# Impact of Trajectory Segmentation on Discovering Trajectory Sequential Patterns

Somayah Karsoum and Le Gruenwald
School of Computer Science
University of Oklahoma
Norman, OK USA
{somayah.karsoum,ggruenwald}@ou.edu

Eleazar Leal
Computer Scicen Department
University of Minnesota Duluth
Duluth, MN USA
eleal@d.umn.edu

*Abstract*— **Nowadays, location-aware devices, such as GPS, provide a huge volume of spatial-temporal data. Analyzing the data to understand the behavior of objects (e.g. people) could be beneficial in many application areas. Due to the spatial and temporal nature and their complexity, researchers have developed various data mining techniques such as trajectory segmentation, which splits the trajectories into sub-trajectories, to prepare them for the mining step. A central issue in discovering knowledge is choosing an appropriate trajectory segmentation technique. In this paper, we provide a comparative study on two trajectory segmentation techniques, density-based and grid-based, when applied to sequential patterns discovery. We conducted experiments using two real-life datasets to evaluate the performance of the methods in terms of execution time and their impact on discovering the sequential patterns. The experimental results showed that the density-based is more efficient, while the grid-based is more effective.**

***Keywords—****Sequential trajectory patterns; Trajectory segmentation; density-based; grid-based*

## I. INTRODUCTION

The goal of sequential pattern mining is to find a set of repeated patterns that occur together in some sequences ordered based on time. The popularity of mobile devices and the improvement in location-based services have led to the collection of huge amounts of spatial-temporal data in many applications. This means that the movements of moving objects (e.g. vehicles, people, or animals) including their latitudes and longitudes, and the timestamp at each location can be recorded. Thus, the sequence of spatial coordinates and associated timestamps, known as a trajectory, can be collected by using these devices. The knowledge about the movement behavior of objects helps us understand, analyze, and predict their frequent patterns. Therefore, discovering frequent sequential patterns from trajectory data has become an attractive area of recent research [4], [6], and [9].

For instance, two trajectories $T_1$ and $T_2$ have the following sequences of locations $l_i$:

$T_1$: $l_1 \rightarrow l_2 \rightarrow l_8$ $T_2$: $l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_8$

We can see that these two trajectories have a common sequence $l_1 \rightarrow l_2 \rightarrow l_8$ of visiting order [27]. If the percentage of the records in the database that contain this pattern, which is called support, exceeds the given minimum support threshold (min_sup), the pattern is reported as a frequent sequential pattern.

The spatial and temporal information that can be extracted from trajectories can make many applications more effective and beneficial. For example, in tourism, finding the repeated paths in a given geospatial region could improve the recommendation systems to advise tourists to visit specific locations and follow the suggested sequences of regularly toured locations [28]. Because of the complex nature of trajectory data, splitting them into disjoint, small, and less complex sub-trajectories based on some criteria would help extract the trajectory sequential patterns. This process is called *trajectory segmentation*. Let us have an example to demonstrate the advantage of using this mechanism. Suppose that we have three trajectories of three people who go to a café (represented as a dot circle) as shown in Fig 1. Even though they are in the same cafe, each one has a different location because they are sitting at different tables with different time stamps recording when they arrived. Therefore, even the cafe appears in the three trajectories of the three people, meaning visiting the café is a common activity of these people, the actual data point associated the café for each person has different values (coordinates and timestamp). Thus, an algorithm for sequential pattern mining cannot identify the repeated movements of people because of each time a different location (coordinates and timestamp) is recorded. On the contrary, if the location (cafe's coordinate) is recorded instead of exact different locations, the algorithm will discover repeated patterns easier. Thus, the location coordinates of these points can be different but belong to the same place [6]. To solve this issue, researchers use trajectory segmentation techniques to gather the data points that belong to the same place of interest into one group or region.

From various categories, we choose density-based [17] and grid-based [4] to conduct the comparative study because we are not working on a particular application such as breaking the road-network into road segments. By using one of these two techniques, the sequence points of a trajectory are transformed into a sequence of regions, then the frequent sequential patterns are discovered from these sequences of regions. As mentioned above, trajectory segmentation is an essential step to prepare trajectories, but it could inhibit

finding the correct frequent sequential patterns or may miss some of them. We will explain this situation in details in Section III.B and Section III.C

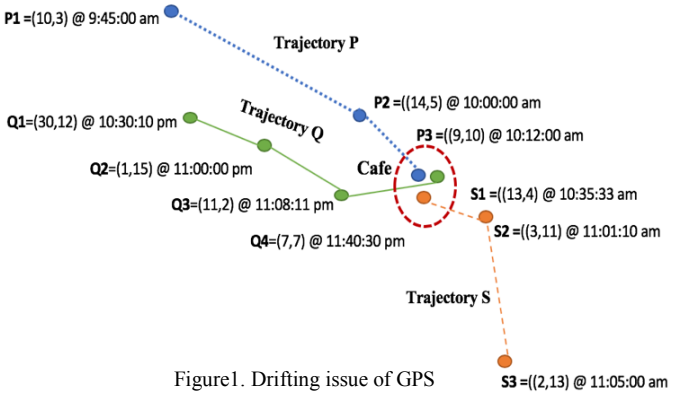



Figure1. Drifting issue of GPS

Our paper highlights the impacts of trajectory segmentation specifically on discovering frequent sequential patterns from trajectory data. The frame work of the paper is as follows: first, we apply the density and grid based algorithms on a raw trajectory dataset; then we run a well-known sequential pattern mining algorithm called Sequential Pattern mining with Restricted Search SPADE [26] on the output of each of the two techniques to discover the frequent trajectory patterns; and we then compare the results in terms of execution time and number of patterns discovered.

The remainder of the paper is organized as follows. Section II reviews the related work. Section III introduces the basic definitions and describes the two existing techniques that have been used for trajectory segmentation in this study, cluster-based and grid-based, and presents the frequent sequential pattern mining algorithm used in our experiments, SPADE [26]. Section IV presents the results of the experiments on three real trajectory datasets: Geolife [29] ,[30], Deer [13]. Finally, Section V, presents the conclusions and future research directions.

## II. Related Work

Discovering sequential patterns was introduced for the first time in [1]. Since then, many methods and techniques have been proposed to mine sequential patterns. For example, the GSP algorithm [23] inspired a technique to generate fewer candidates based on the Apriori algorithm for frequent itemset mining. Therefore, it is classified as an Apriori-based algorithm. In contrast, pattern-growth based algorithms, such as FreeSpan [11] and PrefixSpan[19], used the concept of database projection to improve the performance of mining sequential patterns. Recently, due to a large volume of data generated, many of the proposed sequential pattern mining algorithms tried to handle huge amounts of sequences and large databases efficiently. Ha-GSP [14] adopted the principles of GSP and implemented it on the Hadoop platform for solving the limited computing capacity and insufficient performance with massive data of the traditional GSP. MR-PrefixSpan [25] used the MapReduce platform to implement the parallel version of PrefixSpan to mine sequential patterns on a large database. More literature reviews about the state-of-art sequence mining algorithms can be found in [16], and the state-of-art parallel sequence mining algorithms in [2].

