Trajectory Clustering: A Partition-and-Group Framework*

Jae-Gil Lee, Jiawei Han
Department of Computer Science
University of Illinois at Urbana-Champaign
jaegil@uiuc.edu, hanj@cs.uiuc.edu

Kyu-Young Whang
Department of Computer Science / AITrc
KAIST
kywhang@cs.kaist.ac.kr

ABSTRACT

Existing trajectory clustering algorithms group similar trajectories as a whole, thus discovering common trajectories. Our key observation is that clustering trajectories as a whole could miss common *sub*-trajectories. Discovering common sub-trajectories is very useful in many applications, especially if we have regions of special interest for analysis. In this paper, we propose a new *partition-and-group* framework for clustering trajectories, which partitions a trajectory into a set of line segments, and then, groups similar line segments together into a cluster. The primary advantage of this framework is to discover common *sub*-trajectories from a trajectory database. Based on this partition-and-group framework, we develop a trajectory clustering algorithm *TRACLUS*. Our algorithm consists of two phases: *partitioning* and *grouping*. For the first phase, we present a formal trajectory partitioning algorithm using the minimum description length (MDL) principle. For the second phase, we present a density-based line-segment clustering algorithm. Experimental results demonstrate that TRACLUS correctly discovers common sub-trajectories from real trajectory data.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications – *Data Mining*

General Terms: Algorithms

Keywords: Partition-and-group framework, trajectory clustering, MDL principle, density-based clustering

1. INTRODUCTION

Clustering is the process of grouping a set of physical or abstract objects into classes of *similar* objects [11]. Clustering has been widely used in numerous applications such as market research, pattern recognition, data analysis, and image processing. A number of clustering algorithms have been reported in the literature. Representative algorithms include k-means [16], BIRCH [23], DBSCAN [6], OPTICS [2], and STING [21]. Previous research has mainly dealt with clustering of *point* data.

Recent improvements in satellites and tracking facilities have made it possible to collect a large amount of *trajectory* data of moving objects. Examples include vehicle position data, hurricane track data, and animal movement data. There is increasing interest to perform data analysis over these trajectory data. A typical data analysis task is to find objects that have moved in a similar way [20]. Thus, an efficient clustering algorithm for trajectories is essential for such data analysis tasks.

Gaffney *et al.* [7, 8] have proposed a model-based clustering algorithm for trajectories. In this algorithm, a set of trajectories is represented using a regression mixture model. Then, unsupervised learning is carried out using the maximum likelihood principle. Specifically, the EM algorithm is used to determine the cluster memberships. This algorithm clusters trajectories *as a whole*; *i.e.*, the basic unit of clustering is the whole trajectory.

Our key observation is that clustering trajectories as a whole could not detect *similar portions* of the trajectories. We note that a trajectory may have a long and complicated path. Hence, even though some portions of trajectories show a common behavior, the whole trajectories might not.

Example 1. Consider the five trajectories in Figure 1. We can clearly see that there is a common behavior, denoted by the thick arrow, in the dotted rectangle. However, if we cluster these trajectories as a whole, we cannot discover the common behavior since they move to totally different directions; thus, we miss this valuable information. □

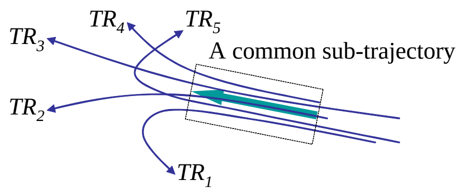


Figure 1: An example of a common sub-trajectory.

Our solution is to partition a trajectory into a set of line segments and then group similar line segments. This framework is called a *partition-and-group* framework. The primary advantage of the partition-and-group framework is the discovery of common *sub*-trajectories from a trajectory

*The work was supported in part by the Korea Research Foundation grant, funded by the Korean Government (MOEHRD), KRF-2006-214-D00129 and the U.S. National Science Foundation grants IIS-05-13678/06-42771 and BDI-05-15813. Any opinions, findings, and conclusions expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

database. This is exactly the reason why we partition a trajectory into a set of line segments.

We contend that discovering the common sub-trajectories is very useful, especially if we have regions of special interest for analysis. In this case, we want to concentrate on the common behaviors within those regions. There are many examples in real applications. We present two application scenarios.

1. *Hurricanes: Landfall Forecasts [17]*

   Meteorologists are trying to improve the ability to forecast the location and time of hurricane landfall. An accurate landfall location forecast is of prime importance since it is crucial for reducing hurricane damages. Meteorologists will be interested in the common behaviors of hurricanes *near the coastline* (*i.e.*, at the time of landing) or *at sea* (*i.e.*, before landing). Thus, discovering the common sub-trajectories helps improve the accuracy of hurricane landfall forecasts.

2. *Animal Movements: Effects of Roads and Traffic [22]*

   Zoologists are investigating the impacts of the varying levels of vehicular traffic on the movement, distribution, and habitat use of animals. They mainly measure the mean distance between the road and animals. Zoologists will be interested in the common behaviors of animals *near the road* where the traffic rate has been varied. Hence, discovering the common sub-trajectories helps reveal the effects of roads and traffic.

Example 2. Consider an animal habitat and roads in Figure 2. Thick lines represent roads, and they have different traffic rates. Wisdom *et al.* [22] explore the spatial patterns of mule deer and elk in relation to roads of varying traffic rates. More specifically, one of the objectives is to assess the degree to which mule deer and elk may avoid *areas near roads* based on variation in rates of motorized traffic. These areas are represented as solid rectangles. Thus, our framework is indeed useful for this kind of research. □

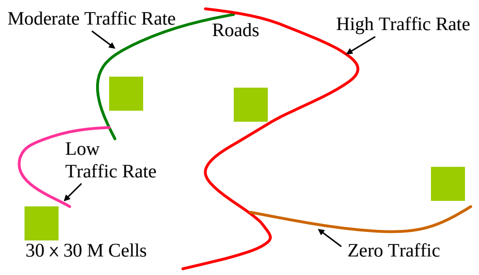


Figure 2: Monitoring animal movements [22][1].

One might argue that, if we prune the useless parts of trajectories and keep only the interesting ones, we can use traditional clustering algorithms that cluster trajectories as a whole. This alternative, however, has two major drawbacks compared with our partition-and-group framework. First, it is tricky to determine which part of the trajectories is useless. Second, pruning "useless" parts of trajectories forbids us to discover unexpected clustering results.

In this paper, we propose a *partition-and-group* framework for clustering trajectories. As indicated by its name, trajectory clustering based on this framework consists of the following two phases:

(1) The *partitioning* phase: Each trajectory is optimally partitioned into a set of line segments. These line segments are provided to the next phase.

(2) The *grouping* phase: Similar line segments are grouped into a cluster. Here, a density-based clustering method is exploited.

Density-based clustering methods [2, 6] are the most suitable for line segments because they can *discover clusters of arbitrary shape* and can *filter out noises* [11]. We can easily see that line segment clusters are usually of arbitrary shape, and a trajectory database contains typically a large amount of noise (*i.e.*, outliers).

In summary, the contributions of this paper are as follows:

- We propose a *partition-and-group* framework for clustering trajectories. This framework enables us to discover common *sub*-trajectories, whereas previous frameworks do not.
- We present a formal trajectory partitioning algorithm using the minimum description length (MDL) [9] principle.
- We present an efficient density-based clustering algorithm for line segments. We carefully design a distance function to define the density of line segments. In addition, we provide a simple but effective heuristic for determining parameter values of the algorithm.
- We demonstrate, by using various real data sets, that our clustering algorithm effectively discovers the representative trajectories from a trajectory database.

The rest of the paper is organized as follows. Section 2 presents an overview of our trajectory clustering algorithm. Section 3 proposes a trajectory partitioning algorithm for the first phase. Section 4 proposes a line segment clustering algorithm for the second phase. Section 5 presents the results of experimental evaluation. Section 6 discusses related work. Finally, Section 7 concludes the paper.

2. TRAJECTORY CLUSTERING

In this section, we present an overview of our design. Section 2.1 formally presents the problem statement. Section 2.2 presents the skeleton of our trajectory clustering algorithm, which we call *TRACLUS*. Section 2.3 defines our distance function for line segments.

2.1 Problem Statement

We develop an efficient clustering algorithm based on the partition-and-group framework. Given a set of *trajectories* $\mathcal{I} = \{TR_1, \cdots, TR_{num_{tra}}\}$, our algorithm generates a set of *clusters* $\mathcal{O} = \{C_1, \cdots, C_{num_{clus}}\}$ as well as a *representative trajectory* for each cluster C_i, where the trajectory, cluster, and representative trajectory are defined as follows.

A *trajectory* is a sequence of multi-dimensional points. It is denoted as $TR_i = p_1p_2p_3 \cdots p_j \cdots p_{len_i}$ $(1 \leq i \leq num_{tra})$. Here, p_j $(1 \leq j \leq len_i)$ is a d-dimensional point. The length len_i of a trajectory can be different from those of other trajectories. A trajectory $p_{c_1}p_{c_2} \cdots p_{c_k}$ $(1 \leq c_1 < c_2 < \cdots < c_k \leq len_i)$ is called a *sub-trajectory* of TR_i.

[1]The figure is borrowed from the authors' slide.

A *cluster* is a set of trajectory partitions. A *trajectory partition* is a line segment $p_i p_j$ $(i < j)$, where p_i and p_j are the points chosen from the same trajectory. Line segments that belong to the same cluster are close to each other according to the distance measure. Notice that a trajectory can belong to multiple clusters since a trajectory is partitioned into multiple line segments, and clustering is performed over these line segments.

A *representative trajectory* is a sequence of points just like an ordinary trajectory. It is an imaginary trajectory that indicates the major behavior of the trajectory partitions (*i.e.*, line segments) that belong to the cluster. Notice that a representative trajectory indicates a common sub-trajectory.

