouped and the similar clusters are merged by searching their common properties. Optimal clusters are obtained by repeating representation computation and clusters merging until meeting the requirements. Different from Agglomerative, Divisive frameworks cluster trajectory data into groups and split them recursively to reach the requirements. And as mentioned before, we chose Ward Linkage method on HAC (Hierarchical Agglomerative Clustering).

Below figures are Mopsi clustering results using 5 distances LCSS, DTW, Hausdorff, Fréchet and HCSIM with Ward Linkage cluter method.

LCSS - Number of Clusters: 11



Figure 22: LCSS Trajectories Clustering

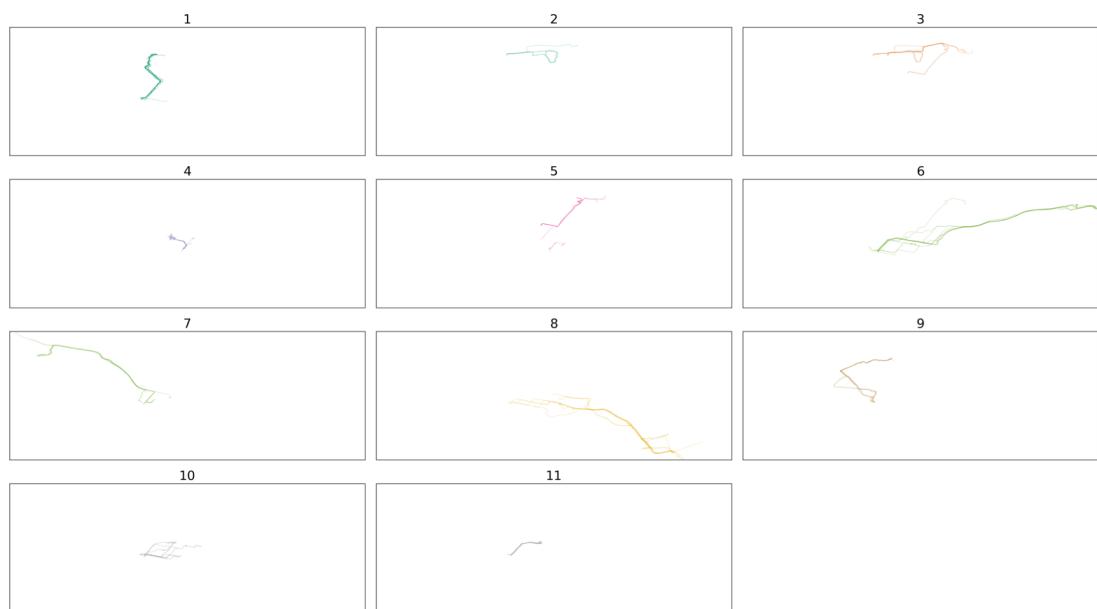


Figure 23: LCSS 11 Clusters

DTW - Number of Clusters: 11

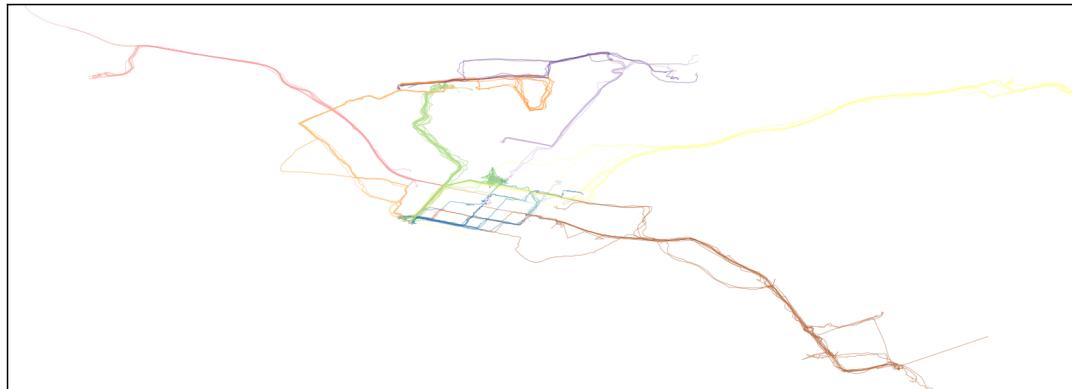


Figure 24: DTW Trajectories Clustering

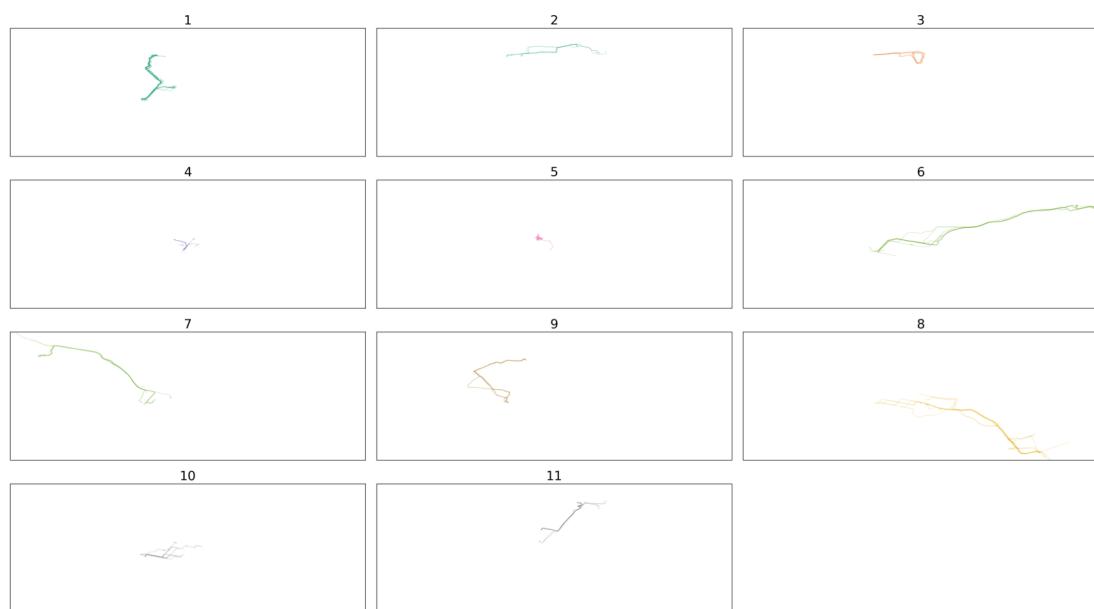


Figure 25: DTW 11 Clusters