On the other hand, discovering the frequent sequential patterns from trajectory data also earned high attentions from researchers as a special case of frequent sequential pattern mining. Mining frequent sequential patterns from trajectory data refers to all the processing steps starting from collecting raw trajectory points from a sensor to providing understandable trajectory patterns. One of the most important pre-processing steps is dividing trajectories into sub-trajectories because the repeated patterns could occur not only among the whole trajectories but also among the sub-trajectories. One of the first studies on mining sequential spatiotemporal patterns was [24].The raw trajectories were defined as lists of spatial locations over time, then a Depth-First-Search-Like algorithm (DFS_MINE) was proposed to discover long sequences from the trajectory database. [8] incorporated the temporal dimension in the sequential patterns by defining temporally annotated sequences (TAS), and [9] proposed the Trajectory Pattern algorithm (T-pattern) to extract a set of TAS. The T-pattern used the grid-based technique, which broke down the space into regions based on a user-defined parameter (a cell size) and transferred the points series to a sequence of cell IDs, to identify the densest regions and take advantage of PrefixSpan to discover the repeated sequences of hot regions. Also, [32] applied the grid-based approach to provide a transformed trajectory for a user to represent the sequences of regions that he/she frequently visits. Then, they used a sequential pattern mining technique to discover the similar movement behavior from a set of transformed trajectories among these hot regions. In contrast, other studies used a different technique that depends on data clustering to identify the hot regions such as [18] and [6]. For parallel frequent sequential algorithms, some works such as [21], [20], and [15], exploited the benefits of the parallel environment and implemented parallel algorithms for trajectory mining sequential patterns to deal with the huge amount of trajectory data.

The prior studies discussed in this section used some techniques to segment trajectories without considering some issues such as sampling rate differences or velocity variation that could affect the main purpose, which is to discover the complete trajectory patterns. Also, they did not provide reasons of why they chose a specific technique. To the best of our knowledge, there is no report studying the impact of trajectory segmentation from a data mining aspect. In our study, we choose two of the most popular techniques to divide trajectories into sub-trajectories and study their impact on discovering trajectory patterns.

## III. Background

### A. Preliminaries

**Definition III.1.** (*Trajectory*) Capturing the movement of an object (e.g. people, animals, vehicles) in a specific period of time in geographical space will produce a trajectory T, which has a series of ordered data points $T = \{P_1, P_2, \ldots\ldots\ldots, P_m\}$, where each data point $P_i$ contains two geospatial coordinates, latitude

(x) and longitude (y), and one time stamp coordinate (t) such that $P_i = (x_i, y_i, t_i)$ [5].

**Definition III.2.** (*Sub-trajectory*) Given a trajectory T and a time interval $[t_i, t_j]$, where $t_i < t_j$, the sub-trajectory t is defined as a subset of T such that t contains all points between $t_i$ and $t_j$ of trajectory T.

**Definition III.3.** (*Frequent sequential trajectory pattern*) Suppose *I* is a set of m unique items (or locations over time ) such that $I = \{i_1, i_2, \ldots, i_m\}$. In general sequence mining algorithms, any unordered subset of *I* is called an itemset denoted as $(i_1, i_2, \ldots, i_n)$, where $i_j$ is an item, and each itemset represents a set of items happening at the same time [23]. Due to the special nature of trajectory data, we do not have an itemset with multiple items at one timestamp because a moving object cannot be in different locations at the same time. A sequence $S = < e_1\ e_2\ e_3 \ldots e_k>$ is an ordered item based on the time-stamps. In other words, the items in a sequence are ordered in the ascending order of their time-stamps, where $e_1$ occurs before $e_2$, $e_2$ occurs before $e_3$, and so on. If a sequence has k items, it is called a k-sequence. Suppose we have two sequences $\alpha = < a_1\ a_2\ a_3 \ldots a_m >$ and $\beta =< b_1\ b_2\ b_3 \ldots b_n >$, we say $\alpha$ is a subsequence of $\beta$ (it is a super-sequence) if there exists the integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$, …., $a_m \subseteq b_{in}$. The support (frequency) of a sequence S in a sequence database SD, denoted as $\sigma$ (S, SD), is defined as the number of sequences in SD that contain S (also it is called support count) divided by the total number of sequences in SD [23].

$$\sigma(S, SD) = \frac{\text{\# sequences containing S}}{\text{Total \# sequences in SD}} \qquad (1)$$

In order to say S is a frequent sequence, the percentage of times (support) that it occurs in the sequence database must be at least equal to the user-defined minimum support threshold Ultimately, the final objective of a frequent sequential trajectory mining algorithm is to find the sequences (ordered based on time-stamps) of the locations that many moving objects visit.

### *B. Density-Based Trajectory*

By using unsupervised methods, a group of trajectories (or sub-trajectories) is gathered into a cluster based on similar features such as time intervals or locations. The final number of clusters and the group of objects that belong to a cluster are unknown. After the classes are decided, all the data points of the trajectories are labeled with their corresponding clusters' IDs. All the trajectories within the same cluster should be very similar to each other. In contrast, trajectories from different clusters should be very different from each other in terms of the chosen features. There are various types of clustering algorithms that differ from each other in terms of how to define clusters and find them. One of these types is density-based algorithms, which define a cluster as a density area of objects [7]. Therefore, a threshold of the density data points within a region is used here to determine the relevant data that belong to a one cluster. One of the density-based clustering algorithms is Density-Based Spatial Clustering of Applications with Noise DBSCAN [7]. Before using this algorithm two input parameters are needed to initialize: Epsilon (ε) and minimum number of points (MinPts). The first parameter ε is used to determine the radius area around a point. DBSCAN chooses a point, say A, randomly, then if any other points are located within the ε(A), which is the radius area around point A, they are identified as the neighbors of A. The second parameter (MinPts) is used to decide whether A is labeled as core point or not . If the number of points within ε , including the point itself, is greater than or equal to MinPts, the point is labeled as core point. Otherwise, if that point has the number of points less than MinPts but it is a neighborhood of a core point, the point is labeled as border point. The algorithm labels a point as a noise point if it is either not a core or a border point. For each connected group of core points, the algorithm makes them into a separate cluster. After that, DBSCAN assigns the border point to a cluster that associate with the closest core point.

The main issue is how to determine the values of these two parameters: ε and MinPts because they decide the result's rigor. Therefore, understanding the nature of the data is very important in order to allow DBSCAN to produce meaningful results. Using this density based technique to segment trajectories according to the dense areas will help us group all points that belong to the same place into a cluster. It means that raw trajectories are transformed from sequences of points to sequences of cluster IDs. For example, as shown in Fig. 2b, we have three trajectories, each with a different number of points, recording in the second column in Fig. 2a. We identify three clusters after DBSCAN is applied, where each cluster is denoted as a dotted circle. We then replace a sequence of points that reside in the first cluster with the ID of the first cluster (CId 1), the second cluster with CId 2, and the third cluster with CId 3 as shown in the third column in Fig.2a.