Example 3. Figure 3 shows the overall procedure of trajectory clustering in the partition-and-group framework. First, each trajectory is partitioned into a set of line segments. Second, line segments which are close to each other according to our distance measure are grouped together into a cluster. Then, a representative trajectory is generated for each cluster. □

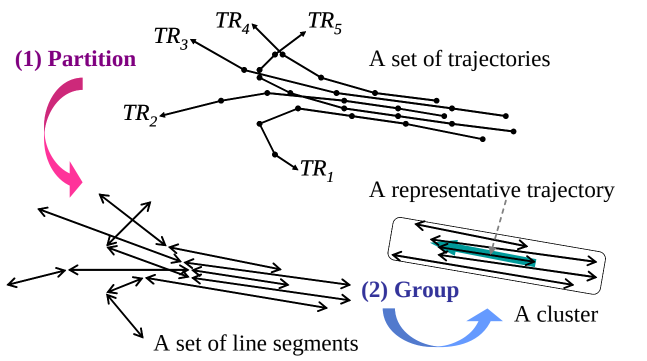


Figure 3: An example of trajectory clustering in the partition-and-group framework.

2.2 The *TRACLUS* Algorithm

Figure 4 shows the skeleton of our trajectory clustering algorithm *TRACLUS*. As illustrated in Figure 3, it goes through the two phases. It executes three algorithms to perform the subtasks (lines 2, 4, and 6). We explain these algorithms in Sections 3.3, 4.2, and 4.3.

2.3 Distance Function

We now define the distance function used in clustering line segments, which is composed of three components: (i) the *perpendicular distance* $(d_\perp)$, (ii) the *parallel distance* $(d_\parallel)$, and (iii) the *angle distance* (d_θ). These components are adapted from similarity measures used in the area of pattern recognition [4]. They are intuitively illustrated in Figure 5.

We formally define the three components through Definitions 1∼3. Suppose there are two d-dimensional line segments $L_i = s_i e_i$ and $L_j = s_j e_j$. Here, s_i, e_i, s_j, and e_j represent d-dimensional points. We assign a longer line segment to L_i and a shorter one to L_j without losing generality.

Definition 1. The *perpendicular distance* between L_i and L_j is defined as Formula (1), which is the Lehmer mean [2] of order 2. Suppose the projection points of the points s_j and e_j onto L_i are p_s and p_e, respectively. $l_{\perp 1}$ is the Euclidean distance between s_j and p_s; $l_{\perp 2}$ is that between e_j and p_e.

$$d_\perp(L_i, L_j) = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}} \quad (1)$$

Definition 2. The *parallel distance* between L_i and L_j is defined as Formula (2). Suppose the projection points of the points s_j and e_j onto L_i are p_s and p_e, respectively. $l_{\parallel 1}$ is the minimum of the Euclidean distances of p_s to s_i and e_i. Likewise, $l_{\parallel 2}$ is the minimum of the Euclidean distances of p_e to s_i and e_i.

$$d_\parallel(L_i, L_j) = \text{MIN}(l_{\parallel 1}, l_{\parallel 2}) \quad (2)$$

REMARK: The parallel distance in Definition 2 is designed to be robust to detection errors, especially *broken line segments*. If we use MAX$(l_{\parallel 1}, l_{\parallel 2})$ instead of MIN$(l_{\parallel 1}, l_{\parallel 2})$, the parallel distance could be significantly perturbed by broken line segments. For more information, refer to studies on the distance measure [4] in the domain of pattern recognition.

Definition 3. The *angle distance* between L_i and L_j is defined as Formula (3). Here, $\|L_j\|$ is the length of L_j, and θ $(0° \le \theta \le 180°)$ is the smaller intersecting angle between L_i and L_j.

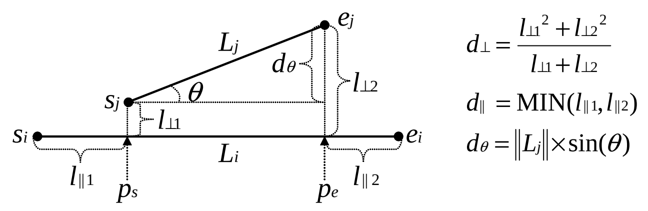
$$d_\theta(L_i, L_j) = \begin{cases} \|L_j\| \times sin(\theta), & \text{if } 0° \le \theta < 90° \\ \|L_j\|, & \text{if } 90° \le \theta \le 180° \end{cases} \quad (3)$$

[2] The Lehmer mean of a set of n numbers $(a_k)_{k=1}^n$ is defined by $L_p(a_1, a_2, \cdots, a_n) = \frac{\sum_{k=1}^n a_k^p}{\sum_{k=1}^n a_k^{p-1}}$.

Algorithm **TRACLUS** (TRAjectory CLUStering)

```
INPUT: A set of trajectories $\mathcal{I} = \{TR_1, \cdots, TR_{num_{tra}}\}$
OUTPUT: (1) A set of clusters $\mathcal{O} = \{C_1, \cdots, C_{num_{clus}}\}$
        (2) A set of representative trajectories
ALGORITHM:
    /* PARTITIONING PHASE */
01: for each ($TR \in \mathcal{I}$) do
        /* Figure 8 */
02:     Execute Approximate Trajectory Partitioning;
        Get a set $\mathcal{L}$ of line segments using the result;
03:     Accumulate $\mathcal{L}$ into a set $\mathcal{D}$;
    /* GROUPING PHASE */
    /* Figure 12 */
04: Execute Line Segment Clustering for $\mathcal{D}$;
    Get a set $\mathcal{O}$ of clusters as the result;
05: for each ($C \in \mathcal{O}$) do
        /* Figure 15 */
06:     Execute Representative Trajectory Generation;
        Get a representative trajectory as the result;
```

Figure 4: The algorithm *TRACLUS*.



e_j
L_j
d_θ
θ
$l_{\perp 2}$
s_j
$l_{\perp 1}$
s_i
e_i
L_i
$l_{\parallel 1}$
p_s
p_e
$l_{\parallel 2}$
$d_\perp = \frac{l_{\perp 1}{}^2 + l_{\perp 2}{}^2}{l_{\perp 1} + l_{\perp 2}}$
$d_\parallel = \text{MIN}(l_{\parallel 1}, l_{\parallel 2})$
$d_\theta = \|L_j\| \times \sin(\theta)$

Figure 5: Three components of the distance function for line segments.

REMARK: The angle distance in Definition 3 is designed for *trajectories with directions.* The entire length contributes to the angle distance when the directions differ significantly. If we handle trajectories without directions, the angle distance is defined as simply $\|L_j\| \times sin(\theta)$.

The three components can be easily calculated using vector operations. Let $\overrightarrow{ab}$ denote a vector constructed by two points a and b. The projection points p_s and p_e in Definitions 1 and 2 are calculated using Formula (4). In addition, the angle θ in Definition 3 is calculated using Formula (5).

$$p_s = s_i + u_1 \cdot \overrightarrow{s_i e_i}, \quad p_e = s_i + u_2 \cdot \overrightarrow{s_i e_i},$$
$$\text{where} \quad u_1 = \frac{\overrightarrow{s_i s_j} \cdot \overrightarrow{s_i e_i}}{\|\overrightarrow{s_i e_i}\|^2}, \quad u_2 = \frac{\overrightarrow{s_i e_j} \cdot \overrightarrow{s_i e_i}}{\|\overrightarrow{s_i e_i}\|^2} \tag{4}$$

$$cos(\theta) = \frac{\overrightarrow{s_i e_i} \cdot \overrightarrow{s_j e_j}}{\|\overrightarrow{s_i e_i}\|\|\overrightarrow{s_j e_j}\|} \tag{5}$$

We finally define the distance between two line segments as follows: $dist(L_i, L_j) = w_\perp \cdot d_\perp(L_i, L_j) + w_\parallel \cdot d_\parallel(L_i, L_j) + w_\theta \cdot d_\theta(L_i, L_j)$. We explain the advantage of this distance function over the sum of the distances of endpoints in Appendix A. Besides, we discuss the need for assigning different weights in Appendix B.

3. TRAJECTORY PARTITIONING

In this section, we propose a trajectory partitioning algorithm for the partitioning phase. We first discuss two desirable properties of trajectory partitioning in Section 3.1. We then describe a formal method for achieving these properties in Section 3.2. Our method transforms trajectory partitioning to MDL optimization. Since the cost of finding the optimal solution is too high, we present an $O(n)$ approximate algorithm in Section 3.3.

3.1 Desirable Properties

We aim at finding the points where the behavior of a trajectory changes rapidly, which we call *characteristic points.* From a trajectory $TR_i = p_1 p_2 p_3 \cdots p_j \cdots p_{len_i}$, we determine a set of characteristic points $\{p_{c_1}, p_{c_2}, p_{c_3}, \cdots, p_{c_{par_i}}\}$ $(c_1 < c_2 < c_3 < \cdots < c_{par_i})$. Then, the trajectory TR_i is partitioned at every characteristic point, and each partition is represented by a line segment between two consecutive characteristic points. That is, TR_i is partitioned into a set of $(par_i - 1)$ line segments $\{p_{c_1}p_{c_2}, p_{c_2}p_{c_3}, \cdots, p_{c_{par_i-1}}p_{c_{par_i}}\}$. We call such a line segment a *trajectory partition.* Figure 6 shows an example of a trajectory and its trajectory partitions.

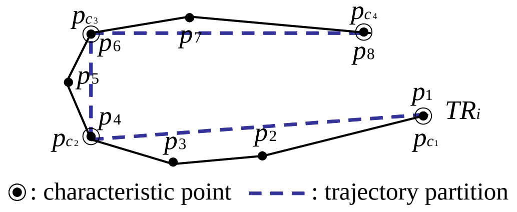


Figure 6: An example of a trajectory and its trajectory partitions.