Hausdorff - Number of Clusters: 11

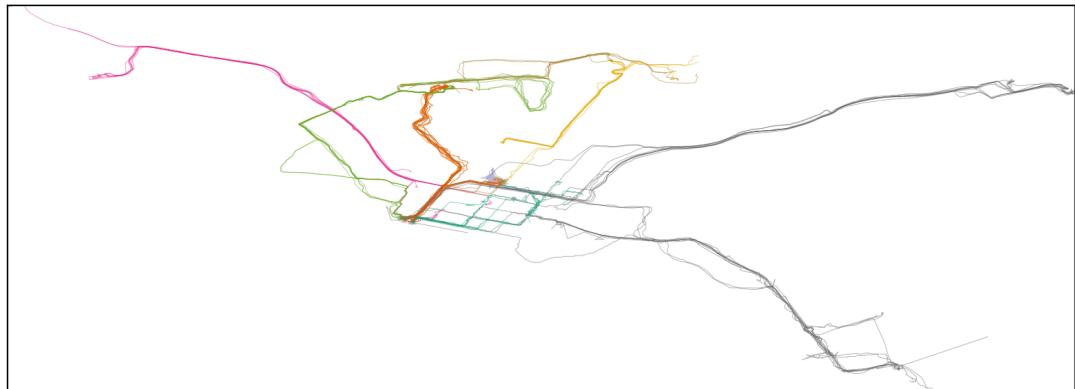


Figure 26: Hausdorff Trajectories Clustering

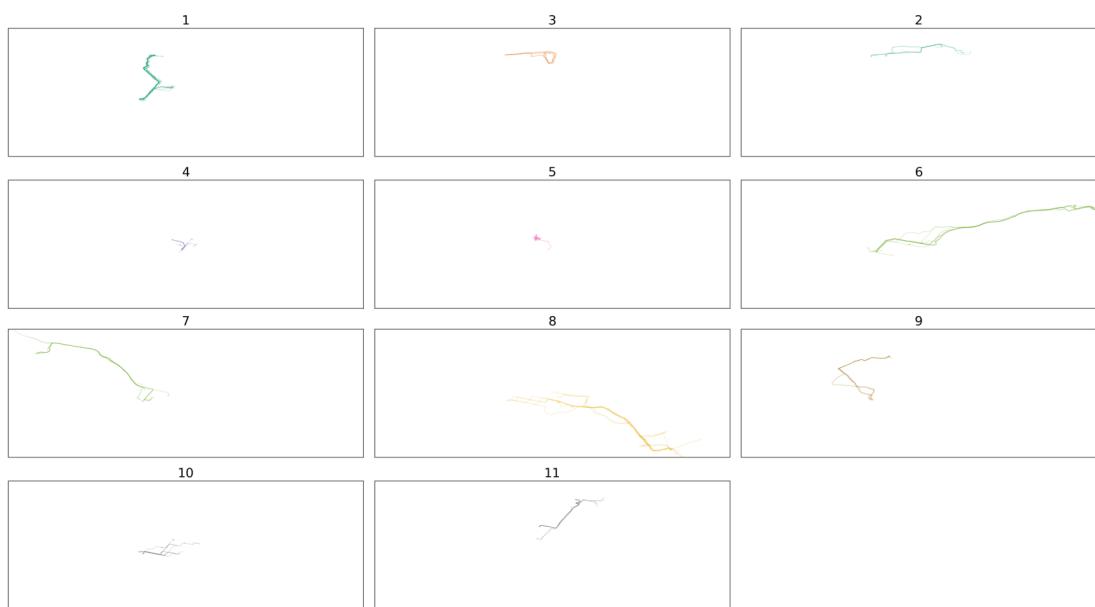


Figure 27: Hausdorff 11 Clusters

Fréchet - Number of Clusters: 11

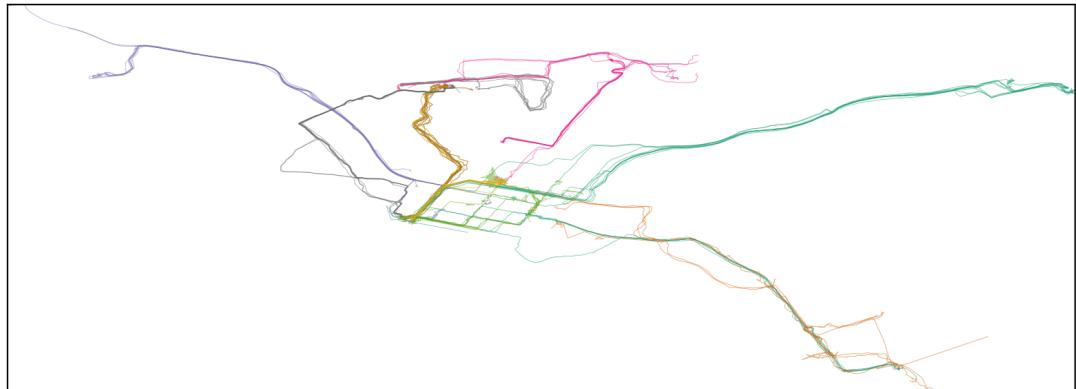


Figure 28: Fréchet Trajectories Clustering



Figure 29: Fréchet 11 Clusters

HCSIM - Number of Clustering: 11



Figure 30: HCSIM Trajectories Clustering

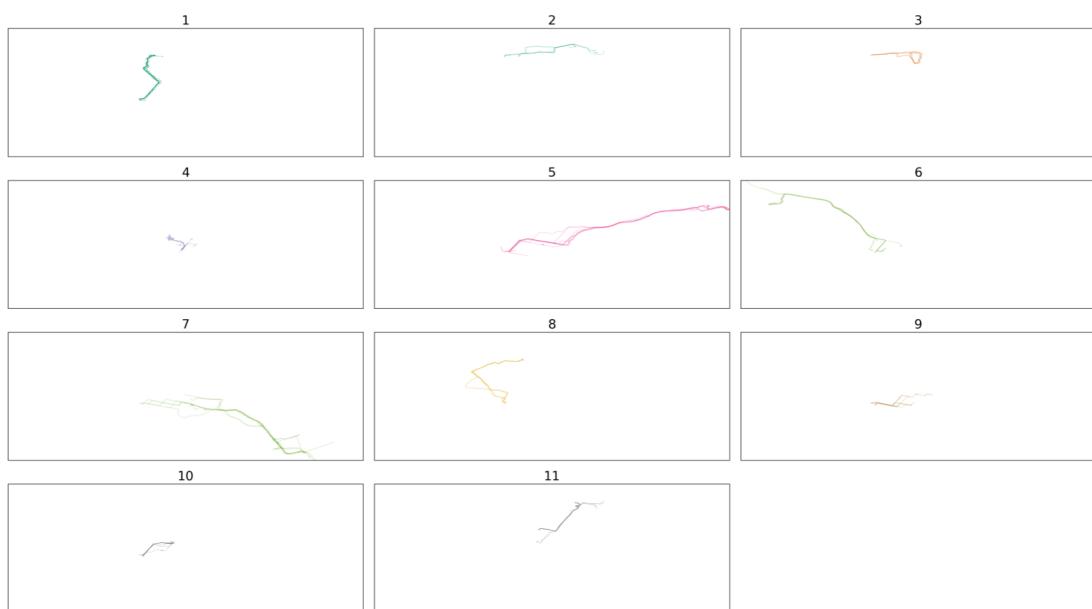


Figure 31: HCSIM 11 Clusters

The Warping-based distances, LCSS and DTW, give the poorest results with LCSS being significantly worse than DTW. The shape-based distance Hausdorff and SSPD give better result. These results confirm that shape-based distances are better adapted than warping-based distances.

5.1.4 Analysis of the number of cluster selection

We used *Silhouette Score* to decided the number of clusters.