Obviously, the number of points (the density of an area) is an important parameter to identify a cluster. We may miss some clusters if we do not have a sufficient number of points; thus, we may also miss some trajectory patterns. One of the issues that we face when we deal with trajectory data is the sample rate difference between difference trajectories. The sampling rate is defined as the time interval between two consecutive points. For example, in Fig. 2 we can see that Trajectory 1 has a higher sampling rate than the other two trajectories because the points are collected, for instance, every one minute, while Trajectory 2 has the lowest sampling rate among trajectories because its points are collected every five minutes. Another issue is the speed of a moving object. Let us assume that the Trajectory 1 in Fig.2 shows the movement of an elderly person, while the trajectory 2 presents the movement of a young person. If we assume their points are collected every ten seconds, same sampling rate, we still have a different number of point as we can see within the circle. The reason behind this is the elderly people walk slower than the young people, so the number of assembled points from their trajectories are different within the same area. Let us suppose that a user defines the MinPts = 8 and Min_sup = 2. Because of the sampling rate issue, after applying DBSCAN on the sequential database in Fig. 2, only two clusters are defined (Cluster 1 and Cluster 3).

Cluster 2 is missed because its number of points, which is six points, is less than the threshold MinPts. This causes the trajectory sequential pattern $CId_1 \rightarrow CId_2$ to be missed, which means that using a density-based technique to group trajectories without considering other issues such as the sampling rate difference between different trajectories could impact the discovery of trajectory patterns.

### C. Grid-Based Trajectory Segmentation

Another way to group the points that belong to the same geographical area is using a regular grid. The space that contains trajectories is divided into a number of cells based on a user-defined parameter called cell size, where each cell will represent a region. Each cell has its ID as shown in Fig. 3. Then, trajectories are mapped to the grid, so the raw trajectories are transferred from points to cell IDs[4]. Using this naïve method to group the points may lead to missing some trajectory patterns because if the points of the trajectories fall into different adjacent cells, some sequential trajectory patterns will be missed. For example, as shown in Fig. 3, we have three trajectories: the first row in the table and black squares (Trajectory 1) are used to illustrate the movement of T1; the second row in the table and white squares (Trajectory 2) are used for T2; and the third row in the table and cross symbol (Trajectory3) are used for T3.

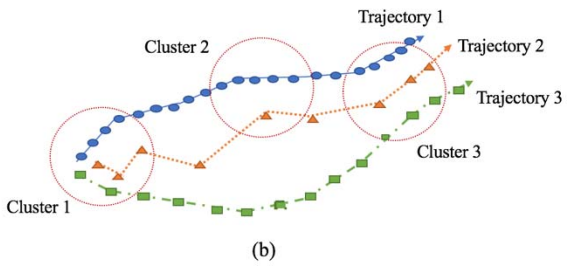
After we apply the grid segmentation technique, all the points that belong to a specific cell are replaced by the ID of the cell (denoted as C) as shown in the second column in Fig. 3. While we mine the database to discover trajectory patterns, assuming min-sup = 3, the pattern $C_6 \rightarrow C_8$ is missed because the area in dotted circle has a black square point of Trajectory 1, which is in cell $C_9$ rather than cell $C_6$. If the size of the cells has been chosen to be bigger, that black square could belong to cell $C_6$, which means the frequency of the pattern $C_6 \rightarrow C_8$ is equal to min_sup, and thus this pattern is frequent and should have been discovered. Therefore, the size of the cells, which is defined by the user, has an enormous impact on discovering frequent sequential trajectory patterns.

### D. Frequent trajectory sequential patterns

We use a well-known algorithm for discovering sequential patterns called SPADE [26] and adopt the implementation of

| Trajectory | Sequence of points | Sequence of segments |
|---|---|---|
| T1 | $P_1, P_2, P_3$....., $P_{143}$ | $CId_1$ ,$CId_2$ , $CId_3$ |
| T2 | $P_1, P_2, P_3$....., $P_{186}$ | $CId_1$ , $CId_2$ , $CId_3$ |
| T3 | $P_1, P_2, P_3$....., $P_{276}$ | $CId_1$ , $CId_3$ |

(a)




(b)

Fig.2. Density-based Trajectory Segmentation

| Trajectory | Sequence of points | Sequence of segments |
|---|---|---|
| T1 | $P_1, P_2, P_3$....., $P_{143}$ | $C_2$ ,$C_9$ , $C_8$, $C_4$ , $C_2$ |
| T2 | $P_1, P_2, P_3$....., $P_{186}$ | $C_1$ , $C_2$ , $C_3$ , $C_6$, $C_8$ , $C_4$, $C_1$ |
| T3 | $P_1, P_2, P_3$....., $P_{276}$ | $C_2$ , $C_3$, $C_6$ , $C_8$ , $C_7$ , $C_4$ , $C_2$ |

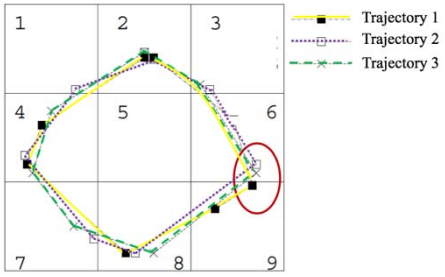



Fig.3. Grid –based Trajectory Segmentation[4]

the algorithm from a package in R [3]. The sequence database contains tuples of trajectories and each trajectory has a sequence of points. For example, Trajectory1 = <(10, 3, 9:45 am),(14, 5,10:00 am)>. For simplicity, we use the letters to identify each point. Thus, Trajectory 1 is <A, B> and so on as shown in Fig. 4.

SPADE uses the vertical format of a sequential database, so each item (location) has an id-list that contains the sequence ID (trajectory ID) and event ID (time stamp). Table 1 shows the vertical database of trajectory sequence database in Fig. 4. Let us take the item D and build its id-list. The SPADE builds a list contains two columns: the first one has the Sequence IDs (Trajectory IDs) that item D appears in, and the second column has the Event IDs which representing the position of item D within a particular sequence. When the algorithm scans the database, it finds that item D occurs in sequence 2 at the position 3, sequence 3 at the positions 1 and 3, and sequence 4 at position 1. As we can see, by scanning the vertical database we can compute the support count of items. For instance, location A exists three times in the database. At the end of this step, all the frequent 1-sequences (i.e. the frequent sequences each member of which is 1 location only) are determined after they are compared against the min-sup. Let us suppose the min_sup = 40%, then the frequent 1-sequences ={ A, B, C, D}.