The optimal partitioning of a trajectory should possess two desirable properties: *preciseness* and *conciseness.* Preciseness means that the difference between a trajectory and a set of its trajectory partitions should be as small as possible. Conciseness means that the number of trajectory partitions should be as small as possible.

We point out that achieving preciseness makes a trajectory partitioned everywhere its behavior changes rapidly. Otherwise, preciseness cannot be achieved. If p_{c_2} were not chosen in Figure 6, preciseness would decrease due to a large difference between TR_i and $p_{c_1}p_{c_3}$.

Preciseness and conciseness are contradictory to each other. For example, if all the points in a trajectory are chosen as characteristic points (*i.e.*, $par_i = len_i$), preciseness is maximized, but conciseness is minimized. In contrast, if only the starting and ending points of a trajectory are chosen as characteristic points (*i.e.*, $par_i = 2$), conciseness is maximized, but preciseness might be minimized. Therefore, we need to find an optimal tradeoff between the two properties.

3.2 Formalization Using the MDL Principle

We propose a method of finding the optimal tradeoff between preciseness and conciseness. We adopt the minimum description length (MDL) principle widely used in information theory.

The MDL cost consists of two components [9]: $L(H)$ and $L(D|H)$. Here, H means the hypothesis, and D the data. The two components are informally stated as follows [9]: "$L(H)$ is the length, in bits, of the description of the hypothesis; and $L(D|H)$ is the length, in bits, of the description of the data when encoded with the help of the hypothesis." The best hypothesis H to explain D is the one that minimizes the sum of $L(H)$ and $L(D|H)$.

In our trajectory partitioning problem, a hypothesis corresponds to a specific set of trajectory partitions. This formulation is quite natural because we want to find the optimal partitioning of a trajectory. As a result, finding the optimal partitioning translates to finding the best hypothesis using the MDL principle.

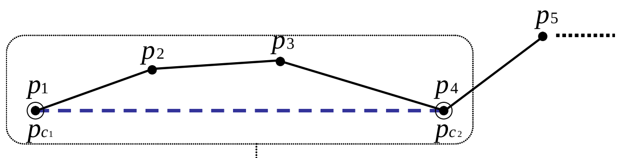
Figure 7 shows our formulation of $L(H)$ and $L(D|H)$. Suppose a trajectory $TR_i = p_1 p_2 p_3 \cdots p_j \cdots p_{len_i}$ and a set of characteristic points $= \{p_{c_1}, p_{c_2}, p_{c_3}, \cdots, p_{c_{par_i}}\}$. Then, we formulate $L(H)$ by Formula (6). Here, $len(p_{c_j}p_{c_{j+1}})$ denotes the length of a line segment $p_{c_j}p_{c_{j+1}}$, *i.e.*, the Euclidean distance between p_{c_j} and $p_{c_{j+1}}$. Hence, $L(H)$ represents the sum of the length of all trajectory partitions.

On the other hand, we formulate $L(D|H)$ by Formula (7). $L(D|H)$ represents the sum of the difference between a trajectory and a set of its trajectory partitions. For each trajectory partition $p_{c_j}p_{c_{j+1}}$, we add up the difference between this trajectory partition and a line segment $p_k p_{k+1}$ $(c_j \le k \le c_{j+1} - 1)$ that belongs to the partition. To measure the difference, we use the sum of the perpendicular distance and the angle distance. We do not consider the parallel distance since a trajectory encloses its trajectory partitions.

$$L(H) = \sum_{j=1}^{par_i - 1} \log_2(len(p_{c_j}p_{c_{j+1}})) \tag{6}$$

$$L(D|H) = \sum_{j=1}^{par_i - 1} \sum_{k=c_j}^{c_{j+1}-1} \{ \log_2(d_\perp(p_{c_j}p_{c_{j+1}}, p_k p_{k+1})) + \log_2(d_\theta(p_{c_j}p_{c_{j+1}}, p_k p_{k+1}))\} \tag{7}$$

The length and distances are real numbers. In practice, we encode the real number x assuming a precision δ, so that the encoded number x_δ satisfies $|x - x_\delta| < \delta$. If x is large and $x \approx x_\delta$, $L(x) = \log_2 x - \log_2 \delta$ [15]. Here, we set δ to be 1. Thus, $L(x)$ is simply equal to $\log_2 x$.



$$L(H) = \log_2(len(p_1p_4))$$
$$L(D|H) = \log_2(d_\perp(p_1p_4, p_1p_2) + d_\perp(p_1p_4, p_2p_3) + d_\perp(p_1p_4, p_3p_4)) + \log_2(d_\theta(p_1p_4, p_1p_2) + d_\theta(p_1p_4, p_2p_3) + d_\theta(p_1p_4, p_3p_4))$$

Figure 7: Formulation of the MDL cost.

We define $L(H)$ using the length of a line segment instead of the endpoints of a line segment. The reason is two-fold. First, our task is to cluster line segments (sub-trajectories) according to their relative distances. The length (reflected in $L(H)$) and the distance function, *i.e.*, the perpendicular, parallel, and angle distances between line segments (reflected in $L(D|H)$), measure the relative distance better than the endpoints of line segments. That is, the new definition fits better the task of sub-trajectory clustering. Second, a very important reason not to use endpoints is to make the clustering results not influenced by the coordinate values of line segments. That is, a bunch of line segments can be shifted from a low coordinate location to a high coordinate location; however, our distance function should still correctly measure the relative distance. If we formulate $L(H)$ using the coordinate values of two endpoints, the clustering results could be distorted by such shifting. Please see Appendix C for an example.

We note that $L(H)$ measures the degree of conciseness, and $L(D|H)$ that of preciseness. Due to the triangle inequality, $L(H)$ increases as the number of trajectory partitions does. Besides, it is obvious that $L(D|H)$ increases as a set of trajectory partitions deviates from the trajectory.

As mentioned before, we need to find the optimal partitioning that minimizes $L(H) + L(D|H)$. This is exactly the tradeoff between preciseness and conciseness. The cost of finding the optimal partitioning is prohibitive since we need to consider every subset of the points in a trajectory.

3.3 Approximate Solution

The key idea of our approximation is to regard the set of local optima as the global optimum. Let $MDL_{par}(p_i, p_j)$ denote the MDL cost ($= L(H) + L(D|H)$) of a trajectory between p_i and p_j ($i < j$) when assuming that p_i and p_j are only characteristic points. Let $MDL_{nopar}(p_i, p_j)$ denote the MDL cost when assuming that there is no characteristic point between p_i and p_j, *i.e.*, when preserving the original trajectory. We note that $L(D|H)$ in $MDL_{nopar}(p_i, p_j)$ is zero. Then, a local optimum is the *longest* trajectory partition p_ip_j that satisfies $MDL_{par}(p_i, p_k) \le MDL_{nopar}(p_i, p_k)$ for every k such that $i < k \le j$. If the former is smaller than the latter, we know that choosing p_k as a characteristic point makes the MDL cost smaller than not choosing it. Furthermore, we increase the length of this trajectory partition to the extent possible for the sake of conciseness.

Figure 8 shows the algorithm *Approximate Trajectory Partitioning*. We compute MDL_{par} and MDL_{nopar} for each point in a trajectory (lines 5~6). If MDL_{par} is greater than MDL_{nopar}, we insert the immediately previous point $p_{currIndex-1}$ into the set CP_i of characteristic points (line 8). Then, we repeat the same procedure from that point (line 9). Otherwise, we increase the length of a candidate trajectory partition (line 11).

Algorithm **Approximate Trajectory Partitioning**

```
INPUT: A trajectory $TR_i = p_1p_2p_3 \cdots p_j \cdots p_{len_i}$
OUTPUT: A set $CP_i$ of characteristic points
ALGORITHM:
01: Add $p_1$ into the set $CP_i$;  /* the starting point */
02: $startIndex := 1$, $length := 1$;
03: while ($startIndex + length \le len_i$) do
04:     $currIndex := startIndex + length$;
05:     $cost_{par} := MDL_{par}(p_{startIndex}, p_{currIndex})$;
06:     $cost_{nopar} := MDL_{nopar}(p_{startIndex}, p_{currIndex})$;
        /* check if partitioning at the current point makes
           the MDL cost larger than not partitioning */
07:     if ($cost_{par} > cost_{nopar}$) then
            /* partition at the previous point */
08:         Add $p_{currIndex-1}$ into the set $CP_i$;
09:         $startIndex := currIndex - 1$, $length := 1$;
10:     else
11:         $length := length + 1$;
12: Add $p_{len_i}$ into the set $CP_i$;  /* the ending point */
```

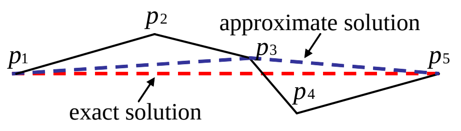
Figure 8: An approximate algorithm for partitioning a trajectory.

We now present, in Lemma 1, the time complexity of the approximate algorithm.

Lemma 1. The time complexity of the algorithm in Figure 8 is $O(n)$, where n is the length (*i.e.*, the number of points) of a trajectory TR_i.

PROOF: The number of MDL computations required is equal to the number of line segments (*i.e.*, $n - 1$) in the trajectory TR_i. □

Of course, this algorithm may fail to find the optimal partitioning. Let us present a simple example in Figure 9. Suppose that the optimal partitioning with the minimal MDL cost is $\{p_1p_5\}$. This algorithm cannot find the exact solution since it stops scanning at p_4 where MDL_{par} is larger than MDL_{nopar}. Nevertheless, the precision of this algorithm is quite high. Our experience indicates that the precision is about 80% on average, which means that 80% of the approximate solutions appear also in the exact solutions.



$$MDL_{par}(p_1, p_4) > MDL_{nopar}(p_1, p_4)$$
$$MDL_{par}(p_1, p_5) < MDL_{nopar}(p_1, p_5)$$

Figure 9: An example where the approximate algorithm fails to find the optimal partitioning.