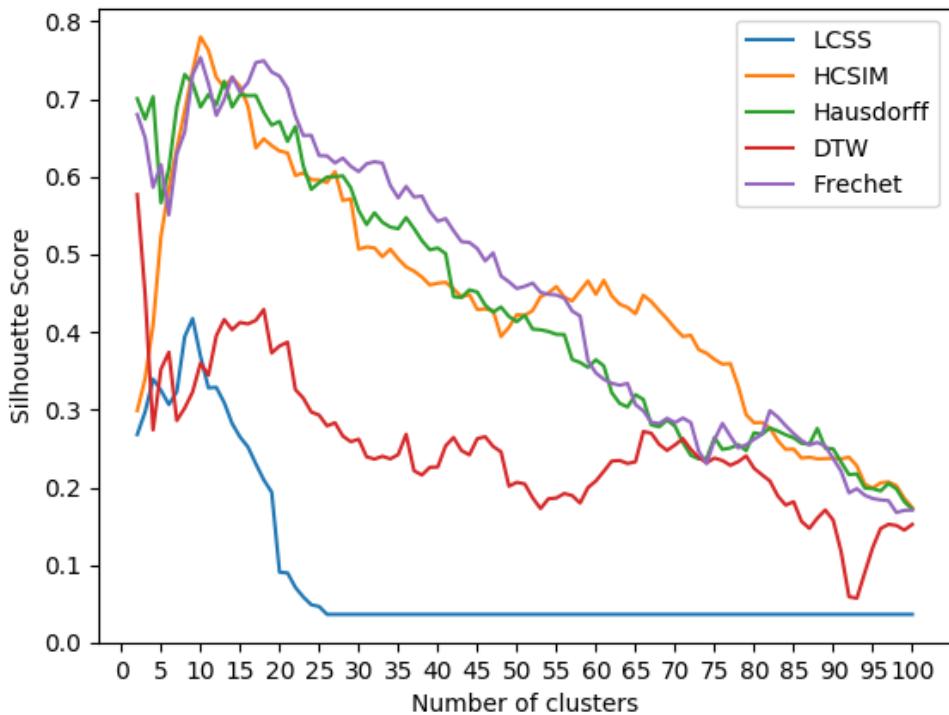


Figure 32: Silhouette Score depending on cluster

Figure 25 illustrated Silhouette Score based on number of clusters using 5 distances. 10 clusters are found with LCSS, 13 with DTW, 9 with Hausdorff and 11 with SSPD, HCSIM. We got the groundtruth data, and the right number of clusters is 11. So SSPD and HCSIM give the best results.

As 11 is the number of clusters, so we used Fowlkes Mallows index to evaluate cluster results from those 5 distances. Figure 31 showed that HCSIM (shaped base distance) give the better cluster results than warping-based distances with

Fowlkes–Mallows Score is over 0.85.

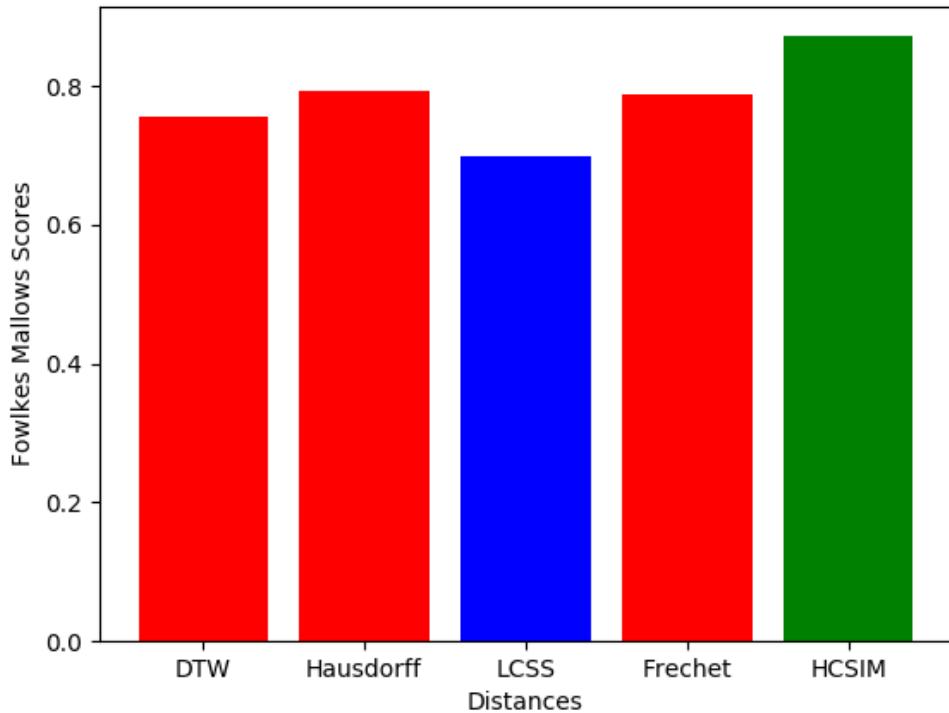


Figure 33: Fowlkes–Mallows Score based on distance

5.2 Unknown Data

5.2.1 Data

The data we used are GPS data from 536 San-Francisco taxis over a 24-day period. These data are public and can be found on Piorkowski et al. (2009). We extracted data as shown in Figure 27.

This data is a blend of 2574 trajectories. All have the same pickup location, the Caltrain station, and all have a drop-off location in downtown San-Francisco.

