Trajectory Outlier Detection: A Partition-and-Detect Framework

Jae-Gil Lee, Jiawei Han, Xiaolei Li

Department of Computer Science, University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{jaegil, hanj, xli10}@uiuc.edu

***Abstract*— Outlier detection has been a popular data mining task. However, there is a lack of serious study on outlier detection for trajectory data. Even worse, an existing trajectory outlier detection algorithm has limited capability to detect outlying *sub*-trajectories. In this paper, we propose a novel *partition-and-detect* framework for trajectory outlier detection, which partitions a trajectory into a set of line segments, and then, detects outlying line segments for trajectory outliers. The primary advantage of this framework is to detect outlying *sub*-trajectories from a trajectory database. Based on this partition-and-detect framework, we develop a trajectory outlier detection algorithm *TRAOD*. Our algorithm consists of two phases: *partitioning* and *detection*. For the first phase, we propose a two-level trajectory partitioning strategy that ensures both high quality and high efficiency. For the second phase, we present a hybrid of the distance-based and density-based approaches. Experimental results demonstrate that TRAOD correctly detects outlying sub-trajectories from real trajectory data.**

I. Introduction

An *outlier* is a data object that is grossly different from or inconsistent with the remaining set of data [1]. It has been known that "one person's noise could be another person's signal." Indeed, the outliers may be of particular interest, such as for the detection of credit card fraud and the monitoring of criminal activities in electronic commerce. There are many outlier detection algorithms reported in the literature. They can be classified into distribution-based [2], distance-based [3], [4], [5], [6], density-based [7], [8], and deviation-based [9] algorithms. Most of them are designed to detect outliers from relational tuples (*i.e.*, multi-dimensional *point* data).

Recent improvements in satellites and tracking facilities have made it possible to collect a huge amount of *trajectory* data of moving objects. Examples include vehicle positioning data, hurricane tracking data, and animal movement data. There is an increasing interest to perform data analysis over trajectory data. Since outlier analysis is a popular data mining task, a powerful outlier detection algorithm for trajectories is needed urgently.

Despite its importance, trajectory outlier detection has not been paid much attention. Knorr *et al.* [5] have presented one of very few attempts. In this technique, a trajectory is represented by a set of key features instead of a sequence of points. That is, a trajectory is summarized by the coordinates of the starting and ending points; the average, minimum, and maximum values of the directional vector; and the average, minimum, and maximum velocities. The distance function is simply defined as the weighted sum of the difference of these values. Then, a distance-based algorithm [3], [4], [5] is applied to detecting trajectory outliers. This technique compares trajectories *as a whole*; *i.e.*, the basic unit of outlier detection is the whole trajectory.

Our key observation is that comparing trajectories as a whole, with help from the summary information, might not be able to detect *outlying portions* of the trajectories. We note that a trajectory may have a long and complicated path. Hence, even though some portions of a trajectory show a unusual behavior, these differences might be averaged out over the whole trajectory.

Example 1: Consider the five trajectories in Figure 1. It is obvious that the thick portion of a trajectory TR_3 is quite different from neighboring trajectories. However, the previous technique [5] cannot detect this unusual behavior since the differences are averaged out over the whole trajectory; *i.e.*, the overall behavior of the trajectory TR_3 is similar to those of the neighboring trajectories. Thus, we miss this possibly important information. □

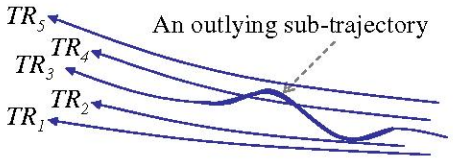


Fig. 1. An example of an outlying sub-trajectory.

Our solution is to partition a trajectory into a set of line segments and then detect outlying line segments. This framework is called a *partition-and-detect* framework. The primary advantage of the partition-and-detect framework is the detection of outlying *sub*-trajectories from a trajectory database. This is exactly the reason why we partition a trajectory into a set of line segments.

We contend that detecting the outlying sub-trajectories is very useful. There are many examples in real situations. Here, we present a possible application scenario.

Example 2: Meteorologists are trying to figure out the cause of *sudden changes in hurricane's path* [10]. Predicting sudden changes is of prime importance since it is crucial for issuing an evacuation order early. Hurricane Charley in

978-1-4244-1837-4/08/$25.00 © 2008 IEEE

ICDE 2008

Aug. 2004 is notorious for its unexpected path (*i.e.*, hard right turn) and rapid intensification. Since Charley was expected to hit the land closer to Tampa, many residents around Punta Gorda, Fla., were caught unprepared. This sudden change can be considered as an outlying sub-trajectory if the previous path follows a usual pattern. Thus, detecting outlying sub-trajectories helps reveal the cause of sudden hurricane track changes. □

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

(1) The *partitioning* phase: Each trajectory is partitioned first in *coarse* granularity and then in *fine* granularity. Two-level trajectory partitioning ensures both high quality and high efficiency. The set of trajectory partitions is provided to the next phase.
(2) The *detection* phase: Outlying trajectory partitions are detected mainly using *distance*, so this phase is intuitive and efficient. Furthermore, to improve detection quality, *density* is also taken into account.