For finding the 2-sequences (i.e. the sequences each member of which has 2 locations), only the id-lists of the frequent 1-sequences are used. The algorithm joins the id-lists of two frequent sequences that have the same SID if the event Id of the first item occurs before the event ID of the second item (i.e. the timestamp of the first item is less than the timestamp of the second item). Then, the result of such a join is shown in the id-list for the new combination. For example, location A and location B have the two common SIDs, 1 and 5, and A always appears before B. The join result is shown in Table 2. The frequent 2-sequences after applying this step is { < A, B >, < D, B >, < A, C >}. Again, from the id-lists of the 2-sequences, the frequent sequences can be identified and used to generate the frequent 3-sequences. This process is repeated until the algorithm enumerates all frequent sequences.

The cSPADE algorithm available as an R package is a modified version of SPADE, which allows users to define some constraints such as max/ min time gaps between adjacent elements in the pattern, width or length condition on the

| TID | Sequences |
|---|---|
| 1 | < A B> |
| 2 | <A,C,D> |
| 3 | <D,B,D> |
| 4 | <D,B,E> |
| 5 | <A,B,C> |

Fig. 4. Trajectory Sequence Database

TABLE 1. Vertical Trajectory Sequence

| A | | B | | C | | D | | E | |
|---|---|---|---|---|---|---|---|---|---|
| SID | EID | SID | EID | SID | EID | SID | EID | SID | EID |
| 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 4 | 3 |
| 2 | 1 | 3 | 2 | 5 | 3 | 3 | 1,3 | | |
| 5 | 1 | 4 | 2 | | | 4 | 1 | | |
| | | 5 | 2 | | | | | | |

TABLE 2. Id-lists for some 2-sequences

| AB | | | AC | | |
|---|---|---|---|---|---|
| SID | EID (A) | EID (B) | SID | EID (A) | EID (C) |
| 1 | 1 | 2 | 2 | 1 | 2 |
| 5 | 1 | 2 | 5 | 1 | 3 |

sequences, or a time window for an entire sequence that must occur within it. In this paper, we do not set any constraint because we want to discover naïve sequential patterns from trajectory data. Therefore, we use word SPADE instead of cSPADE.

## IV. Performance Analysis

In this section, we report the results of the experiments to compare the performance of the density-based and grid-based segmentation techniques and their impacts on trajectory pattern mining

### A. Experimental Setup

All experiments were performed on a computer with a 2.7 GHz Intel Core i5 processor with 8 gigabyte main memory, running MacOS High Sierra version 10.13.4. The grid-based algorithm is implemented in R using RStudio version 1.1.383. In contrast, the implementations of both DBSCAN and SPADE are adapted from the existing packages in R called: dbscan and arulesSequences [3].

#### 1) Datasets

For the experiments, we use two real-life trajectory datasets: Geolife and Deer. The first one, Geolife, was collected by Microsoft Research Asia [31], [29], and [30] after tracking the movements of 182 individuals in Beijing, China from 2007 to 2012. The second one, Deer, was collected for the Starkey project [13]. The movements of 32 deer (20,000+ points) were obtained in northeastern Oregon from 1993 to 1996. To prepare the data for the experiments, we removed the unnecessary attributes from each dataset and kept only the following attributes: object ID, latitude, longitude, and timestamp. Then, to interpret the results more easily, we re-scaled the trajectories data values to measure how many standard deviations a value is from its mean. Thus, the coordinates of the points were standardized.

#### 2) Determing Optimal Values of Paramaters

The two segmentation techniques require values of several parameters before we start performing them. These values may have a significant impact on the performance. However, these values may not be known in advance, or some available values, which are chosen by other studies, are not suitable because of the nature of the datasets. Therefore, we reviewed other studies and used several tools to help us choose the most optimal values for these parameters.

The first technique, DBSCAN, requires two parameters: minimum number of points (MinPts) and epsilon (ε). For (MinPts), there is no automatic method for choosing its optimal value because such value depends on the nature of the dataset. As a result, we reviewed the existing literature and adopted their values for this parameter. [12] used DBSCAN for GeoLife dataset and selected ε = 0.001 and MinPts to be = 60. For the Deer dataset, we chose the default value for MinPts for two-dimensional data, which is 4 [7][22]. For (ε), there is the k-nearest neighbor distance method (KNN)[10] that can be used to estimate the value of (ε).The idea behind the knee point is that the k-distance for the core and border points is expected to be within a particular range, while the noise points' k-distance is much greater. Thus, we can detect a change in a curved shape. In Fig. 5, a knee is visible at around a 4-NN distance of ≅ 0.05, which represents the value of ε for the Deer dataset.

The second technique, Grid, requires one parameters: Cell size. We chose different sizes for the cell, as shown in Table 3, and calculated the overall SSE of all cells by using the following formula (2):

$$\mathrm{SSE} = \sum_{j}^{k} \sum_{i=1}^{n} (x_i - \bar{x})^2 \qquad (2)$$

where $x_i$ is an actual point, $\bar{x}$ is the mean of points, n is the number of points of a cell, and k is the number of cells.

We chose the cell size to be = 3x3 because it gave a low SSE as shown in Table 3. Although the next two cell sizes (3.5 x 3.5 and 4 x 4) have lower SSE, they produced only one cell. This does not help us discover the frequent sequential patterns at the same procedure for Deer and chose the cell size to be equal to 2x2.

#### 3) Experimental Parameters

We describe the dynamic parameters, their ranges, and their default values of this analysis as it is summarized in Table 4.

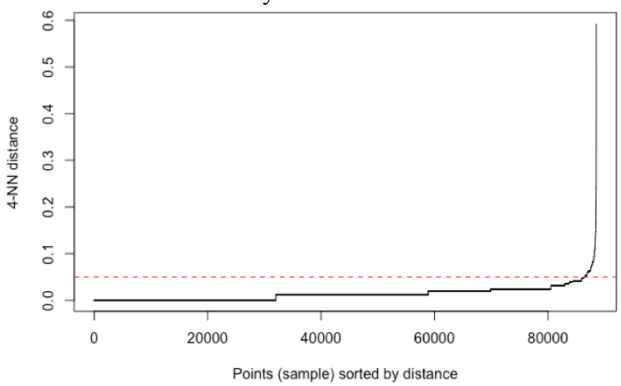



Fig.5. Optimal value of (ε) for Deer dataset

TABLE 3: Cell size vs. SSE measurement for GeoLife dataset

| Cell size | SSE | # Cells |
|---|---|---|
| 1 x 1 | 35025.3073 | 45 |
| 1.5 x 1.5 | 5623373.2063 | 20 |
| 2 x 2 | 11605.6514 | 7 |
| 2.5 x 2.5 | 20168.0529 | 6 |
| 3 x 3 | 11613.6555 | 5 |
| 3.5 x 3.5 | 2319.593 | 1 |
| 4 x 4 | 0.021037 | 1 |

When an experiment is conducted, we choose various values within its range and assign the default value for the other parameters. We assume different ranges for each dataset. For (ε) its default value is equal to the optimal one, which is decided based on the discussion in Section IV.A.2. and the default value is the median value of that range. We consider (MinPts) values in range from 40 to 80 for GeoLife, and from 1 to 8 for Deer. As we mentioned in Section IV.A.2, the default value of MinPts is = 60 and MinPts = 4 based on the literature review for GeoLife and Deer, respectively. For the cell size, we select the range to be from 1x1 to 4x4 from section IV.A.2. The default value is chosen to be equal to the 3x3 for GeoLife, and 2x2 for Deer, based on the technique that has been used in Section IV.A.2. The number of trajectories has different ranges based on the trajectories available for each dataset. Therefore, we choose the default value to be the mean of the range. The GeoLife's range is from 1 to 182 and its default value (mean) is 91, and Deer's range is from 1 to 32 and its mean is 16. The support (min-sup) has a range from 20% to 80% with the default value = 50%, which is the median of the interval.