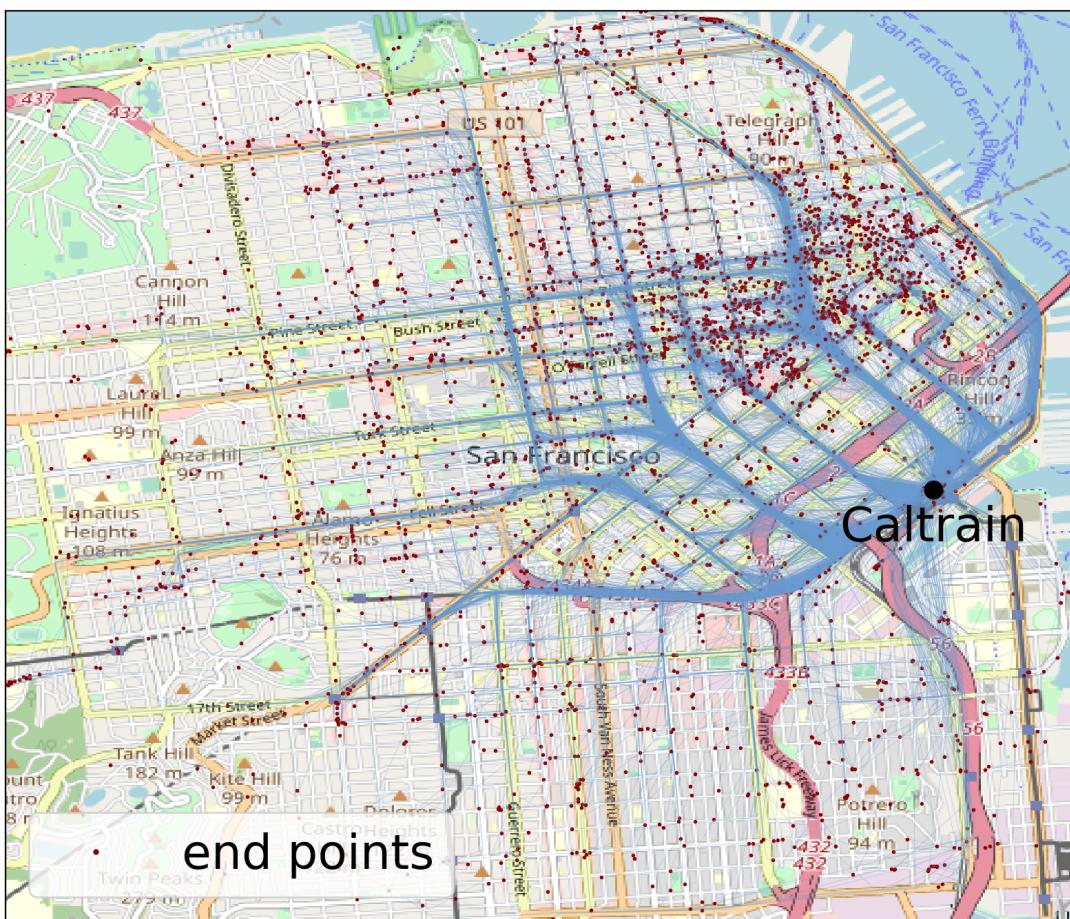


Figure 34: Caltrain Trajectories Dataset

5.2.2 Analysis of the distances

We used Calinski-Harabasz Index (Caliński & Harabasz, 1974) to evaluate cluster size for all distances.

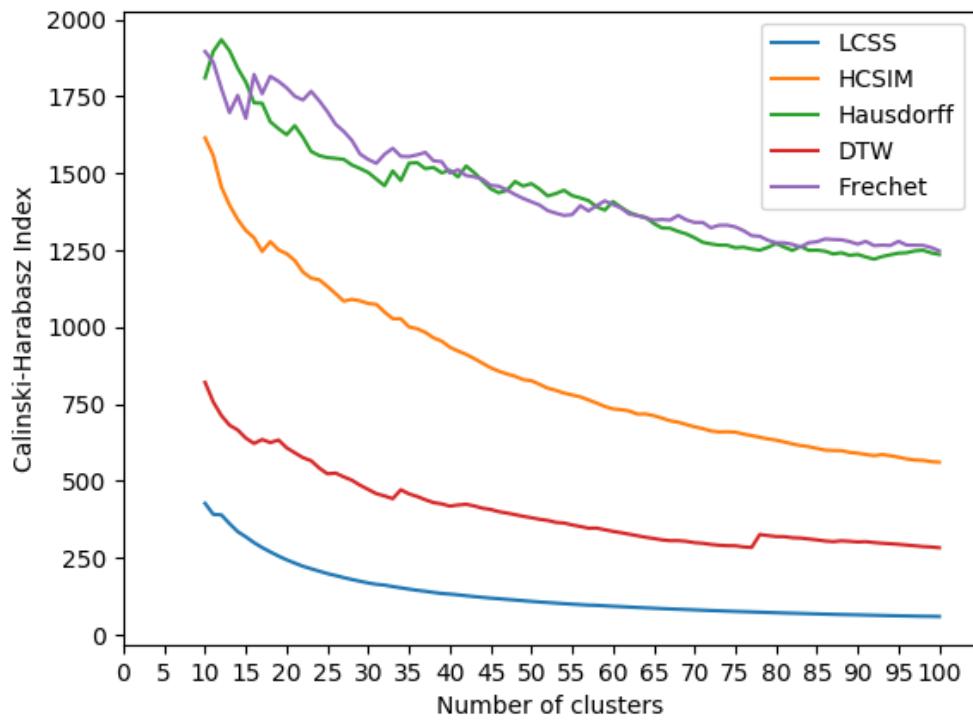


Figure 35: Calinski-Harabasz Index

The Warping-based distances, LCSS and DTW, give the poorest results with LCSS being significantly worse than DTW. And the two shape-based distances Frechet and Hausdorff give better results.

Below is the visual results for all distances with 15 clusters. Based on Mopsi experiment, HCSIM is expected to be the best. We observe that trajectories are well classified by HCSIM according to their paths in 30 and 31. The clusters using HCSIM seem to be consistent. We obtain a partition of traffic based on the taxi drivers' behavior leaving the Caltrain station in San Francisco.