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

- We propose a *partition-and-detect* framework for trajectory outlier detection. This framework enables us to discover outlying *sub*-trajectories, whereas previous frameworks do not.
- We propose a trajectory outlier detection algorithm, which takes advantage of both distance-based and density-based approaches.
- We present a *two-level trajectory partitioning* strategy to speed up outlier detection. Many portions of trajectories can be pruned in a coarse-granularity level and do not need to be inspected further.
- We demonstrate, by using various real data sets, that our outlier detection algorithm effectively discovers outlying sub-trajectories from a trajectory database.

The rest of the paper is organized as follows. Section II discusses related work. Section III presents the problem statement. Section IV defines the concept of trajectory outliers and proposes our trajectory outlier detection algorithm. Section V proposes a two-level trajectory partitioning strategy. Section VI presents the results of experimental evaluation. Finally, Section VII concludes the study.

II. Related Work

A number of outlier detection algorithms have been developed for multi-dimensional *points*. These algorithms are classified mainly into four classes: distribution-based [2], distance-based [3], [4], [5], [6], density-based [7], [8], and deviation-based [9]. For our study, we briefly review only the distance-based and density-based approaches.

(1) **Distance-based approach:** This approach has been originally proposed by Knorr and Ng [3], [4], [5]. "An object O in a data set T is a $DB(p, D)$-*outlier* if at least fraction p of the objects in T lies greater than distance D from O." This approach depends on the overall or "global" distribution of a given set of data points. However, data are usually not uniformly distributed. Thus, this approach encounters difficulties when analyzing data with rather different density distributions.
(2) **Density-based approach:** This approach has been proposed by Breunig *et al.* [7]. An outlier is defined using the *local outlier factor (LOF)* of each object, which depends on the local density of its neighborhood. Here, the neighborhood is defined by the distance to the *MinPts*-th nearest neighbor. Data points with a high LOF value are detected as outliers. The LOF does not suffer from the problem above. However, the computation of LOF values requires a large number of k-nearest neighbor queries, and thus, can be computationally expensive [11].

As mentioned in Section I, Knorr *et al.* [5] have applied the distance-based algorithm to detecting trajectory outliers. Here, the distance is defined between two *whole* trajectories using their summary information. This technique is shown to successfully detect outliers if their directions, starting (or ending) points, or velocities are completely different from those of other trajectories. However, it is not clear whether this technique can detect outlying sub-trajectories from a set of very complicated trajectories. Our algorithm is more powerful than this technique since ours can detect outlying sub-trajectories as well as straightforward trajectory outliers that are detectable by this technique.

Li *et al.* [12] have proposed a trajectory outlier detection algorithm based on *classification*. In this algorithm, common patterns called *motifs* are extracted from trajectories, and the set of motifs forms a feature space in which the trajectories are placed. Through the transformation into a feature vector, the trajectories are fed into a classifier. This algorithm is inherently different from ours in that it depends on *training*. More specifically, a classification model is built using the training set, and a new trajectory is classified into either "normal" or "abnormal." In a real situation, it is not always easy to obtain a good training set. Notice that our algorithm does not require such training.

III. Problem Statement

We develop an efficient outlier detection algorithm based on the partition-and-detect framework. Given a set of *trajectories* $\mathcal{I} = \{TR_1, \cdots, TR_{num_{tra}}\}$, our algorithm discovers a set of *outliers* $\mathcal{O} = \{O_1, \cdots, O_{num_{out}}\}$ with *outlying trajectory partitions* for each outlier O_i, where the trajectory, outlier, and outlying trajectory partition are defined as follows.

A *trajectory* is a sequence of multi-dimensional points and is denoted as $TR_i = p_1 p_2 p_3 \cdots p_j \cdots p_{len_i}$ $(1 \leq i \leq num_{tra})$. Here, p_j $(1 \leq j \leq len_i)$ is a d-dimensional point. The length of a trajectory len_i 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.

An *outlier* is a trajectory that contains outlying 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. A trajectory partition is called a *t-partition* for short. A t-partition is *outlying* if it does not have "enough" similar neighbors (*i.e.*, close trajectories). The outlying t-partition is formally defined in Section IV-A.1.

Example 3: Figure 2 shows the overall procedure of trajectory outlier detection in the partition-and-detect framework. First, each trajectory is partitioned into a set of t-partitions. Second, outlying t-partitions, denoted by thick line segments, are identified based on the distance from neighboring trajectories. Notice that the distance measure also reflects the difference in shape. Then, a trajectory TR_3 with three outlying t-partitions is determined as an outlier. □

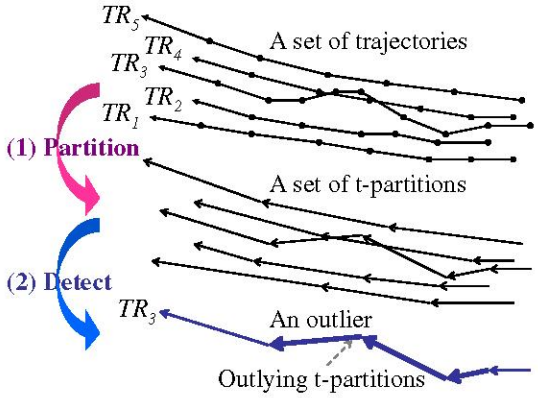


Fig. 2. An example of trajectory outlier detection in the partition-and-detect framework.