*4) Evaluation Metrices*

We measure the impact of trajectory segmentation techniques on discovering trajectory sequential patterns in terms of the execution time (ET) and the number of discovered patterns (NumOfPatt). The execution time (in seconds) is recorded starting from the moment that a dataset has been read to the moment that SPADE produces the patterns. The more patterns SPADE discovers after applying one of the trajectory segmentation techniques, the more accurate and complete results, because [26] stated that SPADE discovers all frequent sequential patterns that satisfy the condition of which their frequency is greater than or equal to min-sup. Since we are using two different segmentation techniques: density-based, and grid-based, each one produces a set of sequential patterns that represent different granularities, which make these patterns not compatible. Therefore, we transfer the patterns to a low compatible level (point level) and compare the patterns that are generated from these two techniques to conclude if they are the same or not. More details are provided in Section IV.B.6.

TABLE 4: List of parameters for experiments

| Parameter Name | GeoLife Dataset | | Deer Dataset | |
|---|---|---|---|---|
| | Range of values | Default values | Range of values | Default values |
| ε | 0.0001 – 0.1 | 0.001 | 0.01 – 0.1 | 0.05 |
| MinPts | 40 - 80 | 60 | 1 - 8 | 4 |
| Cell size | 1×1 - 4× 4 | 3 × 3 | 1×1 - 4× 4 | 2 × 2 |
| Number of trajectories | 1- 182 | 91 | 1- 33 | 16 |
| min_sup | 20% - 80% | 50% | 20% - 80% | 50% |

## B. Experimental Results

Five sets of experiments are conducted to study the impacts of the five parameters (ε), MinPts, cell size, Number of trajectories, and min-sup) on the performance of the two techniques. Table 5, which shows the average performance of the two segmentation techniques: density-based and grid-based for the two datasets: GeoLife, and Deer.

The density-based technique takes less time to discover the frequent sequential patterns than the grid-based. DBSCAN first labels all points either core, border, or noise, then eliminates the noise points. That means the number of points now is less than the original dataset. After that, DBSCAN goes through core and border points and assigns a cluster ID to each point. However, the grid-based, in our implementation, first labels all points with its row and column index, and does not remove any points. That means the number of points is still the same as the original dataset. Then, the algorithm decides the cell ID for each point. Depending on this, when the algorithm converts the original sequence database in the new form of database in terms of cluster ID or cell ID, the length of sequences could be different. For the density-based, some of the points will not belong to any cluster since some of them are noise. Thus, they are removed from the sequence and the length of that sequence is shortened. In contrast, for grid-based, each point must belong to a cell, so the length of the sequence will be longer than the sequence in the density-based sequence database. Therefore, SPADE needs more time to mine frequent trajectory patterns from the grid-based sequence database.

Nevertheless, grid-based outperforms density-based in terms of how many patterns are discovered. The first reason is because the grid-based does not discard any points, and the strongest relationship appears when there are many points. Also, the length of the sequences in grid-based are longer than the length of sequences in density-based, especially if many points belong to adjacent cells. For instance, if we have a sequence $< P_1, P_2, P_3...., P_{10}>$ and each point belongs to the three adjacent cells alternately, the transformed sequence will look like $<C_1,C_2,C_3,C_1,C_2,C_3,C_1,C_2,C_3,C_1>$. Thus, the length of the sequence is larger, which allows SPADE to discover more patterns.

The second reason depends on how the algorithm treats the points. Grid-based depends only on one attribute (cell size), which corresponds to the radius (ε) parameter in the density-based. The DBSCAN has to deal with additional parameters (MinPts) and there are two types of points, after eliminating the noise points, the core and border points. Usually, the number of clusters relies on the core points, and the density of each cluster relies on the border points. When we perform the experiment

TABLE 5: Average of performance results

| Dataset | Density-based segmentation technique | | Grid-based segmentation technique | |
|---|---|---|---|---|
| | ET (sec) | NumOfPatt | ET (sec) | NumOfPatt |
| GeoLife | 5.2822 | 1 | 20.3453 | 3 |
| Deer | 1.5897 | 9 | 3.8445 | 2071 |

by using the default values of the parameters, we always have the same number of clusters or cells. However, the density of clusters is going to be different due to a different implementation of DBSCAN. Each implementation could have a different strategy to decide the cluster of a border point. Some of them may arbitrarily assign it to any cluster and others could assign the border point to the closest core point, if there are many of them around the border point and they belong to different clusters. Therefore, a border point could belong to a different cluster every time the DBSCAN is run. Thus, the frequency of a cluster ID occurrence in a sequence database varies from time to time, so some patterns could be lost. In order to understand how each technique will behave, in some experiments we separated the execution times, which is measured in seconds (sec), for each technique into four types: time required for reading input data (denoted as $T_1$) time needed by DBSCAN to assign cluster IDs to points, or by cells partition to assign cell IDs to points ($T_2$); time to convert the original sequence database to the new one in terms of cluster or cell IDs ($T_3$); and time to execute the sequential pattern mining algorithm, SPADE ($T_4$).

Since each one of the techniques required a user-defined parameter, we studied here the effect of changing the values of these inputs on discovering the frequent sequential patterns at the end.

*1) Impact of Cluster Radius (ε)*

While we increase the value of (ε), we observe that total execution time also increases. To understand the reasons , we studied many factors such as number of clusters and sub-execution times that are already mentioned in Section IV.B. We found that the number of clusters decreases because when the radius of a cluster increases, the number of core points will also increase due to the possibility of finding points that are equal to or more than MinPts. Therefore, more core points will be close enough to be put in the same cluster. Also, depending on that, the number of noise points reduces because the radius of a cluster is getting larger and larger, which means a noise point would be either a core point if it has equal or more points within (ε), or a border point that could be a neighbor of a core point. Minimizing the number of noise points leads to an increase in the numbers of core and border points, which means that DBSCAN needs more time to work on these points to decide which clusters they belong in.

The execution time of SPADE, $T_4$, relies upon the number of clusters. When the number of clusters increases, the number of cluster IDs that are used to replace the points also increases. Hence, the number of ID-lists increases, the number of candidates increases, and the time to check candidates' frequency also increases. In the end, we conclude that increasing the value of (ε) will also increase the overall execution time (ET).