4. LINE SEGMENT CLUSTERING

In this section, we propose a line segment clustering algorithm for the grouping phase. We first discuss the density of line segments in Section 4.1. We then present our density-based clustering algorithm, which is based on the algorithm DBSCAN [6], in Section 4.2. We discuss the reason for choosing DBSCAN in Appendix D. Next, we present a method of computing a representative trajectory in Section 4.3. Finally, since density-based clustering algorithms intrinsically are sensitive to their parameter values [11], we present a heuristic for determining parameter values in Section 4.4.

4.1 Density of Line Segments

4.1.1 Review of Our Distance Function

Our distance function is the weighted sum of three kinds of distances. First, the perpendicular distance measures the positional difference mainly between line segments extracted from different trajectories. Second, the parallel distance measures the positional difference mainly between line segments extracted from the same trajectory. We note that the parallel distance between two adjacent line segments in a trajectory is always zero. Third, the angle distance measures the directional difference between line segments.

Before proceeding, we address the symmetry of our distance function in Lemma 2. Symmetry is important to avoid ambiguity of the clustering result. If a distance function is asymmetric, a different clustering result can be obtained depending on the order of processing.

Lemma 2. The distance function $dist(L_i, L_j)$ defined in Section 2.3 is symmetric, *i.e.*, $dist(L_i, L_j) = dist(L_j, L_i)$.
PROOF: To make $dist(L_i, L_j)$ symmetric, we have assigned a longer line segment to L_i and a shorter one to L_j. (The tie can be broken by comparing the internal identifier of a line segment.) Hence, we can easily know that $dist(L_i, L_j)$ is symmetric. □

4.1.2 Notions for Density-Based Clustering

We summarize the notions required for density-based clustering through Definitions 4∼9. Let $\mathcal{D}$ denote the set of all line segments. We change the definitions for points, originally proposed for the algorithm DBSCAN [6], to those for line segments.

Definition 4. The *ε-neighborhood* $N_\varepsilon(L_i)$ of a line segment $L_i \in \mathcal{D}$ is defined by $N_\varepsilon(L_i) = \{L_j \in \mathcal{D} \mid dist(L_i, L_j) \le \varepsilon\}$.

Definition 5. A line segment $L_i \in \mathcal{D}$ is called a *core line segment* w.r.t. ε and $MinLns$ if $|N_\varepsilon(L_i)| \ge MinLns$.

Definition 6. A line segment $L_i \in \mathcal{D}$ is *directly density-reachable* from a line segment $L_j \in \mathcal{D}$ w.r.t. ε and $MinLns$ if (1) $L_i \in N_\varepsilon(L_j)$ and (2) $|N_\varepsilon(L_j)| \ge MinLns$.

Definition 7. A line segment $L_i \in \mathcal{D}$ is *density-reachable* from a line segment $L_j \in \mathcal{D}$ w.r.t. ε and $MinLns$ if there is a chain of line segments $L_j, L_{j-1}, \cdots, L_{i+1}, L_i \in \mathcal{D}$ such that L_k is directly density-reachable from L_{k+1} w.r.t. ε and $MinLns$.

Definition 8. A line segment $L_i \in \mathcal{D}$ is *density-connected* to a line segment $L_j \in \mathcal{D}$ w.r.t. ε and $MinLns$ if there is a line segment $L_k \in \mathcal{D}$ such that both L_i and L_j are density-reachable from L_k w.r.t. ε and $MinLns$.

Definition 9. A non-empty subset $\mathcal{C} \subseteq \mathcal{D}$ is called a *density-connected set* w.r.t. ε and $MinLns$ if $\mathcal{C}$ satisfies the following two conditions:

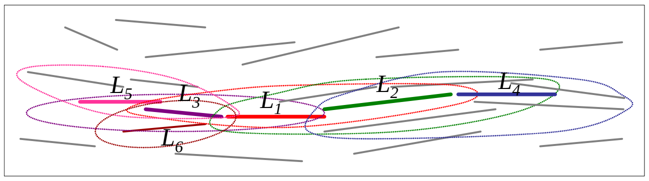
(1) *Connectivity*: $\forall L_i, L_j \in \mathcal{C}$, L_i is density-connected to L_j w.r.t. ε and $MinLns$;

(2) *Maximality*: $\forall L_i, L_j \in \mathcal{D}$, if $L_i \in \mathcal{C}$ and L_j is density-reachable from L_i w.r.t. ε and $MinLns$, then $L_j \in \mathcal{C}$.

We depict these notions in Figure 10. Density-reachability is the transitive closure of direct density-reachability, and this relation is asymmetric. Only core line segments are mutually density-reachable. Density-connectivity, however, is a symmetric relation.

Example 4. Consider the line segments in Figure 10. Let $MinLns$ be 3. Thick line segments indicate core line segments. ε-neighborhoods are represented by irregular ellipses. Based on the above definitions,

- L_1, L_2, L_3, L_4, and L_5 are core line segments;
- L_2 (or L_3) is directly density-reachable from L_1;
- L_6 is density-reachable from L_1, but not vice versa;
- L_1, L_4, and L_5 are all density-connected. □



$$\begin{tikzcd} L_6 & L_5 \arrow[l] \arrow[r, leftrightarrow] \arrow[l, phantom] & L_3 \arrow[r, leftrightarrow] \arrow[ll, bend right] & L_1 \arrow[r, leftrightarrow] & L_2 \arrow[r, leftrightarrow] & L_4 \end{tikzcd}$$

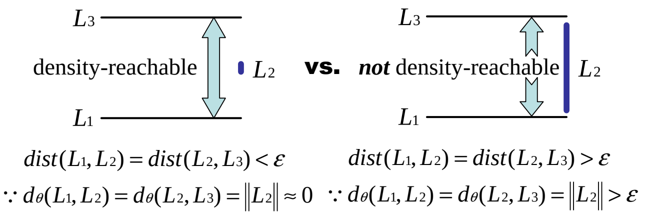
Figure 10: Density-reachability and density-connectivity.

4.1.3 Observations

Since line segments have both direction and length, they show a few interesting features in relation to density-based clustering. Here, we mention our two observations.

We observe that the shape of an ε-neighborhood in line segments is not a circle or a sphere. Instead, its shape is very dependent on data and is likely to be an ellipse or an ellipsoid. Figure 10 shows various shapes of ε-neighborhoods. They are mainly affected by the direction and length of a core line segment and those of border line segments.

Another interesting observation is that a short line segment could drastically degrade the clustering quality. We note that the length of a line segment represents its directional strength; *i.e.*, if a line segment is short, its directional strength is low. This means that the angle distance is small regardless of the actual angle if one of line segments is very short. Let us compare the two sets of line segments in Figure 11. The only difference is the length of L_2. L_1 can be density-reachable from L_3 (or vice versa) via L_2 in Figure 11 (a), but cannot be density-reachable in Figure 11 (b). The result in Figure 11 (a) is counter-intuitive since L_1 and L_3 are far apart. Thus, it turns out that a short line segment might induce *over-clustering*.



L_3
density-reachable
L_2
L_1
vs.
L_3
not density-reachable
L_2
L_1
$dist(L_1, L_2) = dist(L_2, L_3) < \varepsilon$
$\because d_\theta(L_1, L_2) = d_\theta(L_2, L_3) = \|L_2\| \approx 0$
$dist(L_1, L_2) = dist(L_2, L_3) > \varepsilon$
$\because d_\theta(L_1, L_2) = d_\theta(L_2, L_3) = \|L_2\| > \varepsilon$

(a) L_2 is very short. (b) L_2 is not short.

Figure 11: Influence of a very short line segment on clustering.

A simple solution of this problem is to make trajectory partitions longer by adjusting the partitioning criteria in Figure 8. That is, we suppress partitioning at the cost of preciseness. More specifically, to suppress partitioning, we add a small constant to $cost_{nopar}$ in line 6. Our experience indicates that increasing the length of trajectory partitions by 20∼30% generally improves the clustering quality.

4.2 Clustering Algorithm

We now present our density-based clustering algorithm for line segments. Given a set $\mathcal{D}$ of line segments, our algorithm generates a set $\mathcal{O}$ of clusters. It requires two parameters ε and $MinLns$. We define a cluster as a density-connected set. Our algorithm shares many characteristics with the algorithm DBSCAN.

Unlike DBSCAN, however, not all density-connected sets can become clusters. We need to consider the number of trajectories from which line segments have been extracted. This number of trajectories is typically smaller than that of line segments. For example, in the extreme, all the line segments in a density-connected set could be those extracted from one trajectory. We prevent such clusters since they do not explain the behavior of a sufficient number of trajectories. Here, we check the *trajectory cardinality* defined in Definition 10.

Definition 10. The *set of participating trajectories* of a cluster C_i is defined by $PTR(C_i) = \{TR(L_j) \mid \forall L_j \in C_i\}$. Here, $TR(L_j)$ denotes the trajectory from which L_j has been extracted. Then, $|PTR(C_i)|$ is called the *trajectory cardinality* of the cluster C_i.