IV. Trajectory Outlier Detection

In this section, we define a trajectory outlier and propose our trajectory outlier detection algorithm. Section IV-A formally defines a trajectory outlier. Section IV-B discusses a trajectory partitioning strategy. Section IV-C presents a basic trajectory outlier detection algorithm. Section IV-D provides guidelines for determining parameter values.

A. Definition of Trajectory Outliers

1) Formalization Using the Distance-Based Outlier: A trajectory outlier is defined mainly using *distance*. More specifically, an outlying t-partition is identified based on the number of close trajectories, which is determined by the distance from neighboring trajectories. Before proceeding, we summarize the necessary notation in Table I.

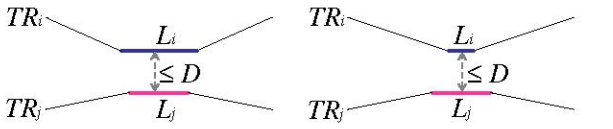
We first define a *close trajectory* in Definition 1. The concept of a close trajectory is described in Figure 3. This definition conforms to our intuition: unless a sufficient portion of a trajectory is close to a t-partition, the trajectory should not be regarded as close.

Definition 1: A trajectory TR_i is *close* to a t-partition $L_j \in P(TR_j)$ $(TR_i \neq TR_j)$ if $\sum_{L_i \in CP(TR_i, L_j, D)} len(L_i) \geq len(L_j)$. Here, D is a parameter given by a user.

TABLE I

The Notation for the Trajectory Outlier.[1]

Symbol	Definition
$len(L_i)$	The length of a t-partition L_i
$dist(L_i, L_j)$	The distance between L_i and L_j (See Section IV-A.3)
$P(TR_i)$	The set of all t-partitions of TR_i
$CP(TR_i, L_j, D)$	The set of TR_i's t-partitions within the distance D from $L_j \in P(TR_j)$ $(TR_i \neq TR_j)$, *i.e.*, $\{ L_i \mid L_i \in P(TR_i) \wedge dist(L_i, L_j) \leq D \}$
$CTR(L_i, D)$	The set of trajectories *close* to L_i
$OP(TR_i, D, p)$	The set of *outlying* t-partitions of TR_i



(a) TR_i is close to L_j. (b) TR_i is *not* close to L_j.

Fig. 3. The concept of the close trajectory.

We then define an *outlying t-partition* in Definition 2. This definition is adapted from the $DB(p, D)$-outlier [3], [4], [5] originally defined for points. Intuitively, a t-partition L_i is outlying if at least fraction p of the trajectories in $\mathcal{I}$ is not close to L_i.

Definition 2: A t-partition $L_i \in P(TR_i)$ is *outlying* if Ineq. (1) is true. $|\mathcal{I}|$ indicates the total number of trajectories. Here, p is a parameter given by a user.

$$| CTR(L_i, D) | \leq \lceil (1-p) | \mathcal{I} | \rceil \quad (1)$$

We now define an *outlier* in Definition 3. Intuitively, a trajectory becomes an outlier if the trajectory contains non-negligible (designated by F) outlying t-partitions. By this definition, a trajectory with just slight deviation is not included in the detection result.

Definition 3: A trajectory TR_i is an *outlier* if Ineq. (2) is true. Here, F is a parameter given by a user.

$$Ofrac(TR_i) = \frac{\sum_{L_i \in OP(TR_i, D, p)} len(L_i)}{\sum_{M_i \in P(TR_i)} len(M_i)} \geq F \quad (2)$$

2) Incorporation of Density: The definition in the previous section may lead to a problem when the data set has both dense and sparse regions. A t-partition in a dense region tends to have relatively a larger number of close trajectories than that in a sparse region. As a result, t-partitions in dense regions are favored over those in sparse regions, and thus, outlying t-partitions may not be even detected in dense regions.

To alleviate this problem, we incorporate density into trajectory outlier detection. We first define the *density* of a t-partition in Definition 4. Using the definition of the density,

[1]The naming convention is as follows: the prefix C means "Close," and O "Outlying"; the postfix P means a "t-Partition," and TR a "TRajectory."

we introduce the notion of the *adjusting coefficient* of a t-partition in Definition 5.