For the number of patterns, we see that it is not predictable and unstable. When the value of (ε) gets bigger, the number of patterns could stay the same, increase, or decrease. In some cases, the number of patterns does not change even though the number of clusters decreases, because the density (number of points) of some clusters is less than other clusters. If we lose these clusters, they do not have a major impact on the number of discovered patterns because they do not appear often in the sequence database. However, in other cases even when the number of clusters decreases, the number of patterns increases because some clusters merge with other clusters, which are becoming denser. For instance, we have discovered 3 patterns when $\varepsilon = 0.03$ and 9 patterns when $\varepsilon = 0.05$ for the Deer dataset. The number of clusters was 216 for the first case and 49 for the second case. When ε is bigger the number of clusters is smaller and thus, the number of points in each cluster is larger. When ε is increased, the radius covers a larger area, so some previous clusters are merged with other clusters. This leads to the more denser clusters that apparently occur frequently because many points now belong to fewer clusters.

In the last case, the number of patterns goes down because the number of clusters decreases, so the length of a sequence in the database changes. For example, for the Deer dataset, when we set $\varepsilon = 0.07$, we have 16 clusters. Then, when we replace the points with its associated cluster ID, we have sequences of varied length starting from 1 itemset to 60 itemsets. In contrast, when we set $\varepsilon = 0.1$, we have 7 clusters. Then, when we replace the points with its associated cluster ID, we have sequences of varied length starting from 1 itemset to 40 itemsets. As long as we have long sequences in a database, we have a high chance to discover more patterns that reflect more details regarding the tracking of an object's movement. In the end, we conclude that the number of discovered patterns will behave differently when increasing the value of (ε).

*2) Impact of Miniumum Number of Points(MinPts)*

We conclude that when the MinPts increases, the execution time (ET) decreases, because the execution time of DBSCAN ($T_2$) also dominates (ET). For example, 80% of overall execution time (ET) is taken by ($T_2$) for the GeoLife dataset. The number of clusters can be increased or decreased depending on the status of a core point and a border point. Increasing the number of MinPts means that we require a denser neighborhood for a point, which produces fewer core points because it becomes harder to satisfy the MinPts conditions when the radius (ε) of a cluster ID is fixed. If the core points are still close enough within (ε) from a former cluster, the number of clusters does not change because DBSCAN put them in the same cluster. If the core points are close enough to some core points but no longer close to others, the number of clusters increases because each group of connected core points is assigned to a separate cluster. When a core point is not surrounded by a sufficient number of points which are equal to or more than MinPts, or a point is not a border, the number of clusters decreases because this point is labeled as a noise point. For a border point, if it is still not surrounded by the acceptable number of points, but in the vicinity of a core point, it will belong to the cluster that's associated with the core point. If the border point is within a neighborhood of two core points, it will belong to either one. Usually, border points are assigned to the cluster they are first discovered from. Therefore, each time DBSCAN is run, the

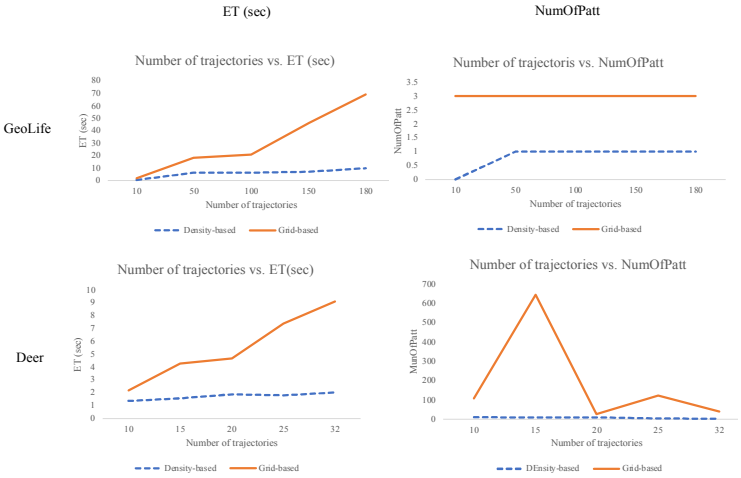



Fig.5. Impact of the number of trajectories

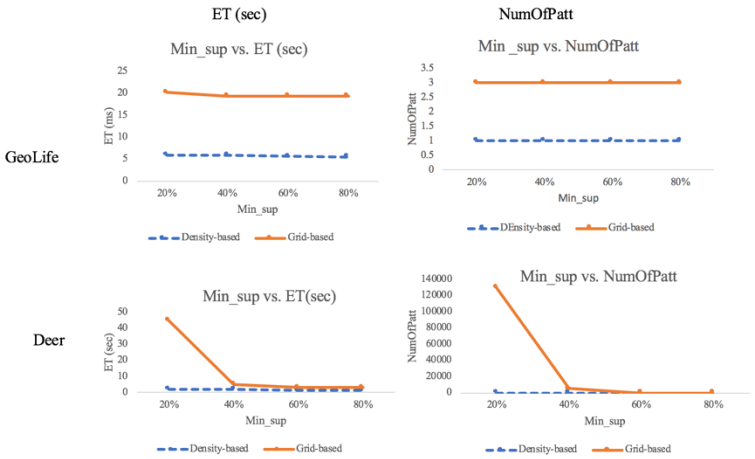



Fig.6. Impact of minimum support (Min-sup)

size of the cluster could change. However, the border point would not belong to any clusters because if its closest core point is not be classified as a core point any more or the border point cannot be reached from other core points, the point is labeled as noise point.

Based on that, the number of noise points increases, because if a point doesn't have a sufficient number of points within (ε), it will not be a core point. Thus, it is less likely to also find a border point that is in a neighborhood of a core point within that fixed cluster size. Due to the number of noise points increasing, the number of core and border points are less. Thus, the algorithm needs less time to determine the cluster ID for each point. The number of clusters and the performance of SPADE has a direct relationship. As we explained in Section IV.B.1, if the number of clusters increases, the execution time of SPADE also increases, and vice versa.

For the number of patterns that can be discovered by SPADE, it may not change, decrease, or increase. If the number of clusters decreases, but the existing ones still contain a huge number of points, that means these cluster IDs occur frequently in a sequence database, so SPADE always discovers trajectory patterns from these clusters. In other words, the density of a cluster (number of points in cluster) also affects the number of patterns. Some of the clusters always have thousands of points, for example, versus some small clusters that have hundreds of points. Disappearing of the small clusters does not impact the number of discovered patterns, because they do not appear periodically in the sequence database. However, since the number of clusters decreases, and the number of noise points increases, more points will be ignored. Thus, the possibility to find the most frequent sequences is decreasing. For the last case, when the number of patterns is increasing, the number of clusters decreases because other former clusters are merged with other clusters since they are not fulfilling the MinPts condition. Therefore, many clusters have higher number of points than before and their IDs will most likely arise in the database. In conclusion, we notice that increasing the number of minimum points has different impacts on the overall execution time (ET) and number of patterns (numOfPatt).