Figure 12 shows the algorithm *Line Segment Clustering*. Initially, all the line segments are assumed to be unclassified. As the algorithm progresses, they are classified as either a cluster or a noise. The algorithm consists of three steps. In the first step (lines 1∼12), the algorithm computes the ε-neighborhood of each unclassified line segment L. If L is determined as a core line segment (lines 7∼10), the algorithm performs the second step to expand a cluster (line 9). The cluster currently contains only $N_\varepsilon(L)$. In the second step (lines 17∼28), the algorithm computes the density-connected set of a core line segment. The procedure *ExplandCluster*() computes the directly density-reachable line segments (lines 19∼21) and adds them to the current cluster (lines 22∼24). If a newly added line segment is unclassified, it is added to the queue $\mathcal{Q}$ for more expansion since it might be a core line segment (lines 25∼26); otherwise, it is not added to $\mathcal{Q}$ since we already know that it is not a core line segment. In the third step (lines 13∼16), the algorithm checks the trajectory cardinality of each cluster. If its trajectory cardinality is below the threshold, the algorithm filters out the corresponding cluster.

In Lemma 3, we present the time complexity of this clustering algorithm.

Lemma 3. The time complexity of the algorithm in Figure 12 is $O(n \log n)$ if a spatial index is used, where n is the total number of line segments in a database. Otherwise, it is $O(n^2)$ for high (≥ 2) dimensionality.

Proof: From the lines 5 and 20 in Figure 12, we know that an ε-neighborhood query must be executed for each line segment. Hence, the time complexity of the algorithm is $n \times$ (the time complexity of an ε-neighborhood query). If we do not use any index, we have to scan all the line segments in a database. Thus, the time complexity would be $O(n^2)$. If we use an appropriate index such as the R-tree [10], we are able to efficiently find the line segments in an ε-neighborhood by traversing the index. Thus, the time complexity is reduced to $O(n \log n)$. □

```
Algorithm Line Segment Clustering

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

    /* STEP 2: compute a density-connected set */
17: ExpandCluster($\mathcal{Q}$, $clusterId$, $\varepsilon$, $MinLns$) {
18:     while ($\mathcal{Q} \neq \emptyset$) do
19:         Let $M$ be the first line segment in $\mathcal{Q}$;
20:         Compute $N_\varepsilon(M)$;
21:         if ($|N_\varepsilon(M)| \geq MinLns$) then
22:             for each ($X \in N_\varepsilon(M)$) do
23:                 if ($X$ is unclassified or noise) then
24:                     Assign $clusterId$ to $X$;
25:                 if ($X$ is unclassified) then
26:                     Insert $X$ into the queue $\mathcal{Q}$;
27:         Remove $M$ from the queue $\mathcal{Q}$;
28: }
```

Figure 12: A density-based clustering algorithm for line segments.

This algorithm can be easily extended so as to support *trajectories with weights*. This extension will be very useful in many applications. For example, it is natural that a stronger hurricane should have a higher weight. To accomplish this, we need to modify the method of deciding the cardinality of an ε-neighborhood (*i.e.*, $|N_\varepsilon(L)|$). Instead of simply counting the number of line segments, we compute the weighted count by summing up the weights of the line segments.

We note that our distance function is not a metric since it does not obey the triangle inequality. It is easy to see an example of three line segments L_1, L_2, and L_3 where $dist(L_1, L_3) > dist(L_1, L_2) + dist(L_2, L_3)$. This makes direct application of traditional spatial indexes difficult. Several techniques can be used to address this issue. For example, we can adopt *constant shift embedding* [18] to convert a distance function that does not follow the triangle inequality to another one that follows. We do not consider it here leaving it as the topic of a future paper.

4.3 Representative Trajectory of a Cluster

The *representative trajectory* of a cluster describes the overall movement of the trajectory partitions that belong to the cluster. It can be considered a model [1] for clusters. We need to extract quantitative information on the movement within a cluster such that domain experts are able to understand the movement in the trajectories. Thus, in order to gain full practical potential from trajectory clustering, this representative trajectory is absolutely required.

Figure 13 illustrates our approach of generating a representative trajectory. A representative trajectory is a sequence of points $RTR_i = p_1 p_2 p_3 \cdots p_j \cdots p_{len_i}$ $(1 \le i \le num_{clus})$. These points are determined using a sweep line approach. While sweeping a vertical line across line segments in the direction of the *major axis* of a cluster, we count the number of the line segments hitting the sweep line. This number changes only when the sweep line passes a starting point or an ending point. If this number is equal to or greater than $MinLns$, we compute the *average coordinate* of those line segments with respect to the major axis and insert the average into the representative trajectory; otherwise, we skip the current point (*e.g.*, the 5th and 6th positions in Figure 13). Besides, if a previous point is located too close (*e.g.*, the 3rd position in Figure 13), we skip the current point to smooth the representative trajectory.

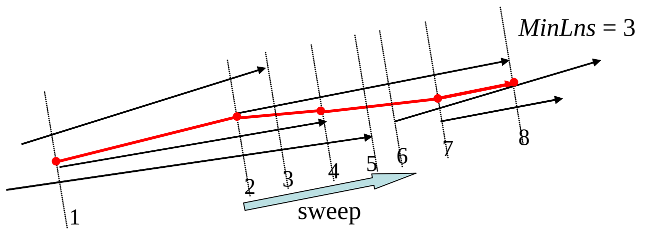


Figure 13: An example of a cluster and its representative trajectory.

We explain the above approach in more detail. To represent the major axis of a cluster, we define the *average direction vector* in Definition 11. We add up vectors instead of their direction vectors (*i.e.*, unit vectors) and normalize the result. This is a nice heuristic giving the effect of a longer vector contributing more to the average direction vector. We compute the average direction vector over the set of vectors representing each line segment in the cluster.

Definition 11. Suppose a set of vectors $\mathcal{V} = \{\overrightarrow{v_1}, \overrightarrow{v_2}, \overrightarrow{v_3}, \cdots, \overrightarrow{v_n}\}$. The *average direction vector* $\overrightarrow{\mathcal{V}}$ of $\mathcal{V}$ is defined as Formula (8). Here, $|\mathcal{V}|$ is the cardinality of $\mathcal{V}$.

$$\overrightarrow{\mathcal{V}} = \frac{\overrightarrow{v_1} + \overrightarrow{v_2} + \overrightarrow{v_3} + \cdots + \overrightarrow{v_n}}{|\mathcal{V}|} \tag{8}$$

As stated above, we compute the average coordinate *with respect to the average direction vector.* To facilitate this computation, we rotate the axes so that the X axis is made to be parallel to the average direction vector. Here, the rotation matrix in Formula (9) is used.[3] The angle ϕ can be obtained using the inner product between the average direction vector and the unit vector $\hat{x}$. After computing an average in the $X'Y'$ coordinate system as in Figure 14, it is translated back to a point in the XY coordinate system.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{9}$$

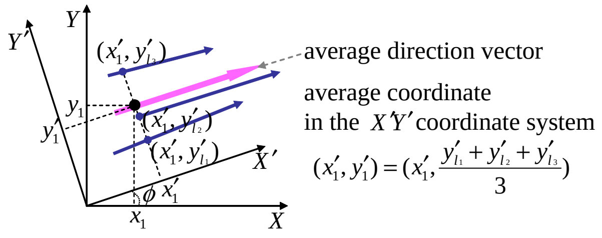


Figure 14: Rotation of the X and Y axes.

Figure 15 shows the algorithm *Representative Trajectory Generation*. We first compute the average direction vector and rotate the axes temporarily (lines 1∼2). We then sort the starting and ending points by the coordinate of the rotated axis (lines 3∼4). While scanning the starting and ending points in the sorted order, we count the number of line segments and compute the average coordinate of those line segments (lines 5∼12).

```
Algorithm **Representative Trajectory Generation**

INPUT: (1) A cluster $C_i$ of line segments, (2) $MinLns$,
       (3) A smoothing parameter $\gamma$
OUTPUT: A representative trajectory $RTR_i$ for $C_i$
ALGORITHM:
01: Compute the average direction vector $\overrightarrow{\mathcal{V}}$;
02: Rotate the axes so that the $X$ axis is parallel to $\overrightarrow{\mathcal{V}}$;
03: Let $\mathcal{P}$ be the set of the starting and ending points of
    the line segments in $C_i$;
    /* $X'$-value denotes the coordinate of the $X'$ axis */
04: Sort the points in the set $\mathcal{P}$ by their $X'$-values;
05: for each ($p \in \mathcal{P}$) do
        /* count $num_p$ using a sweep line (or plane) */
06:     Let $num_p$ be the number of the line segments
        that contain the $X'$-value of the point $p$;
07:     if ($num_p \ge MinLns$) then
08:         $diff$ := the difference in $X'$-values between $p$
                  and its immediately previous point;
09:         if ($diff \ge \gamma$) then
10:             Compute the average coordinate $avg'_p$;
11:             Undo the rotation and get the point $avg_p$;
12:             Append $avg_p$ to the end of $RTR_i$;
```

Figure 15: An algorithm for generating the representative trajectory.

[3]We assume two dimensions for ease of exposition. The same approach can be applied also to three dimensions.

4.4 Heuristic for Parameter Value Selection

We first present the heuristic for selecting the value of the parameter ε. We adopt the entropy [19] theory. In information theory, the entropy relates to the amount of uncertainty about an event associated with a given probability distribution. If all the outcomes are equally likely, then the entropy should be maximal.

Our heuristic is based on the following observation. In the worst clustering, $|N_\varepsilon(L)|$ tends to be uniform. That is, for too small an ε, $|N_\varepsilon(L)|$ becomes 1 for almost all line segments; for too large an ε, it becomes num_{ln} for almost all line segments, where num_{ln} is the total number of line segments. Thus, the entropy becomes maximal. In contrast, in a good clustering, $|N_\varepsilon(L)|$ tends to be skewed. Thus, the entropy becomes smaller.