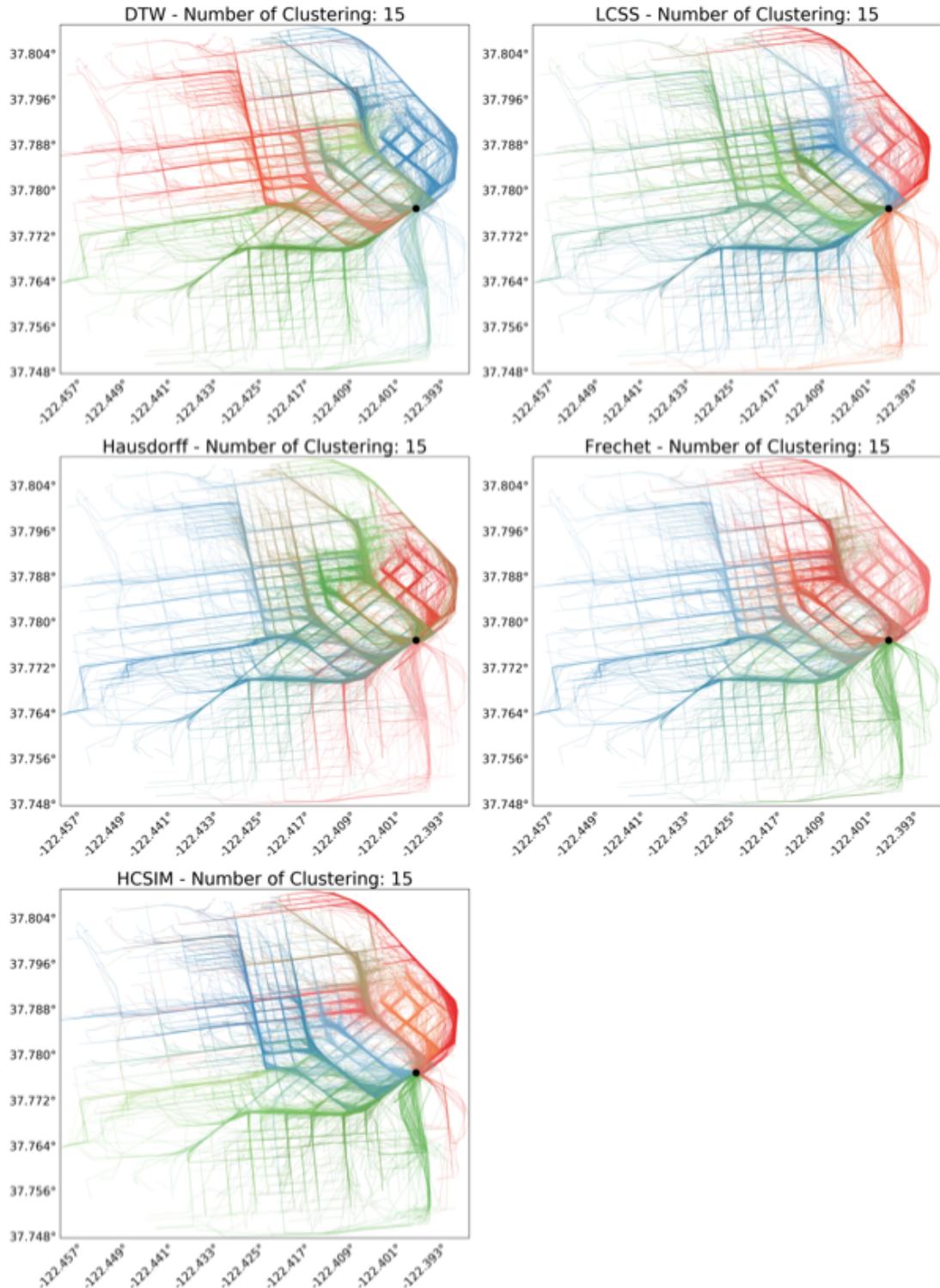


Figure 36: Caltrain Clusters

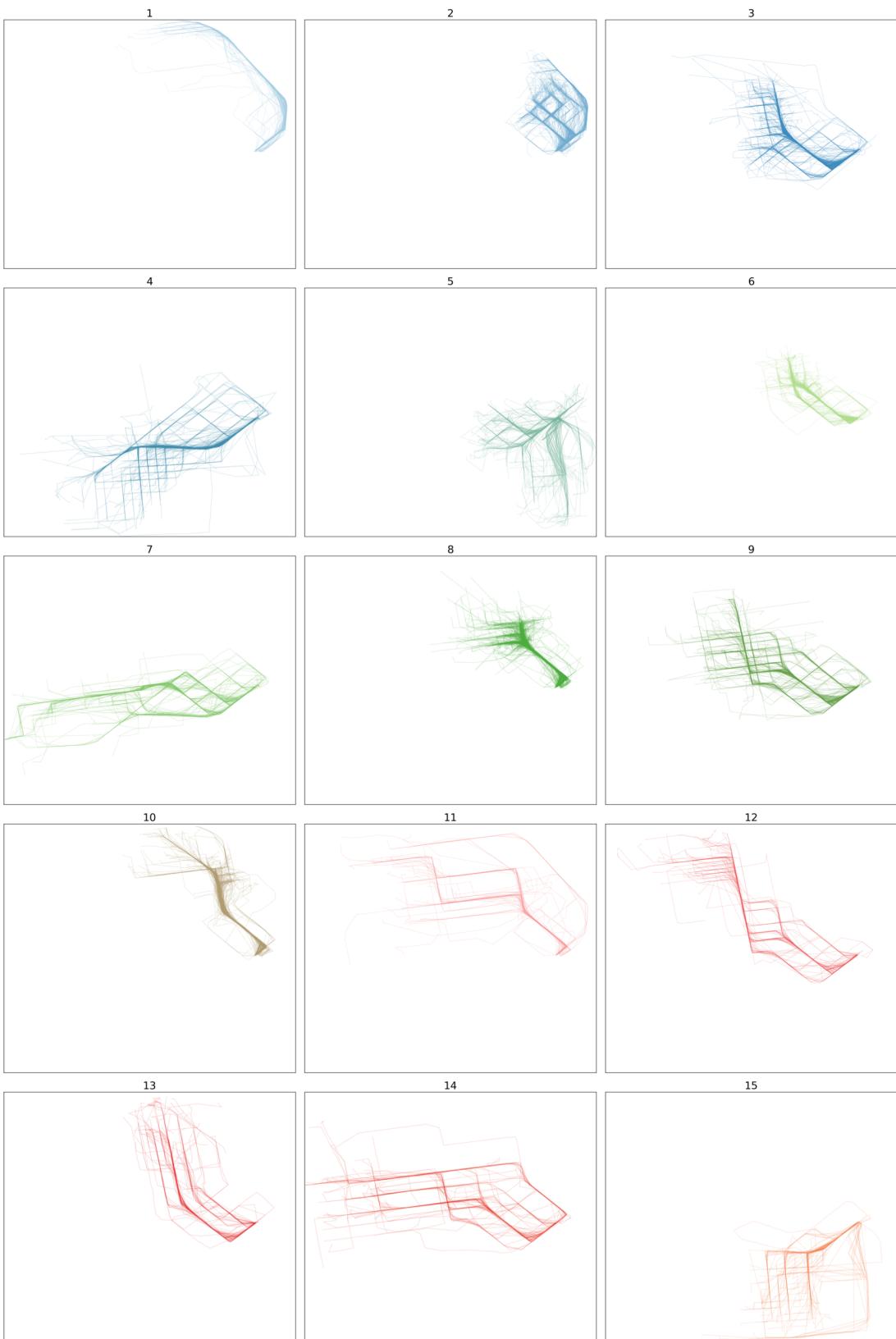


Figure 37: Caltrain DTW Clusters

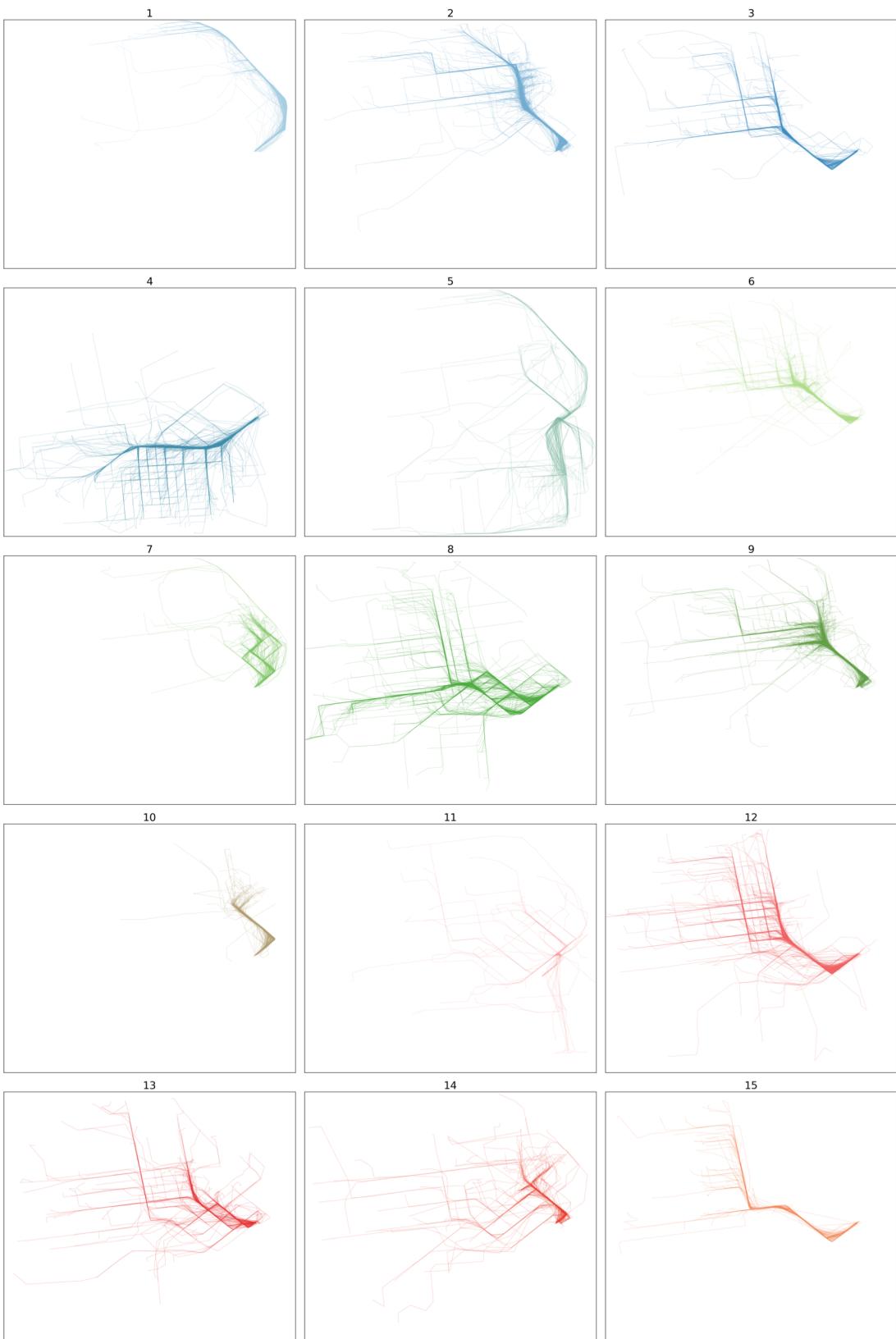


Figure 38: Caltrain LCSS Clusters

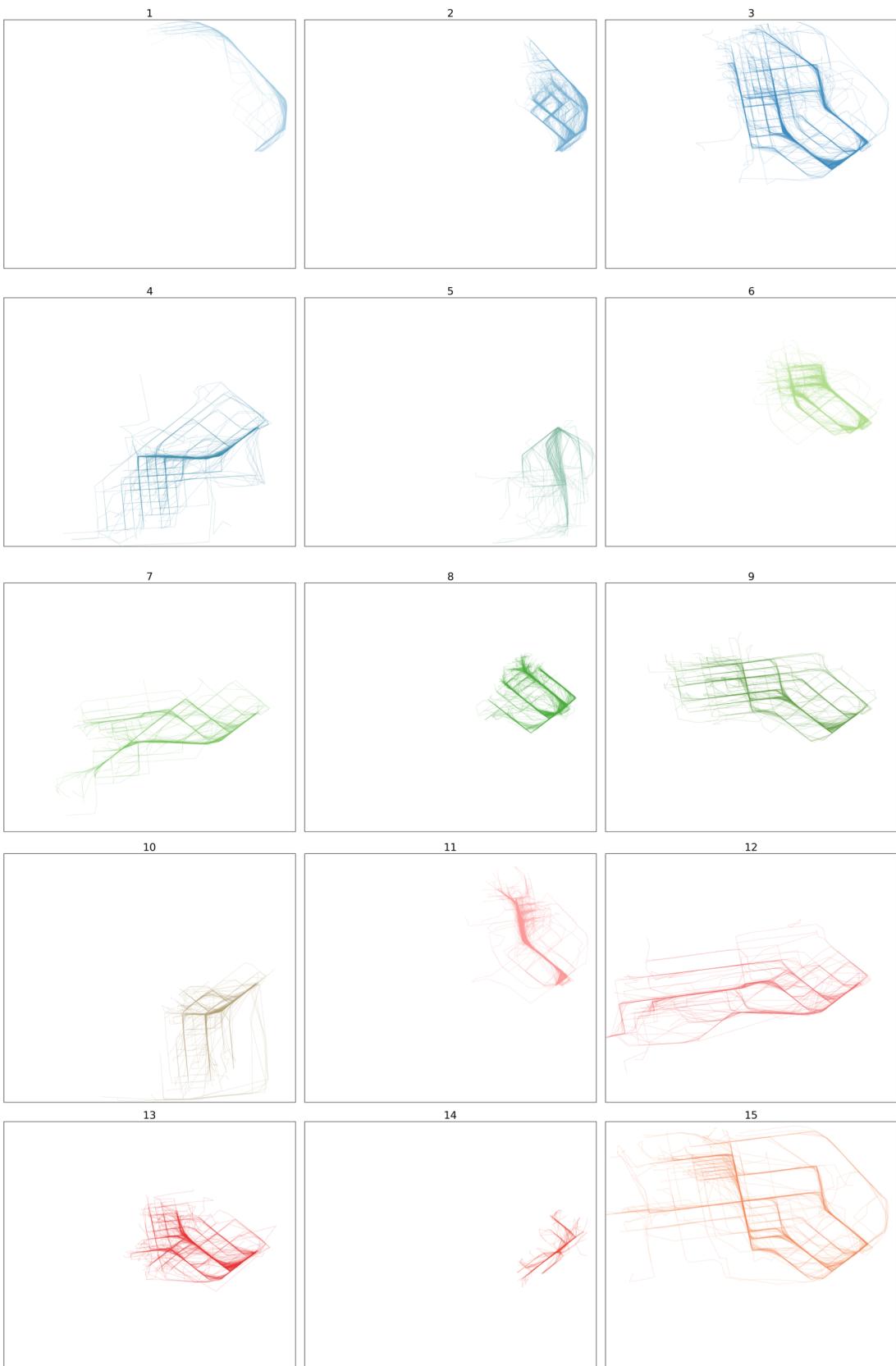


Figure 39: Caltrain Hausdorff Clusters

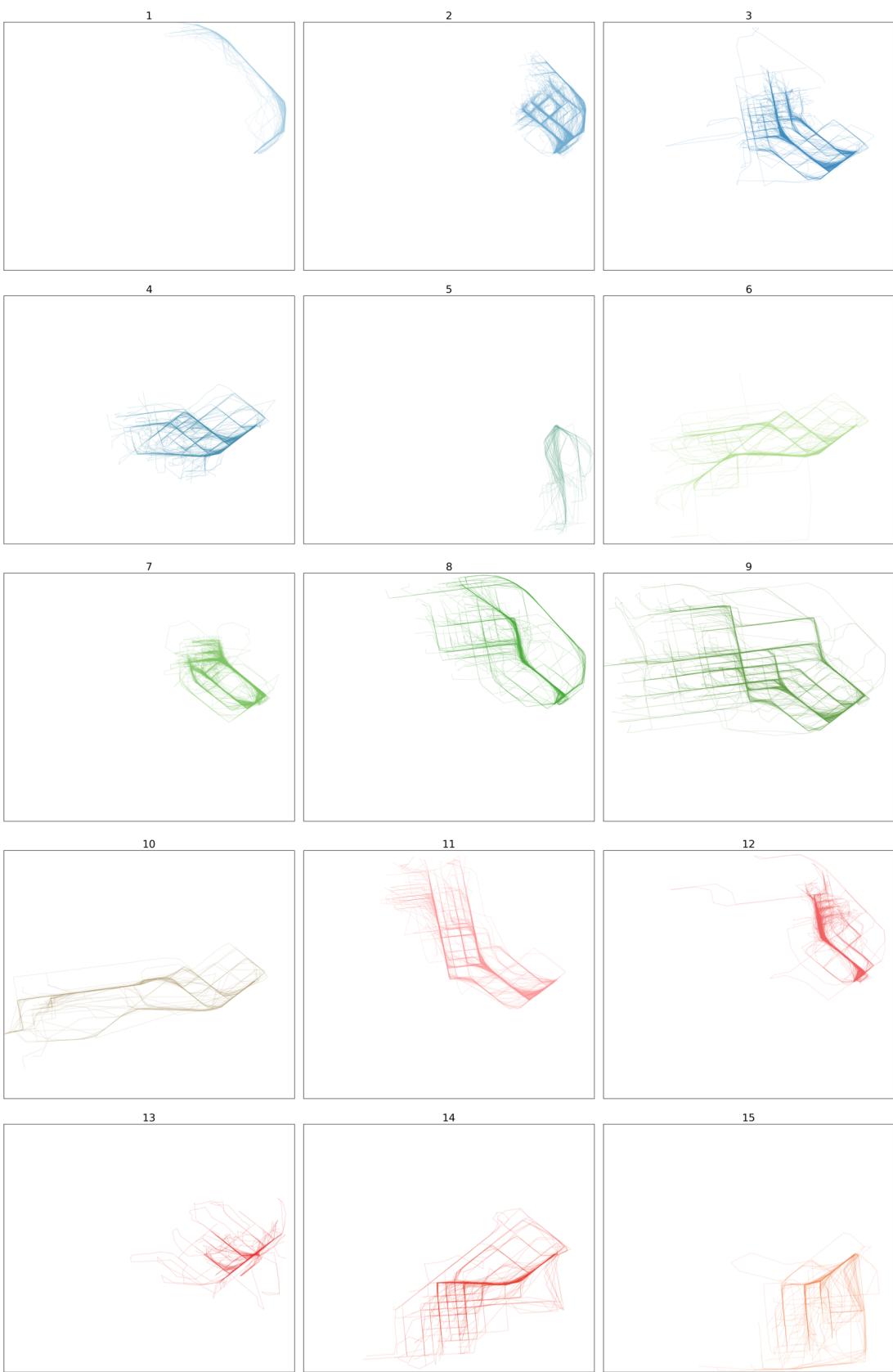


Figure 40: Caltrain Frechet Clusters