Definition 4: The *density* of a t-partition L_i is defined as Eq. (3), which is the number of t-partitions within the distance σ from L_i. Here, σ is the standard deviation of pairwise distances between t-partitions.

$$density(L_i) = \mid \bigcup_{TR_j \in \mathcal{I}} CP(TR_j, L_i, \sigma) \mid \tag{3}$$

Definition 5: The *adjusting coefficient* of a t-partition L_i is defined as Eq. (4), which is the ratio of the average density to the density of L_i.

$$adj(L_i) = \frac{\sum_{L_j \in \mathcal{L}} density(L_j) \ / \ \mid \mathcal{L} \mid}{density(L_i)}, \quad \text{where} \quad \mathcal{L} = \bigcup_{TR_k \in \mathcal{I}} P(TR_k) \tag{4}$$

The number of close trajectories $|CTR(L_i, D)|$ is multiplied by the adjusting coefficient $adj(L_i)$. After the adjustment, $|CTR(L_i, D)|$ is decreased in a dense region due to a low $adj(L_i)$ value (< 1.0), but it is increased in a sparse region due to a high $adj(L_i)$ value (> 1.0).

This scheme has a nice property that the density of a t-partition is not affected by the parameters D, p, and F. In contrast, the local reachability density [7] is affected by the parameter *MinPts*. This property allows us to precompute the densities of all t-partitions (unless the data set is updated). Thus, we are able to exploit the density almost for free during outlier detection.

3) Distance between T-Partitions: The distance between t-partitions (*i.e.*, line segments) is the main tool for trajectory outlier detection. Our distance function is composed of three components: (i) the *perpendicular distance* ($d_\perp$), (ii) the *parallel distance* ($d_\parallel$), and (iii) the *angle distance* (d_θ). They are adapted from similarity measures used in the area of pattern recognition [13] and are intuitively illustrated in Figure 4.

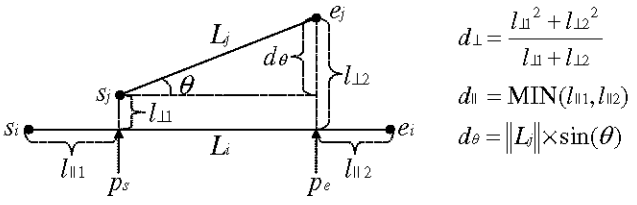


Fig. 4. Three components of the distance function for line segments.

We formally define the three components through Definitions 6∼8. Suppose there are two line segments $L_i = s_i e_i$ and $L_j = s_j e_j$. We assign a longer line segment to L_i and a shorter one to L_j without losing generality.

Definition 6: The *perpendicular distance* between L_i and L_j is defined as Eq. (5), 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.

$$d_\perp(L_i, L_j) = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}} \tag{5}$$

Definition 7: The *parallel distance* between L_i and L_j is defined as Eq. (6). 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.

$$d_\parallel(L_i, L_j) = \text{MIN}(l_{\parallel 1}, l_{\parallel 2}) \tag{6}$$

Definition 8: The *angle distance* between L_i and L_j is defined as Eq. (7). Here, $\|L_j\|$ is the length of L_j, and θ ($0° \leq \theta \leq 180°$) is the smaller intersecting angle between L_i and L_j.

$$d_\theta(L_i, L_j) = \begin{cases} \|L_j\| \times sin(\theta), & \text{if } 0° \leq \theta < 90° \\ \|L_j\|, & \text{if } 90° \leq \theta \leq 180° \end{cases} \tag{7}$$

We finally define the distance between two line segments as follows: $dist(L_i, L_j) = w_\perp \cdot d_\perp(L_i, L_j) + w_\parallel \cdot d_\parallel(L_i, L_j) + w_\theta \cdot d_\theta(L_i, L_j)$. The weights $w_\perp$, $w_\parallel$, and w_θ are determined depending on applications.

4) Two Types of Trajectory Outliers: The definition of a trajectory outlier should be neutral to applications. In other words, the definition needs to cover various kinds of applications. Not everyone has the same idea of what constitutes a trajectory outlier, and not all data sets conform to the same definitions or rules. Our definition can detect two types of trajectory outliers:

- **Positional outlier:** The *location* of a trajectory is different from those of neighboring trajectories.
- **Angular outlier:** The *direction* of a trajectory is different from those of neighboring trajectories.

An interesting observation is that we have a knob of emphasizing one of them. This is very useful to satisfy various requirements: some may consider that the positional difference is more critical in their applications, but others may consider that the angular difference is more critical. Our distance function is flexible enough to support both cases. For the former case, the weights $w_\perp$ or $w_\parallel$ need to be increased; for the latter case, the weight w_θ needs to be increased. Domain experts determine these weights based on visual inspection or background knowledge.

B. Discussion of Trajectory Partitioning

We now discuss the desiderata of trajectory partitioning in the partition-and-detect framework. In principle, any partitioning strategy, such as line simplification [14], can be exploited. However, careless partitioning (especially, in a long length) could miss possible outliers. Figure 5 shows a typical case, in which a long t-partition averages out the differences from neighboring trajectories. Notice that such a long t-partition

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

may be generated from a trajectory TR_{out} if the distance between each line segment and the t-partition is below a threshold. Even though TR_{out} behaves differently from its neighboring trajectories, these differences are completely ignored due to careless trajectory partitioning.

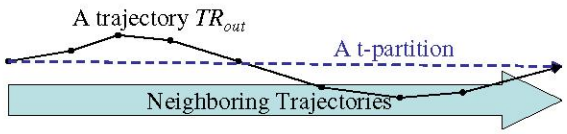


Fig. 5. An example of missing possible outliers.