We use the entropy definition of Formula (10). Then, we find the value of ε that minimizes $H(X)$. This optimal ε can be efficiently obtained by a simulated annealing [14] technique.

$$H(X) = \sum_{i=1}^{n} p(x_i)\log_2 \frac{1}{p(x_i)} = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i), \quad (10)$$
$$\text{where } p(x_i) = \frac{|N_\varepsilon(x_i)|}{\sum_{j=1}^{n}|N_\varepsilon(x_j)|} \text{ and } n = num_{ln}$$

We then present the heuristic for selecting the value of the parameter $MinLns$. We compute the average $avg_{|N_\varepsilon(L)|}$ of $|N_\varepsilon(L)|$ at the optimal ε. This operation induces no additional cost since it can be done while computing $H(X)$. Then, we determine the optimal $MinLns$ as $(avg_{|N_\varepsilon(L)|} + 1 \sim 3)$. This is natural since $MinLns$ should be greater than $avg_{|N_\varepsilon(L)|}$ to discover meaningful clusters.

It is not obvious that these parameter values estimated by our heuristic are truly optimal. Nevertheless, we believe our heuristic provides a reasonable range where the optimal value is likely to reside. Domain experts are able to try a few values around the estimated value and choose the optimal one by visual inspection.

5. EXPERIMENTAL EVALUATION

In this section, we evaluate the effectiveness of our trajectory clustering algorithm TRACLUS. We describe the experimental data and environment in Section 5.1. We present the results for two real data sets in Sections 5.2 and 5.3. We briefly discuss the effects of parameter values in Section 5.4. Finally, we show robustness to noises in Section 5.5.

5.1 Experimental Setting

We use two real trajectory data sets: the hurricane track data set[4] and the animal movement data set[5].

The hurricane track data set is called *Best Track*. Best Track contains the hurricane's latitude, longitude, maximum sustained surface wind, and minimum sea-level pressure at 6-hourly intervals. We extract the latitude and longitude from Best Track for experiments. We use the Atlantic hurricanes from the years 1950 through 2004. This data set has 570 trajectories and 17736 points.

The animal movement data set has been generated by the Starkey project. This data set contains the radio-telemetry locations (with other information) of elk, deer, and cattle from the years 1993 through 1996. We extract the x and y coordinates from the telemetry data for experiments. We use elk's movements in 1993 and deer's movements in 1995. We call each data set *Elk1993* and *Deer1995*, respectively. Elk1993 has 33 trajectories and 47204 points; Deer1995 has 32 trajectories and 20065 points. We note that trajectories in the animal movement data set are much longer than those in the hurricane track data set.

We attempt to measure the clustering quality while varying the values of ε and $MinLns$. Unfortunately, there is no well-defined measure for density-based clustering methods. Thus, we define a simple quality measure for a ballpark analysis. We use the Sum of Squared Error (SSE) [11]. In addition to the SSE, we consider the noise penalty to penalize incorrectly classified noises. The noise penalty becomes larger if we select too small ε or too large $MinLns$. Consequently, our quality measure is the sum of the total SSE and the noise penalty as Formula (11). Here, $\mathcal{N}$ denotes the set of all noise line segments. The purpose of this measure is to get a *hint* of the clustering quality.

$$\begin{aligned} QMeasure &= Total\ SSE + Noise\ Penalty \quad (11)\\ &= \sum_{i=1}^{num_{clus}} \left(\frac{1}{2|C_i|} \sum_{x \in C_i} \sum_{y \in C_i} dist(x,y)^2 \right) + \\ &\quad \frac{1}{2|\mathcal{N}|} \sum_{w \in \mathcal{N}} \sum_{z \in \mathcal{N}} dist(w,z)^2 \end{aligned}$$

We conduct all the experiments on a Pentium-4 3.0 GHz PC with 1 GBytes of main memory, running on Windows XP. We implement our algorithm and visual inspection tool in C++ using Microsoft Visual Studio 2005.

5.2 Results for Hurricane Track Data

Figure 16 shows the entropy as ε is varied. The minimum is achieved at $\varepsilon = 31$. Here, $avg_{|N_\varepsilon(L)|}$ is 4.39. According to our heuristic, we try parameter values around $\varepsilon = 31$ and $MinLns = 5{\sim}7$. Using visual inspection and domain knowledge, we are able to obtain the optimal parameter values: $\varepsilon = 30$ and $MinLns = 6$. We note that the optimal value $\varepsilon = 30$ is very close to the estimated value $\varepsilon = 31$.

Figure 17 shows the quality measure as ε and $MinLns$ are varied. The smaller $QMeasure$ is, the better the clustering quality is. There is a little discrepancy between the actual clustering quality and our measure. Visual inspection results show that the best clustering is achieved at $MinLns = 6$, not at $MinLns = 5$. Nevertheless, our measure is shown to be a good indicator of the actual clustering quality within the same $MinLns$ value. That is, if we consider only the result for $MinLns = 6$, we can see that our measure becomes nearly minimal when the optimal value of ε is used.

Figure 18 shows the clustering result using the optimal parameter values. Thin green lines display trajectories, and thick red lines *representative trajectories*. We point out that these representative trajectories are exactly *common sub-trajectories*. Here, the number of clusters is that of red lines. We observe that seven clusters are identified.

The result in Figure 18 is quite reasonable. We know that some hurricanes move along a curve, changing their direction from east-to-west to south-to-north, and then to west-to-east. On the other hand, some hurricanes move along a straight east-to-west line or a straight west-to-east line. The lower horizontal cluster represents the east-to-west move-

[4]http://weather.unisys.com/hurricane/atlantic/
[5]http://www.fs.fed.us/pnw/starkey/data/tables/

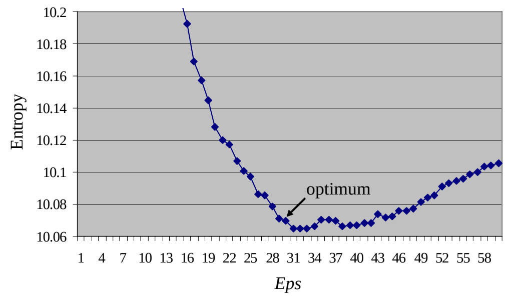


Figure 16: Entropy for the hurricane data.

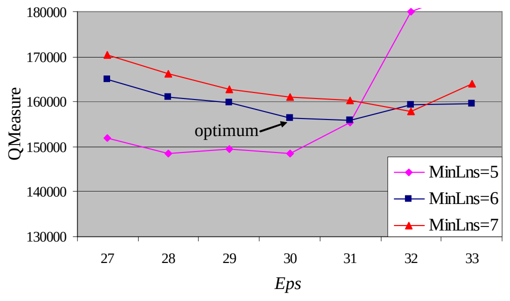
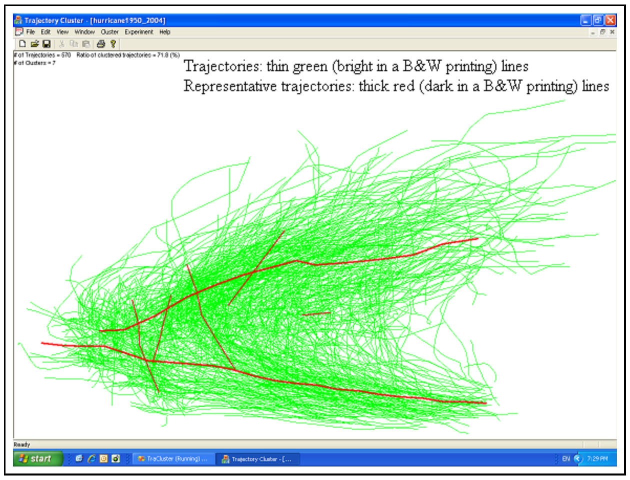


Figure 17: Quality measure for the hurricane data.



Trajectory Cluster - [hurricane1950_2004]
File Edit View Window Cluster Experiment Help
Trajectories: thin green (bright in a B&W printing) lines
Representative trajectories: thick red (dark in a B&W printing) lines
Ready
start

Figure 18: Clustering result for the hurricane data.

ments, the upper horizontal one the west-to-east movements, and the vertical ones the south-to-north movements.

5.3 Results for Animal Movement Data

5.3.1 *Elk's Movements in 1993*

Figure 19 shows the entropy as ε is varied. The minimum is achieved at $\varepsilon = 25$. Here, $avg_{|N_\varepsilon(L)|}$ is 7.63. Visually, we obtain the optimal parameter values: $\varepsilon = 27$ and $MinLns = 9$. Again, the optimal value $\varepsilon = 27$ is very close to the estimated value $\varepsilon = 25$.

Figure 20 shows the quality measure as ε and $MinLns$ are varied. We observe that our measure becomes nearly minimal when the optimal parameter values are used. The correlation between the actual clustering quality and our measure, $QMeasure$, is shown to be stronger in Figure 20 than in Figure 17.

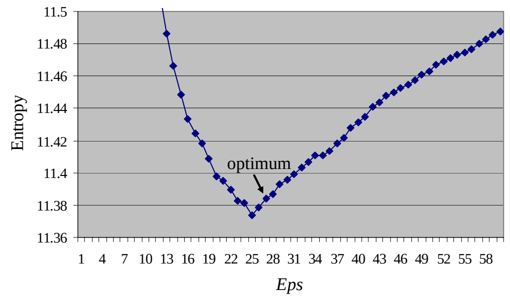


Figure 19: Entropy for the Elk1993 data.

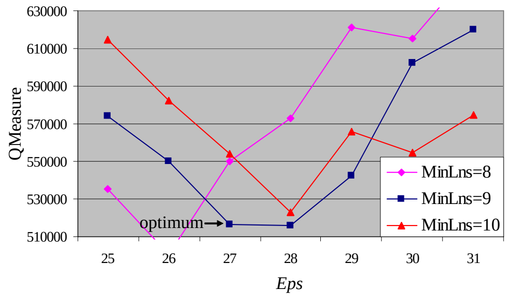


Figure 20: Quality measure for the Elk1993 data.