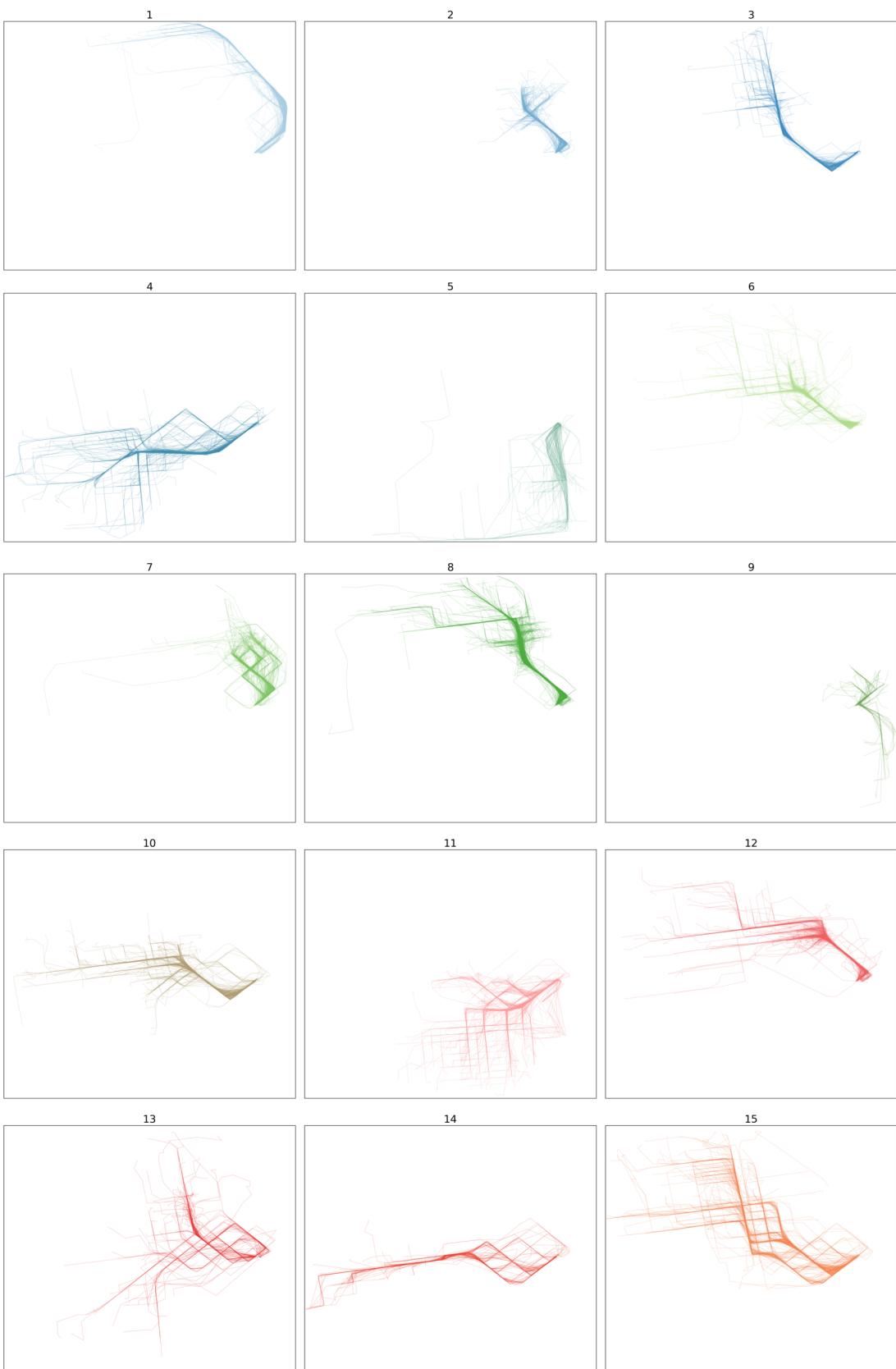


Figure 41: Caltrain HCSIM Clusters

6 Conclusions

Clustering is an efficient way to analyze and find the massive, hidden, unknown and interesting knowledge in large scale dataset, which facilitates the rapid development of data mining technology in recent decades. With the development of location based service, moving object clustering becomes a burgeoning topic in related fields as an essential part of data mining technology. Clustering of non Euclidean objects deeply relies on the choice of a proper distance. In this thesis, we presented different distances focusing on different features of such objects. Based on the experiment, shape-based distances especially give better clusters.