To avoid missing possible outliers, we decide to begin with a simple strategy which partitions a trajectory *at a base unit*. A base unit is defined as the smallest meaningful unit of a trajectory in a given application, thus being application-dependent. Here, the interval of recording the locations of moving objects needs to be taken into account. If the recording interval is longer than the minimum interval of interest, the base unit can be every single point. Otherwise, it is preferable that the base unit should include multiple points. This simple strategy generally leads to high-quality detection results.

An immediate problem is poor performance since the number of t-partitions tends to proliferate due to fine-granularity partitioning. First, in Section IV-C, we present a basic version that uses the simple strategy. Then, in Section V, we address the performance issue.

C. Basic Algorithm

Figure 6 shows a basic version of our trajectory outlier detection algorithm TRAOD. This algorithm consists of two phases: *partitioning* and *detection*. In the partitioning phase, the algorithm partitions each trajectory at a base unit (lines 1∼2). In the detection phase, the algorithm finds outlying t-partitions (lines 3∼6) and detects trajectory outliers using the outlying t-partitions (lines 7∼9). Notice that $|CTR(L_i, D)|$ is multiplied by $adj(L_i)$ in line 5.

Time complexity: The time complexity of the basic version is $O(n_t^2)$, where n_t is the total number of t-partitions. To count $|CTR(L_i, D)|$ in line 4, the algorithm compares every pair of t-partitions.

TRAOD takes advantage of both distance-based and density-based approaches. TRAOD is very intuitive since it uses basically the distance-based approach, but it does not suffer from the local density problem since it takes account of the density as well. At the same time, the overhead of incorporating the density is very small since it can be precomputed beforehand.

D. Guideline for Parameter Values

Let us begin by stating there is no universally correct parameter value even for the same data set and application [5]. Some may want to find a larger number of outliers even though the deviation is small, but others may want to find only very few outliers that deviate significantly. Thus, our guideline resorts on user feedback. We believe this is sort of unavoidable because only users familiar with the data set and application at hand can determine whether the results returned are meaningful or not.

Algorithm **TRAOD** (TRAjectory Outlier Detection)

INPUT: A set of trajectories $\mathcal{I} = \{TR_1, \cdots, TR_{num_{tra}}\}$, three parameters: D, p, and F
OUTPUT: A set of outliers $\mathcal{O} = \{O_1, \cdots, O_{num_{out}}\}$ with outlying t-partitions of each O_i
ALGORITHM:
/* I. PARTITIONING PHASE */
01: **for each** $TR_i \in \mathcal{I}$ **do**
02: Partition TR_i at a base unit;
/* II. DETECTION PHASE */
/* $\mathcal{L}$ denotes the set of t-partitions */
03: **for each** $L_i \in \mathcal{L}$ **do**
/* Definition 1 */
04: Count $|CTR(L_i, D)|$ by computing $dist(L_i, L_j)$, /* $TR(L_i)$ means the trajectory enclosing L_i */ where $L_j \in \mathcal{L}$ and $TR(L_i) \neq TR(L_j)$;
/* Definition 2 */
05: **if** $\lceil |CTR(L_i, D)| \cdot adj(L_i) \rceil \leq \lceil (1-p)|\mathcal{I}| \rceil$ **then**
06: Mark L_i as outlying;
07: **for each** $TR_i \in \mathcal{I}$ **do**
/* Definition 3 */
08: **if** $Ofrac(TR_i) \geq F$ **then**
09: Output TR_i with its outlying t-partitions;

Fig. 6. The outlier detection algorithm TRAOD (basic).

Our algorithm TRAOD requires three parameters: D, p, and F. The most tricky and sensitive parameter is D. We suggest that users change the value of D and check the result repeatedly during the search for outliers. A larger value of D generates a smaller number of outliers, and a smaller value of D a larger number of outliers.

For the parameter p, it is reasonable to select a value of p very close to unity since an outlier occurs relatively infrequently. Our experience indicates that p should be closer to unity as the number of trajectories $|\mathcal{I}|$ increases. For example, $p = 0.95$ may suffice when $|\mathcal{I}| < 10^3$, but $p = 0.99$ may be more appropriate when $|\mathcal{I}| \approx 10^6$.

The parameter F represents the threshold of allowable noises that are not regarded as outliers. Our experience indicates that F should be smaller as the length of trajectories len_i gets longer. For example, $F = 0.2$ may suffice when $avg(len_i) < 100$, but $F = 0.1$ may be more appropriate when $avg(len_i) > 1000$.

V. TWO-LEVEL TRAJECTORY PARTITIONING

In this section, we propose a *two-level trajectory partitioning* strategy to speed up outlier detection. Given a trajectory

$TR_i = p_1 p_2 p_3 \cdots p_j \cdots p_{len_i}$, a *fine* t-partition is defined as a line segment $p_i p_{i+b}$ where b is a base unit, and a *coarse* t-partition as a line segment $p_i p_{i+j*b}$ $(j \geq 1)$. A coarse t-partition may enclose multiple fine t-partitions. Hereafter, to avoid confusion, a coarse t-partition is denoted as a capital letter (*e.g.*, L_i or L_j), and a fine t-partition as a small letter (*e.g.*, l_i or l_j).