Figure 21 shows the clustering result using the optimal parameter values. We observe that thirteen clusters are discovered in the most of the dense regions. Although the upper-right region looks dense, we find out that the elks actually moved along different paths. Hence, the result having no cluster in that region is verified to be correct.

5.3.2 *Deer's Movements in 1995*

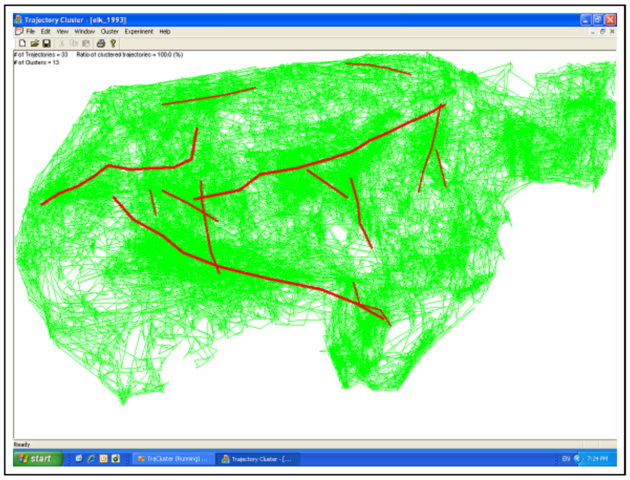
Figure 22 shows the clustering result using the optimal parameter values ($\varepsilon = 29$ and $MinLns = 8$). The result indicates that two clusters are discovered in the two most dense regions. This result is exactly what we expect. The center region is not so dense to be identified as a cluster. Due to space limit, we omit the detailed figures for the entropy and quality measure.

5.4 Effects of Parameter Values

We have tested the effects of parameter values on the clustering result. If we use a smaller ε or a larger $MinLns$ compared with the optimal ones, our algorithm discovers a larger number of smaller (*i.e.*, having fewer line segments) clusters. In contrast, if we use a larger ε or a smaller $MinLns$, our algorithm discovers a smaller number of larger clusters. For example, in the hurricane track data, when $\varepsilon = 25$, nine clusters are discovered, and each cluster contains 38 line segments on average; in contrast, when $\varepsilon = 35$, three clusters are discovered, and each cluster contains 174 line segments on average.

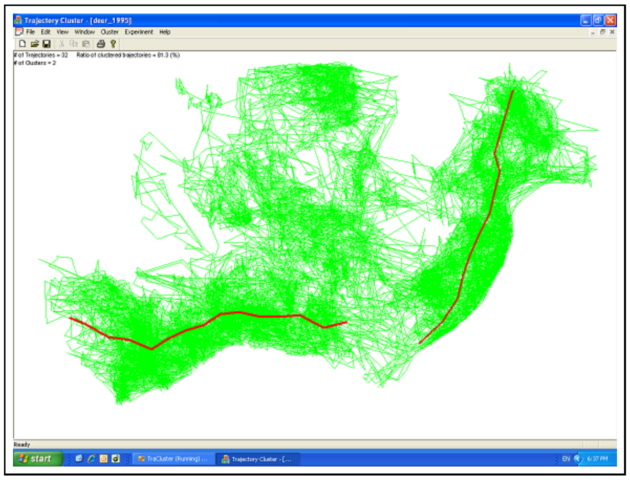
5.5 Robustness to Noises

Our algorithm is robust to noises because it is based on the DBSCAN algorithm. DBSCAN is known to be robust to noises. We have executed our algorithm over a synthetic data set that contains many noises. Here, 25% of trajectories are generated as noises. Figure 23 shows the clustering result. We observe that the clusters are correctly identified despite many noises.


Trajectory Cluster - [elk_1993]
File Edit View Window Cluster Experiment Help
Ready
start

```
# of Trajectories = 33    Ratio of clustered trajectories = 100.0 (%)
# of Clusters = 13
```

Figure 21: Clustering result for the Elk1993 data.


Trajectory Cluster - [deer_1995]
File Edit View Window Cluster Experiment Help
Ready
start

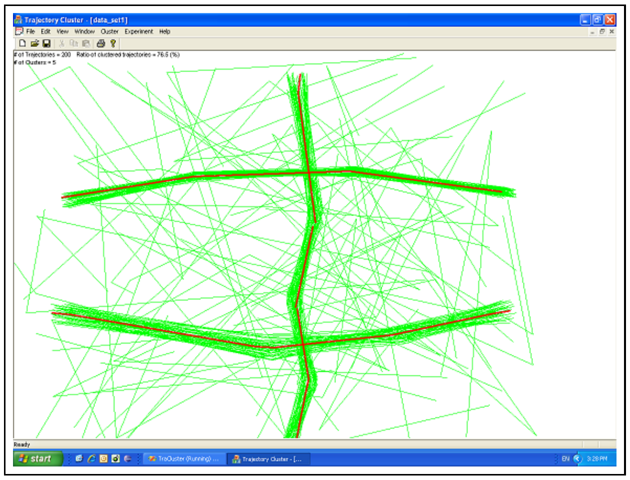
```
# of Trajectories = 32    Ratio of clustered trajectories = 81.3 (%)
# of Clusters = 2
```

Figure 22: Clustering result for the Deer1995 data.

6. RELATED WORK

Clustering has been extensively studied in the data mining area. Clustering algorithms can be classified into four categories: partitioning methods (*e.g.*, k-means [16]), hierarchical methods (*e.g.*, BIRCH [23]), density-based methods (*e.g.*, DBSCAN [6] and OPTICS [2]), and grid-based methods (*e.g.*, STING [21]). Our algorithm TRACLUS falls in the category of density-based methods. In density-based methods, clusters are regions of high density separated by regions of low density. DBSCAN has been regarded as the most representative density-based clustering algorithm, and OPTICS has been devised to reduce the burden of determining parameter values in DBSCAN. The majority of previous research has been focused on clustering of point data.

The most similar work to ours is the trajectory clustering algorithm proposed by Gaffney *et al.* [7, 8]. It is based on probabilistic modeling of a set of trajectories. Formally, the probability density function of observed trajectories is a mixture density: $P(y_j|x_j,\theta) = \sum_k^K f_k(y_j|x_j,\theta_k)w_k$, where $f_k(y_j|x_j,\theta_k)$ is the density component, w_k is the weight, and θ_k is the set of parameters for k-th component. Here, θ_k and w_k can be estimated from the trajectory data using the Expectation-Maximization (EM) algorithm. The estimated density components $f_k(y_j|x_j,\theta_k)$ are then interpreted as clusters. The fundamental difference of this algorithm from TRACLUS is being based on probabilistic clustering and clustering trajectories as a whole.


Trajectory Cluster - [data_set1]
File Edit View Window Cluster Experiment Help
Ready
start

```
# of Trajectories = 200    Ratio of clustered trajectories = 76.5 (%)
# of Clusters = 5
```

Figure 23: Clustering result for a synthetic data set having many noises.

Distance measures for searching similar trajectories have been proposed recently. Vlachos *et al.* [20] have proposed the distance measure LCSS, and Chen *et al.* [5] the distance measure EDR. Both LCSS and EDR are based on the edit distance and are extended so as to be robust to noises, shifts, and different lengths that occur due to sensor failures, errors in detection techniques, and different sampling rates. EDR can represent the gap between two similar subsequences more precisely compared with LCSS [5]. Besides, dynamic time warping has been widely adopted as a distance measure for time series [12]. These distance measures, however, are not adequate for our problem since they are originally designed to compare the whole trajectory (especially, the whole time-series sequence). In other words, the distance could be large although some portions of trajectories are very similar. Hence, it is hard to detect only similar portions of trajectories.

The MDL principle has been successfully used for diverse applications, such as graph partitioning [3] and distance function design for strings [13]. For graph partitioning, a graph is represented as a binary matrix, and then, the matrix is divided into disjoint row and column groups such that the rectangular intersections of groups are homogeneous. Here, the MDL principle is used to automatically select the number of row and column groups [3]. For distance function design, data compression is used to measure the similarity between two strings. This idea is tightly connected with the MDL principle [13].

7. DISCUSSION AND CONCLUSIONS

7.1 Discussion

We discuss some possible extensions of our trajectory clustering algorithm.

1. *Extensibility*: We can support *undirected* or *weighted* trajectories. We handle undirected ones using the simplified angle distance and weighted ones using the extended cardinality of an ε-neighborhood.

2. *Parameter Insensitivity*: We can make our algorithm more insensitive to parameter values. A number of approaches, *e.g.*, OPTICS [2], have been developed for this purpose in the context of point data. We are applying these approaches to trajectory data.
3. *Efficiency*: We can improve the clustering performance by using an index to execute an ε-neighborhood query. The major difficulty is that our distance function is not a metric. We will adopt an indexing technique for a non-metric distance function [18].
4. *Movement Patterns*: We will extend our algorithm to support various types of movement patterns, especially *circular motion*. Our algorithm primarily supports straight motion. We believe this extension can be done by enhancing the approach of generating a representative trajectory.
5. *Temporal Information*: We will extend our algorithm to take account of temporal information during clustering. One can expect that time is also recorded with location. This extension will significantly improve the usability of our algorithm.

7.2 Conclusions

In this paper, we have proposed a novel framework, the *partition-and-group* framework, for clustering trajectories. Based on this framework, we have developed the trajectory clustering algorithm *TRACLUS*. As the algorithm progresses, a trajectory is partitioned into a set of line segments at characteristic points, and then, similar line segments in a dense region are grouped into a cluster. The main advantage of TRACLUS is the discovery of common sub-trajectories from a trajectory database.