### 3) *Impact of Cell's Size*

When the size of the cell is increased, the total execution time (ET) decreases. After we looked closely, we inferred that the SPADE execution time ($T_4$) affects the total execution time (ET). When the cell size is increased, the number of cells decreases, and the execution time of SPADE ($T_4$) also decreases. The reason being that the length of sequences decreases because the number of points is replaced by the smaller number of cells. However, the execution time of cells partitioning ($T_2$) does have a huge impact on (ET), for example 90% of (ET) is taken by $T_2$ but the differences in term of times between different experiments are negligible when the cell size is increased. In other words, we can consider ($T_2$) as constant for all experiments because it depends on the number of the points, which is static. For this technique, there are no longer any noise points because each point must belong to a specific cell.

For the task of discovering sequential patterns, we observe that changing the cell size will produce a different number of patterns. The patterns could not change and SPADE would always discover the same number. Even though the grid-based considers the whole points, some cells may not contain any points. So, these cell IDs do not show up in the database at all. Increasing cell size would provide a bigger cell, but there still could not be any points. Therefore, the database will contain the only cell that has points and SPADE finds the same patterns every time. On the other hand, the number of patterns could increase or decrease. In some scenarios, the patterns increased because this technique suffers from the issue of adjacent cells that belong to different cells, which is mentioned in Section III.C. When we have many small adjacent cells that consist of points, some of the points belong to the $cell_i$ and other belong to $cell_{i+1}$ and at the end, all these points are supposed to belong to the same cell. In other scenarios, the number of patterns decreases because it also depends on the length of sequences in the database. As we discussed in Section IV.B.1, since the cluster or cell number decreases, the IDs become fewer. So, the length of sequences lessens, and SPADE cannot find sequential patterns longer than the longest sequence (in the worst case). Finally, increasing the size of a cell will reduce the overall execution time (ET) and produce an unpredictable number of patterns.

*4) Impact of the Number of Trajectories*

In this set of experiments, we compared the execution when the number of trajectories was increased. From Fig. 5, we can observe that overall execution time (ET) increases. This is acceptable because when we increase the number of trajectories, the number of the points also increases. Accordingly, the time of reading input ($T_1$), time for grouping points into a cluster or cell ($T_2$), time to convert sequence database ($T_3$) and time to apply SPADE increase too. Also, we notice that the time for grouping points into a cluster or cell ($T_2$) dominates the overall execution time (ET). For the GeoLife dataset, when the density-based is applied DBSCAN takes around 65% of ET, and for the grid-based cell, segmentation takes 85% of ET. For the Deer dataset, when the density-based is applied DBSCAN takes around 50% of ET, and for the grid-based cell, segmentation takes 60% of ET. Comparing the performance of the two techniques, the grid-based takes more time than density-based, which is expected since ($T_3$) dominates the overall execution time. Due to the increasing number of points, the execution time of grid-based is always higher. By using DBSCAN, some points are eliminated because they are noise points, while all the points are kept by grid-based.

From Fig. 5 the number of patterns shows unstable behavior because , as we discuss previously, it depends on the number of clusters/cells, and the number of points in each cluster/cell. If some former clusters/cells become denser when the number of trajectories is increased while others are not, the number of patterns remains the same. If increasing the number of trajectories creates more clusters or makes some previous cell denser, the number of patterns will increase. If the cluster/cell IDs decrease, or the length of sequences shortens, the number of patters decrease. The details of all cases are already discussed in Sections IV.B.1, IV.B.2, and IV.B.3.

*5) Impact of Minimum Support*

In this set of experiments, we compared (ET) and (NumOfPatt) for different values of minimum support threshold. From Fig. 6, we can see that when the minimum support is increased, the execution time of both techniques is decreased (note that the plot has straight lines because the difference of (ET) was about two to three seconds). The reason is that the SPADE algorithm generates less candidates when the min-sup is high. In other words, SPADE needs less time to eliminate infrequent sequences by scanning the ID-lists of candidates and testing each support count of a candidate against the min-sup. Since all other parameters (ε, MinPts, cell size, and the number of trajectories) are steady, the SPADE execution time ($T_4$) does not change that much and does not control the (ET). Again, the execution time of segmentation dominates the overall execution time (ET). Therefore, we can observe that the ET of grid-based is less than the ET of density-based for the GeoLife dataset, and vice versa for the Deer dataset. We have already discussed that in Section IV.B.4. The other observation was based on the number of frequent sequences that are generated by these segmentation techniques. As mentioned above, when the min-sup is increased, the number of patterns that are discovered by SPADE is going to be less because finding large patterns that satisfy the min-sup condition are difficult. For example, if a sequence database has 100 records, the number of patterns that exist at least in 30 records (min-sup =30%) is larger than the number of patterns that exist at least in 80 records (min-sup = 80%). As [26] stated, SPADE discovers all frequent sequential patterns that satisfy the condition of which their frequency is greater than or equal to min-sup, so every time we ran the algorithm, the grid-based technique provided either equal to or more patterns than the density-based technique. For example, when min-sup was set to 40% for the GeoLife dataset, the grid-based technique discovered three patterns versus the density-based technique that discovered one pattern. Therefore, the grid-based technique needs more time to mine the ID-lists of all candidates that are generated by SPADE. Thus, there is a necessary trade-off between getting more accurate patterns and minimizing SPADE 's execution time.

*6) Matching Sequential Patterns*

Since we use different techniques with different parameters and strategies to segment trajectories, we wanted to make sure that these two techniques discovered the same patterns. Therefore, after we assigned the cluster ID and cell ID to each point, we used the data structure, HashMap, and linked the list map to find the matching areas between them, as well as maintain the order of sequences based on time stamps. Then, we found the matching points between each cluster and cell. For example, in one experiment for GeoLife when the cell size was larger than cluster size, we had 2 clusters and 20 cells. Then, we found how many matching points were between each cluster and cell. In the case that a cell matches a cluster with zero points, it means that they do not overlap. After that, for each cell that had some matching points with a cluster, we changed the cell ID to be similar to the cluster ID. Next, we compared each sequential pattern of cells with the sequential patterns of clusters, and made sure that all discovered patterns by the grid-based were matching all the discovered patterns by the density-based. In contrast, if the cluster radius was larger than the cell size, we found the matching points between the clusters and cells. Then, we changed the cluster ID to be similar to the cell ID and compared the discovered sequential patterns. We found that all the sequential patterns that were discovered by the density-based method were also discovered by the grid-based method, but not conversely. The patterns that were discovered by the grid-based were not discovered by the density-based method. From this result, we can conclude that the grid-based method helps discover more hidden sequential patterns than the density-based method, and as it was mentioned, SPADE discovers all frequent sequential patterns.