At the first level, each trajectory is partitioned into coarse t-partitions (Section V-A). At the second level, coarse t-partitions that are likely to be outlying are selected, and then, only selected ones are partitioned into fine t-partitions (Section V-B). In this way, we narrow the search space that needs to be inspected in fine granularity (*i.e.*, by partitioning a trajectory at a base unit). As a result, many portions of trajectories can be *pruned* early on.

To identify outlying coarse t-partitions, we derive the distance bounds between two coarse t-partitions L_i and L_j: $lb(L_i, L_j, dist)$ and $ub(L_i, L_j, dist)$. Here, $lb(L_i, L_j, f)$ and $ub(L_i, L_j, f)$ denote the minimum and maximum of $f(l_i, l_j)$, where l_i is a fine t-partition in L_i, and l_j that in L_j. These bounds are derived in Section V-B.

A. Coarse Granularity Partitioning

1) Desirable Properties: Coarse-granularity partitioning should possess two desirable properties: *preciseness* and *conciseness*. Preciseness means that the difference between a trajectory and a set of its coarse t-partitions should be as small as possible; it is required for making the bounds tight. The tighter the bounds are, the higher the pruning power becomes. Conciseness means that the number of coarse t-partitions should be as small as possible; it is required for reducing the number of comparisons between coarse t-partitions.

Preciseness and conciseness are contradictory to each other. Hence, we need to find an optimal tradeoff between the two properties.

2) Formalization Using the MDL Principle: We adopt our trajectory partitioning method originally proposed for trajectory clustering [15]. This method relies on the minimum description length (MDL) principle.

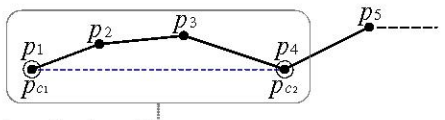
The MDL cost consists of two components: $L(H)$ and $L(D|H)$. Here, H means the hypothesis, and D the data. The two components are informally stated as follows [16]: "$L(H)$ is the length, in bits, of the description of the hypothesis; and $L(D|H)$ is the length, in bits, of the description of the data 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 $L(D|H)$.

The MDL principle fits very well our problem. A set of coarse t-partitions corresponds to H, and a trajectory corresponds to D. Most importantly, $L(H)$ measures conciseness, and $L(D|H)$ preciseness. Thus, finding the optimal coarse-granularity partitioning translates to finding the best hypothesis using the MDL principle. One advantage of this method is that it does not require any additional parameters as opposed to line simplification [14].

Figure 7 shows our formulation of $L(H)$ and $L(D|H)$. We formulate $L(H)$ by Eq. (8).[3] $L(H)$ represents the sum of the length of a coarse t-partition. On the other hand, we formulate $L(D|H)$ by Eq. (9). $L(D|H)$ represents the sum of the difference between a trajectory and a coarse t-partition. For each coarse t-partition $p_{c_j} p_{c_{j+1}}$, we add up the difference between $p_{c_j} p_{c_{j+1}}$ and $p_k p_{k+1}$ $(c_j \leq k \leq c_{j+1}-1)$. To measure the difference, the sum of $d_\perp$ and d_θ is used, but $d_\parallel$ is not since a trajectory always encloses its coarse t-partitions.

$$L(H) = \sum_{j=1}^{par_i - 1} \log_2(len(p_{c_j} p_{c_{j+1}})) \tag{8}$$

$$L(D|H) = \sum_{j=1}^{par_i - 1} \sum_{k=c_j}^{c_{j+1}-1} \{ \log_2(d_\perp(p_{c_j} p_{c_{j+1}}, p_k p_{k+1})) + \log_2(d_\theta(p_{c_j} p_{c_{j+1}}, p_k p_{k+1})) \} \tag{9}$$



$$L(H) = \log_2(len(p_1 p_4))$$
$$L(D|H) = \log_2(d_\perp(p_1 p_4, p_1 p_2) + d_\perp(p_1 p_4, p_2 p_3) + d_\perp(p_1 p_4, p_3 p_4)) + \log_2(d_\theta(p_1 p_4, p_1 p_2) + d_\theta(p_1 p_4, p_2 p_3) + d_\theta(p_1 p_4, p_3 p_4))$$

Fig. 7. Formulation of the MDL cost.

We find the optimal coarse-granularity partitioning that minimizes $L(H) + L(D|H)$. This is exactly the tradeoff between preciseness and conciseness. For the sake of efficiency, we use an $O(n)$ greedy algorithm [15].

B. Fine Granularity Partitioning

Two kinds of information in Table II is used for deriving the lower and upper bounds. The first three values are collected during coarse-granularity partitioning and are maintained per coarse t-partition. The next two values are computed when comparing two coarse t-partitions.