To show the effectiveness of TRACLUS, we have performed extensive experiments using two real data sets: hurricane track data and animal movement data. Our heuristic for parameter value selection has been shown to estimate the optimal parameter values quite accurately. We have implemented a visual inspection tool for cluster validation. The visual inspection results have demonstrated that TRACLUS effectively identifies common sub-trajectories as clusters.

Overall, we believe that we have provided a new paradigm in trajectory clustering. Data analysts are able to get a new insight into trajectory data by virtue of the common sub-trajectories. This work is just the first step, and there are many challenging issues discussed above. We are currently investigating into detailed issues as a further study.

8. REFERENCES

[1] Achtert, E., Böhm, C., Kriegel, H.-P., Kröger, P., and Zimek, A., "Deriving Quantitative Models for Correlation Clusters," In *Proc. 12th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, Philadelphia, Pennsylvania, pp. 4–13, Aug. 2006.

[2] Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J., "OPTICS: Ordering Points to Identify the Clustering Structure," In *Proc. 1999 ACM SIGMOD Int'l Conf. on Management of Data*, Philadelphia, Pennsylvania, pp. 49–60, June 1999.

[3] Chakrabarti, D., Papadimitriou, S., Modha, D. S., and Faloutsos, C., "Fully Automatic Cross-Associations," In *Proc. 10th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, Seattle, Washington, pp. 79–88, Aug. 2004.

[4] Chen, J., Leung, M. K. H., and Gao, Y., "Noisy Logo Recognition Using Line Segment Hausdorff Distance," *Pattern Recognition*, 36(4): 943–955, 2003.

[5] Chen, L., Özsu, M. T., and Oria, V., "Robust and Fast Similarity Search for Moving Object Trajectories," In *Proc. 2005 ACM SIGMOD Int'l Conf. on Management of Data*, Baltimore, Maryland, pp. 491–502, June 2005.

[6] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X., "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," In *Proc. 2nd Int'l Conf. on Knowledge Discovery and Data Mining*, Portland, Oregon, pp. 226–231, Aug. 1996.

[7] Gaffney, S. and Smyth, P., "Trajectory Clustering with Mixtures of Regression Models," In *Proc. 5th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, San Diego, California, pp. 63–72, Aug. 1999.

[8] Gaffney, S., Robertson, A., Smyth, P., Camargo, S., and Ghil, M., Probabilistic Clustering of Extratropical Cyclones Using Regression Mixture Models, Technical Report UCI-ICS 06-02, University of California, Irvine, Jan. 2006.

[9] Grünwald, P., Myung, I. J., and Pitt, M., *Advances in Minimum Description Length: Theory and Applications*, MIT Press, 2005.

[10] Guttman, A., "R-Trees: A Dynamic Index Structure for Spatial Searching," In *Proc. 1984 ACM SIGMOD Int'l Conf. on Management of Data*, Boston, Massachusetts, pp. 47–57, June 1984.

[11] Han, J. and Kamber, M., *Data Mining: Concepts and Techniques*, 2nd ed., Morgan Kaufmann, 2006.

[12] Keogh, E. J., "Exact Indexing of Dynamic Time Warping," In *Proc. 28th Int'l Conf. on Very Large Data Bases*, Hong Kong, China, pp. 406–417, Aug. 2002.

[13] Keogh, E. J., Lonardi, S., and Ratanamahatana, C. A., "Towards Parameter-Free Data Mining," In *Proc. 10th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, Seattle, Washington, pp. 206–215, Aug. 2004.

[14] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, 220(4598): 671–680, 1983.

[15] Lee, T. C. M., "An Introduction to Coding Theory and the Two-Part Minimum Description Length Principle," *International Statistical Review*, 69(2): 169–184, 2001.

[16] Lloyd, S., "Least Squares Quantization in PCM," *IEEE Trans. on Information Theory*, 28(2): 129–137, 1982.

[17] Powell, M. D. and Aberson, S. D., "Accuracy of United States Tropical Cyclone Landfall Forecasts in the Atlantic Basin (1976-2000)," *Bull. of the American Meteorological Society*, 82(12): 2749–2767, 2001.

[18] Roth, V., Laub, J., Kawanabe, M., and Buhmann, J. M., "Optimal Cluster Preserving Embedding of Nonmetric Proximity Data," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(12): 1540–1551, 2003.

[19] Shannon, C. E., "A Mathematical Theory of Communication," *The Bell System Technical Journal*, 27: 379–423 and 623–656, 1948.

[20] Vlachos, M., Gunopulos, D., and Kollios, G., "Discovering Similar Multidimensional Trajectories," In *Proc. 18th Int'l Conf. on Data Engineering*, San Jose, California, pp. 673–684, Feb./Mar. 2002.

[21] Wang, W., Yang, J., and Muntz, R. R., "STING: A Statistical Information Grid Approach to Spatial Data Mining," In *Proc. 23rd Int'l Conf. on Very Large Data Bases*, Athens, Greece, pp. 186–195, Aug. 1997.

[22] Wisdom, M. J., Cimon, N. J., Johnson, B. K., Garton, E. O., and Thomas, J. W., "Spatial Partitioning by Mule Deer and Elk in Relation to Traffic," *Trans. of the North American Wildlife and Natural Resources Conf.*, Spokane, Washington, pp. 509–530, Mar. 2004.

[23] Zhang, T., Ramakrishnan, R., and Livny, M., "BIRCH: An Efficient Data Clustering Method for Very Large Databases," In *Proc. 1996 ACM SIGMOD Int'l Conf. on Management of Data*, Montreal, Canada, pp. 103–114, June 1996.

APPENDIX

A. Discussion about the advantage of our distance measure over a naive one

The sum of the distances of endpoints may not be adequate in many cases. Figure 24 shows a typical example. Suppose there are three line segments L_1, L_2, and L_3. We can easily see that L_2 is more similar to L_1 than L_3 is. Assume that we use the sum of the distances of the endpoints as a distance measure. Then, the distance between L_1 and L_2 is the same as that between L_1 and L_3, *i.e.*, $200\sqrt{2}$. Thus, we cannot decide which one is more similar to L_1 even though it is obvious. This simple example intuitively illustrates the importance of the angle distance.

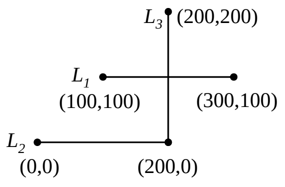


Figure 24: The importance of the angle distance.

B. Supporting example for different weights in our distance function

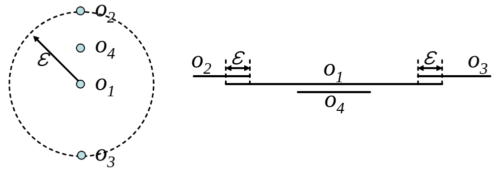
We include the weights $w_\perp$, $w_\parallel$, and w_θ to make our distance function more general. Our experience indicates that the default value ($w_\perp = w_\parallel = w_\theta = 1$) generally works well in many applications and data sets. On the other hand, different weights may be beneficial to some applications. For example, the direction of a hurricane having different angles with and distances from equators may strongly influence the weather. Thus, clustering a set of hurricane-trail trajectories by taking different weights for the perpendicular, parallel, and angle distances may lead to the discovery of clusters that are more correlated to weather patterns. Therefore, assigning different weights may sometimes produce more interesting clustering results than assigning a uniform weight.

C. Supporting example for our formulation of $L(H)$

Let two trajectories be TR_1 = (100,100) → (200,200) → (300,100) and TR_2 = (200,200) → (300,300) → (400,200). In addition, by shifting TR_1 and TR_2 by 10000×10000, we derive two new trajectories: TR_3 and TR_4. That is, TR_3 = (10100,10100) → (10200,10200) → (10300,10100), and TR_4 = (10200,10200) → (10300,10300) → (10400,10200). In principle, the clustering result of TR_1 and TR_2 should be the same as that of TR_3 and TR_4, due to the exactly same relative distances. However, if we describe the hypothesis by its endpoints, we cannot obtain the same clustering result. $L(H)$ calculated for TR_3 (or TR_4) is much larger than that for TR_1 (or TR_2), so TR_3 (or TR_4) will be partitioned at different points compared with TR_1 (or TR_2). This will lead to undesirable clustering results. Intuitively, suppose we would like to cluster a set of wind-trail trajectories in US, with San Diego defined close to (0,0) but Boston defined closed to (10000,10000). If we define $L(H)$ based on endpoints, a similar set of wind-trails at San Diego could be clustered rather differently from that at Boston. This is obviously undesirable. Using $L(H)$ defined based on the length of a line segment, we will derive the same and desirable results in such trajectory clustering.

D. Discussion about our design decision using DBSCAN rather than OPTICS

One might argue that we had better use the algorithm OPTICS [2] to reduce the burden of parameter value selection. However, based on our tests, we expect that directly using OPTICS for line segments would provide results not as good as those for points. The reason is that the reachability-distances [2] of cluster objects tend to be higher (*i.e.*, closer to ε) in line segments than in points and that cluster objects are made more indistinguishable from noises. Consider the pairwise distance among objects in an ε-neighborhood. As shown in Figure 25, this distance in points is limited to $2\,\varepsilon$, whereas that in line segments is not. Then, consider the probability that another core object exists within the current reachability-distance. This probability is lower in line segments than in points, thus rendering the reachability-distance still high. For example, o_4 in Figure 25 (a) can decrease the reachability-distance of o_2, whereas o_4 in Figure 25 (b) can not.



$dist(o_2, o_3) = 2\,\varepsilon$ (a) points. $dist(o_2, o_3) \gg 2\,\varepsilon$ (b) line segments.

Figure 25: Comparison of the distance among objects in an ε-neighborhood.