References

- Abbaspour, R. A., Shaeri, M., & Chehreghan, A. (2017). A method for similarity measurement in spatial trajectories. *Spatial Information Research*, 25(3), 491–500.
- Aghabozorgi, S., Shirkhorshidi, A. S., & Wah, T. Y. (2015). Time-series clustering—a decade review. *Information Systems*, 53, 16–38.
- Atev, S., Masoud, O., & Papanikolopoulos, N. (2006). Learning traffic patterns at intersections by spectral clustering of motion trajectories. In *2006 ieee/rsj international conference on intelligent robots and systems* (pp. 4851–4856).
- Berkhin, P. (2006). A survey of clustering data mining techniques. In *Grouping multidimensional data* (pp. 25–71). Springer.
- Besse, P., Guillouet, B., Loubes, J.-M., & François, R. (2015). Review and perspective for distance based trajectory clustering. *arXiv preprint arXiv:1508.04904*.
- Bian, J., Tian, D., Tang, Y., & Tao, D. (2019). Trajectory data classification: A review. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(4), 1–34.
- Caliński, T., & Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1), 1–27.
- Campello, R. J., Moulavi, D., & Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In *Pacific-asia conference on knowledge discovery and data mining* (pp. 160–172).
- Chen, J., Wang, R., Liu, L., & Song, J. (2011). Clustering of trajectories based on hausdorff distance. In *2011 international conference on electronics, communications and control (icecc)* (pp. 1940–1944).
- Chen, Z., Shen, H. T., Zhou, X., Zheng, Y., & Xie, X. (2010). Searching trajectories by locations: an efficiency study. In *Proceedings of the 2010 acm sigmod international conference on management of data* (pp. 255–266).

- Cheng, Z., Jiang, L., Liu, D., & Zheng, Z. (2018). Density based spatio-temporal trajectory clustering algorithm. In *Igarss 2018-2018 ieee international geo-science and remote sensing symposium* (pp. 3358–3361).
- Ertöz, L., Steinbach, M., & Kumar, V. (2003). Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 siam international conference on data mining* (pp. 47–58).
- Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (Vol. 96, pp. 226–231).
- Faloutsos, C., Ranganathan, M., & Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. *Acm Sigmod Record*, 23(2), 419–429.
- Fowlkes, E. B., & Mallows, C. L. (1983). A method for comparing two hierarchical clusterings. *Journal of the American statistical association*, 78(383), 553–569.
- Fränti, P., & Mariescu-Istodor, R. (2019). Averaging gps segments challenge 2019. *unpublished work*.
- Franti, P., Virmajoki, O., & Hautamaki, V. (2006). Fast agglomerative clustering using a k-nearest neighbor graph. *IEEE transactions on pattern analysis and machine intelligence*, 28(11), 1875–1881.
- Gaffney, S., & Smyth, P. (1999). Trajectory clustering with mixtures of regression models. In *Proceedings of the fifth acm sigkdd international conference on knowledge discovery and data mining* (pp. 63–72).
- Gariel, M., Srivastava, A. N., & Feron, E. (2011). Trajectory clustering and an application to airspace monitoring. *IEEE Transactions on Intelligent Transportation Systems*, 12(4), 1511–1524.
- Han, J., Pei, J., & Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.
- Hu, W., Xiao, X., Fu, Z., Xie, D., Tan, T., & Maybank, S. (2006). A system for learning statistical motion patterns. *IEEE transactions on pattern analysis and machine intelligence*, 28(9), 1450–1464.

- Ignacio Gonzalez, S. L., et al. (2017). Hierarchical clustering tutorial [Computer software manual]. Retrieved from http://genoweb.toulouse.inra.fr/~formation/CATIBIOS4BIOL_stats/Learning_clustering_current.pdf
- Keogh, E. J., & Pazzani, M. J. (2000). Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth acm sigkdd international conference on knowledge discovery and data mining* (pp. 285–289).
- Kotsiantis, S., & Pintelas, P. (2004). Recent advances in clustering: A brief survey. *WSEAS Transactions on Information Science and Applications*, 1(1), 73–81.
- Kriegel, H.-P., Kröger, P., Sander, J., & Zimek, A. (2011). Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3), 231–240.
- Kruskal, J. B. (1983). An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM review*, 25(2), 201–237.
- Lee, J.-G., Han, J., & Whang, K.-Y. (2007). Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 acm sigmod international conference on management of data* (pp. 593–604).
- MacQueen, J., et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth berkeley symposium on mathematical statistics and probability* (Vol. 1, pp. 281–297).
- Mariescu-Istodor, R. (2013). Detecting user actions in mopsi. *University of Eastern Finland School of Computing Thesis*.
- Mariescu-Istodor, R., & Fränti, P. (2017). Grid-based method for gps route analysis for retrieval. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 3(3), 1–28.
- Moayedi, A., Abbaspour, R. A., & Chehreghan, A. (2019). An evaluation of the efficiency of similarity functions in density-based clustering of spatial trajectories. *Annals of GIS*, 25(4), 313–327.
- Myers, C., Rabiner, L., & Rosenberg, A. (1980). Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(6), 623–635.

- Pfeifer, P. E., & Deutrch, S. J. (1980). A three-stage iterative procedure for space-time modeling phillip. *Technometrics*, 22(1), 35–47.
- Piorowski, M., Sarafijanovic-Djukic, N., & Grossglauser, M. (2009). *Crawdad data set epfl/mobility (v. 2009-02-24)*.
- Priestley, M. (1980). State-dependent models: A general approach to non-linear time series analysis. *Journal of Time Series Analysis*, 1(1), 47–71.
- Qian, H., & Lu, Y. (2017). Simplifying gps trajectory data with enhanced spatial-temporal constraints. *ISPRS International Journal of Geo-Information*, 6(11), 329.
- Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20, 53–65.
- Steinley, D. (2004). Properties of the hubert-arable adjusted rand index. *Psychological methods*, 9(3), 386.
- Su, H., Liu, S., Zheng, B., Zhou, X., & Zheng, K. (2020). A survey of trajectory distance measures and performance evaluation. *The VLDB Journal*, 29(1), 3–32.
- Sultana, S. I. (2020). How the hierarchical clustering algorithm works [Computer software manual]. Retrieved from <https://dataaspirant.com/hierarchical-clustering-algorithm/>
- Vasquez, D., & Fraichard, T. (2004). Motion prediction for moving objects: a statistical approach. In *Ieee international conference on robotics and automation, 2004. proceedings. icra'04. 2004* (Vol. 4, pp. 3931–3936).
- Waga, K., Tabarcea, A., Chen, M., & Fränti, P. (2012). Detecting movement type by route segmentation and classification. In *8th international conference on collaborative computing: networking, applications and worksharing (collab-oreatecom)* (pp. 508–513).
- Wang, H., Su, H., Zheng, K., Sadiq, S., & Zhou, X. (2013). An effectiveness study on trajectory similarity measures. In *Proceedings of the twenty-fourth australasian database conference-volume 137* (pp. 13–22).

- Xu, R., & Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3), 645–678.
- Ying, J. J.-C., Lee, W.-C., Weng, T.-C., & Tseng, V. S. (2011). Semantic trajectory mining for location prediction. In *Proceedings of the 19th acm sigspatial international conference on advances in geographic information systems* (pp. 34–43).
- Ying, J. J.-C., Lu, E. H.-C., Lee, W.-C., Weng, T.-C., & Tseng, V. S. (2010). Mining user similarity from semantic trajectories. In *Proceedings of the 2nd acm sigspatial international workshop on location based social networks* (pp. 19–26).
- Yuan, G., Sun, P., Zhao, J., Li, D., & Wang, C. (2017). A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, 47(1), 123–144.
- Zhao, L., & Shi, G. (2019). A trajectory clustering method based on douglas-peucker compression and density for marine traffic pattern recognition. *Ocean Engineering*, 172, 456–467.