TABLE II
THE NOTATION FOR THE DISTANCE BOUNDS.

SYMBOL	DEFINITION
$maxl_\perp(L_i)$	The maximum $l_\perp$ between a coarse t-partition L_i and its fine t-partitions
$minlen(L_i)$ $maxlen(L_i)$	The minimum (or maximum) length of fine t-partitions in L_i
$max\theta(L_i)$	The maximum angle between a coarse t-partition L_i and its fine t-partitions
$l_{\perp 1}$, $l_{\perp 2}$	The Euclidean distance from an endpoint to a projection point (See Definition 6)
θ	The angle between two coarse t-partitions L_i and L_j (See Definition 8)

Due to complicatedness of our distance function, we make the following simplification: fine t-partitions are made to be

[3]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 detect outlying t-partitions according to their *relative distances*. Second, a very important reason not to use endpoints is to make the outlier detection result *not influenced by the coordinate values of line segments*.

parallel to their coarse t-partition. This simplification is applied to only deriving the bounds for $d_\perp$ and $d_\parallel$. It is expected not to induce large errors since the angle between a coarse t-partition and its fine t-partitions is usually small.

We derive the lower and upper bounds separately for $d_\perp$, $d_\parallel$, and d_θ in Lemmas 1∼3. These lemmas are used to derive the lower and upper bounds for our distance function $dist(L_i, L_j)$ in Lemma 4.

Lemma 1: The lower and upper bounds for $d_\perp(L_i, L_j)$ are formulated by Eqs. (10) and (11), respectively.

$$lb(L_i, L_j, d_\perp) = \text{MIN}(l_{\perp 1}, l_{\perp 2}) - (maxl_\perp(L_i) + maxl_\perp(L_j)) \quad (10)$$

$$ub(L_i, L_j, d_\perp) = \text{MAX}(l_{\perp 1}, l_{\perp 2}) + (maxl_\perp(L_i) + maxl_\perp(L_j)) \quad (11)$$

PROOF: The proof involves a series of geometric calculations. See Appendix. □

Lemma 2: The lower and upper bounds for $d_\parallel(L_i, L_j)$ are formulated by Eqs. (12) and (13), respectively. Here, three cases are determined depending on whether both, one, or none of the projection points in Figure 4 are within L_i.

$$lb(L_i, L_j, d_\parallel) = \begin{cases} 0 & \text{if enclose or overlap} \\ d_\parallel(L_i, L_j) & \text{if disjoint} \end{cases} \quad (12)$$

$$ub(L_i, L_j, d_\parallel) = \begin{cases} \text{MAX}(len(L_i), len(L_j)) & \text{if enclose} \\ len(L_i) + len(L_j) - d_\parallel(L_i, L_j) & \text{if overlap} \\ len(L_i) + len(L_j) + d_\parallel(L_i, L_j) & \text{if disjoint} \end{cases} \quad (13)$$

PROOF: Straightforward. Omitted due to lack of space. □

Lemma 3: The lower and upper bounds for $d_\theta(L_i, L_j)$ are formulated by Eqs. (14) and (15), respectively. If the argument of the sine function is $< 0°$ (or $> 90°$), it is considered to be $0°$ (or $90°$).

$$lb(L_i, L_j, d_\theta) = \text{MIN}(minlen(L_i), minlen(L_j)) \times sin(\theta - maxθ(L_i) - maxθ(L_j)) \quad (14)$$

$$ub(L_i, L_j, d_\theta) = \text{MIN}(maxlen(L_i), maxlen(L_j)) \times sin(\theta + maxθ(L_i) + maxθ(L_j)) \quad (15)$$

PROOF: Straightforward. Omitted due to lack of space. □

Lemma 4: $lb(L_i, L_j, dist)$ is the weighted sum of Eqs. (10), (12), and (14), where the weights are $w_\perp$, $w_\parallel$, and w_θ, respectively. Similarly, $ub(L_i, L_j, dist)$ is the weighted sum of Eqs. (11), (13), and (15).

PROOF: The $lb()$'s and $ub()$'s in Lemmas 1∼3 represent the possible minimum and maximum values for each component of $dist(L_i, L_j)$. Therefore, their weighted sums naturally formulate the possible minimum and maximum values of $dist(L_i, L_j)$. □

We finally present two rules for fine-granularity partitioning as below. If $lb(L_i, L_j, dist) > D$, *none* of the fine t-partitions in L_i can contribute to the number of close trajectories of any fine t-partitions in L_j, and vice versa. On the other hand, if $ub(L_i, L_j, dist) \leq D$, *all* of the fine t-partitions in L_i can contribute to the number of close trajectories of any fine t-partitions in L_j, and vice versa.

- **Rule 1:** If $lb(L_i, L_j, dist) > D$, fine-granularity partitioning is not required when comparing the coarse t-partitions L_i and L_j.
- **Rule 2:** If $ub(L_i, L_j, dist) \leq D$, fine-granularity partitioning is required, but the distance between the fine t-partitions in L_i and L_j needs not be computed.