## V. CONCLUSIOIN AND FUTURE WORK

In this paper, we presented our efforts in studying the impacts of the two trajectory segmentation techniques, the density-based clustering technique, and the grid-based technique, on the discovery of the sequential patterns from the trajectories. Our study showed that the density-based technique outperforms the grid-based technique in terms of execution

time (ET). In contrast, the grid-based outperforms the density-based in terms of the number of discovered patterns. In addition, the study showed how the parameters impact the overall achievement of these two techniques. Therefore, choosing the optimal values is an important step to guarantee efficient and effective results. Our work is subject to further improvement in implementing a parallel version of the trajectory segmentation techniques and the SPADE algorithm in order to be able to mine frequent sequential patterns from big trajectory databases efficiently. Also, we plan to extend our work to study the impacts of trajectory segmentation on other data mining tasks, such as trajectory outlier detection.

## ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under Grant No. 1302439 and 1302423.

## REFERENCES

[1] Agrawal, Rakesh, and Ramakrishnan Srikant. "Mining sequential patterns." In *Data Engineerin. Proceedings of the Eleventh IEEE International Conference*, pp. 3-14,1995.

[2] Anastasiu, David C., Jeremy Iverson, Shaden Smith, and George Karypis. "Big data frequent pattern mining." In *Frequent Pattern Mining*, pp. 225-259. Springer, Cham, 2014.

[3] Buchta, Christian, Michael Hahsler. "Package 'arulesSequences.". 2018.

[4] Cao, Huiping, Nikos Mamoulis, and David W. Cheung. "Mining frequent spatio-temporal sequential patterns." In *Data Mining, Fifth IEEE International Conference*, pp. 8-pp, 2005.

[5] Cao, Hu, Ouri Wolfson, and Goce Trajcevski. "Spatio-temporal data reduction with deterministic error bounds." *The VLDB Journal—The International Journal on Very Large Data Bases*15, no. 3, 2006.

[6] Chen, Chien-Cheng, and Meng-Fen Chiang. "Trajectory pattern mining: exploring semantic and time information." In *IEEE Technologies and Applications of Artificial Intelligence (TAAI), 2016 Conference on*, pp. 130-137, 2016.

[7] Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise." In *Kdd*, vol. 96, no. 34, pp. 226-231, 1996.

[8] Giannotti, Fosca, Mirco Nanni, and Dino Pedreschi. "Efficient mining of temporally annotated sequences." In *Proceedings of thr SIAM International Conference on Data Mining*, pp. 348-359. Society for Industrial and Applied Mathematics, 2006.

[9] Giannotti, Fosca, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. "Trajectory pattern mining." In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 330-339, 2007.

[10] Han, J., Kamber, M. *Data Mining: Concepts and Technique.* San Francisco, CA :Elsevier Inc.2006.

[11] Han, Jiawei, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. "FreeSpan: frequent pattern-projected sequential pattern mining." In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 355-359, 2000.

[12] Ho, Shen-Shyang, and Shuhua Ruan. "Preserving privacy for interesting location pattern mining from trajectory data.".2013.

[13] Lee, Jae-Gil, Jiawei Han, and Kyu-Young Whang. "Trajectory clustering: a partition-and-group framework." In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 593-604 2007.

[14] Li, Huanhuan, Xiaofeng Zhou, and Chaojun Pan. "Study on GSP algorithm based on Hadoop." In *Electronics Information and Emergency Communication (ICEIEC),5th IEEE International Conference*, pp. 321-324, 2015.

[15] Liang, Yen-hui, and Shiow-yang Wu. "Sequence-growth: A scalable and effective frequent itemset mining algorithm for big data based on MapReduce framework." In *Big Data (BigData Congress), IEEE International Congress*, pp. 393-400, 2015.

[16] Mabroukeh, Nizar R., and Christie I. Ezeife. "A taxonomy of sequential pattern mining algorithms." *ACM Computing Surveys (CSUR)* 43, no. 1 2010.

[17] Mamoulis, Nikos, Huiping Cao, George Kollios, Marios Hadjieleftheriou, Yufei Tao, and David W. Cheung. "Mining, indexing, and querying historical spatiotemporal data." In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 236-245, 2004.

[18] Masciari, Elio, Gao Shi, and Carlo Zaniolo. "Sequential pattern mining from trajectory data." In *Proceedings of the 17th ACM International Database Engineering & Applications Symposium*, pp. 162-167, 2013.

[19] Pei, Jian, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth." In *icccn*, p. 0215, 2001.

[20] Qiao, Shaojie, Tianrui Li, Jing Peng, and Jiangtao Qiu. "Parallel sequential pattern mining of massive trajectory data." *International Journal of Computational Intelligence Systems* 3, no. 3, 2010.

[21] Qiao, Shaojie, Changjie Tang, Shucheng Dai, Mingfang Zhu, Jing Peng, Hongjun Li, and Yungchang Ku. "Partspan: Parallel sequence mining of trajectory patterns." In *Fuzzy Systems and Knowledge Discovery FSKD'08. Fifth International Conference on*, vol. 5, pp. 363-367, 2008.

[22] Schubert, Erich, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. "DBSCAN revisited, revisited: why and how you should (still) use DBSCAN." *ACM Transactions on Database Systems (TODS)* 42, no. 3, 2017.

[23] Srikant, Ramakrishnan, and Rakesh Agrawal. "Mining sequential patterns: Generalizations and performance improvements." In *International Conference on Extending Database Technology*, pp. 1-17. Springer, Berlin, Heidelberg, 1996.

[24] Tsoukatos, Ilias, and Dimitrios Gunopulos. "Efficient mining of spatiotemporal patterns." In *International Symposium on Spatial and Temporal Databases*, pp. 425-442. Springer, Berlin, Heidelberg, 2001.

[25] Wei, Yong-qing, Dong Liu, and Lin-shan Duan. "Distributed PrefixSpan algorithm based on MapReduce." In *IEEE Information Technology in Medicine and Education (ITME),International Symposium on*, vol. 2, pp. 901-904, 2012.

[26] Zaki, Mohammed J. "SPADE: An efficient algorithm for mining frequent sequences." *Machine learning* 42, no. 1-2, 2001.

[27] Zheng, Yu. "Trajectory data mining: an overview." *ACM Transactions on Intelligent Systems and Technology (TIST)* 6, no. 3, 2015.

[28] Zheng, Yu, and Xing Xie. "Learning travel recommendations from user-generated GPS traces." *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, no. 1, 2011.

[29] Zheng, Yu, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. "Understanding mobility based on GPS data." In *Proceedings of the 10th international conference on Ubiquitous computing*, pp. 312-321, 2008.

[30] Zheng, Yu, Xing Xie, and Wei-Ying Ma. "Geolife: A collaborative social networking service among user, location and trajectory." *IEEE Data Eng. Bull.* 33, no. 2, 2010.

[31] Zheng, Yu, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. "Mining interesting locations and travel sequences from GPS trajectories." In *Proceedings of the 18th ACM international conference on World wide web*, pp. 791-800, 2009.

[32] Zhu, Wen-Yuan, Wen-Chih Peng, Chih-Chieh Hung, Po-Ruey Lei, and Ling-Jyh Chen. "Exploring sequential probability tree for movement-based community discovery." *IEEE Transactions on Knowledge and Data Engineering* 26, no. 11, 2014.