C. Performance-Enhanced Algorithm

Figure 8 shows a performance-enhanced version of TRAOD. The partitioning phase becomes more sophisticated. Each trajectory is first partitioned in coarse granularity (lines 1∼2). If a pair of coarse t-partitions triggers Rule 1, the pair is not inspected further (lines 4∼5). Otherwise, two coarse t-partitions in the pair are partitioned in fine granularity (lines 6∼7); unless the pair of coarse t-partitions triggers Rule 2, the distance between fine t-partitions is checked, and then, a close pair is stored in the lists $CL(l_i)$ and $CL(l_j)$ (lines 10∼13). The detection phase is the same as that of the basic version, except that the number of close trajectories $|CTR(l_i, D)|$ is counted using the list $CL(l_i)$ instead of comparing every pair of fine t-partitions.

Time complexity: The time complexity of the enhanced version is $O(n_c^2 + n_f^2)$, where n_c is the number of coarse t-partitions, and n_f is that of fine ones. Notice that $n_c \ll n_t$ and $n_f \ll n_t$ compared with the basic version. This is exactly the reason why the performance gain is achieved in the enhanced version.

VI. Experimental Evaluation

A. 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 2006. This data set has 608 trajectories and 18951 points. A small portion (1990∼2006) of the data set, which has 221 trajectories and 7270 points, is also used.

The animal movement data set has been generated by the Starkey project. This data set contains the radio-telemetry

[4] http://weather.unisys.com/hurricane/atlantic/

[5] http://www.fs.fed.us/pnw/starkey/data/tables/

Algorithm **TRAOD** (TRAjectory Outlier Detection)

INPUT: A set of trajectories $\mathcal{I} = \{TR_1, \cdots, TR_{num_{tra}}\}$, three parameters: D, p, and F
OUTPUT: A set of outliers $\mathcal{O} = \{O_1, \cdots, O_{num_{out}}\}$ with outlying fine t-partitions of each O_i
ALGORITHM:

```
    /* I. PARTITIONING PHASE */
01: for each TR_i ∈ I do
02:     Partition TR_i in coarse granularity;
    /* C denotes the set of coarse t-partitions */
03: for each pair of L_i ∈ C and L_j ∈ C (L_i ≠ L_j) do
04:     if lb(L_i, L_j, dist) > D then   /* RULE 1 */
05:         Continue to the next pair;   /* Do nothing */
06:     else
07:         Partition L_i and L_j in fine granularity;
08:     if ub(L_i, L_j, dist) ≤ D then   /* RULE 2 */
            /* CL(l_i) holds the t-partitions close to l_i */
09:         ∀l_i ∈ L_i and ∀l_j ∈ L_j,
            Insert l_i into CL(l_j) and l_j into CL(l_i);
10:     else
            /* Every pair of fine t-partitions is compared */
11:         for each pair of l_i ∈ L_i and l_j ∈ L_j do
12:             if dist(l_i, l_j) ≤ D then
13:                 Insert l_i into CL(l_j) and l_j into CL(l_i);
    /* II. DETECTION PHASE */
    /* F denotes the set of fine t-partitions */
14: for each l_i ∈ F do
15:     Count |CTR(l_i, D)| by using CL(l_i);
16:     if ⌈|CTR(l_i, D)| · adj(l_i)⌉ ≤ ⌈(1 − p)|I|⌉ then
17:         Mark l_i as outlying;
18: for each TR_i ∈ I do
19:     if Ofrac(TR_i) ≥ F then
20:         Output TR_i with its outlying fine t-partitions;
```

Fig. 8. The outlier detection algorithm TRAOD (enhanced).

locations (with other information) of elk, deer, and cattle from the years 1993 through 1996. The locations are recorded at 30-minute intervals. We extract the x and y coordinates from the telemetry data for experiments. We use elk's movements in June 1993 (*Elk1993*), deer's movements in 1995 (*Deer1995*), and cattle's movements in 1993 (*Cattle1993*). Elk1993 has 33 trajectories and 15422 points; Deer1995 32 trajectories and 20065 points; Cattle1993 41 trajectories and 19556 points.

We put more weights on the angular outliers in experiments. For the hurricane track data set, w_θ is set to be five times larger than $w_\perp$ or $w_\parallel$. For the animal movement data set, w_θ is set to be two times larger than $w_\perp$ or $w_\parallel$.

We conduct all 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. All images in this paper are obtained by executing the enhanced version of TRAOD. [6]

B. Results for Hurricane Track Data

Figure 9 shows the result for a small portion of the *Hurricane* data set. The parameters are set as follows: $D = 85$, $p = 0.95$, and $F = 0.2$. Here, thick red lines represent outlying t-partitions, thin red lines trajectory outliers with outlying t-partitions, and thin green lines normal trajectories. We observe that thirteen trajectory outliers are detected. Notice that thin red lines are not outlying by themselves, but thick red lines are. We can easily see that outlying t-partitions appear if their directions are significantly different from those of neighboring trajectories or if they have very few neighboring trajectories. In Figure 9, the outlying t-partitions in the middle region are moving to totally different directions, and the ones in the right region have almost no neighboring trajectory.

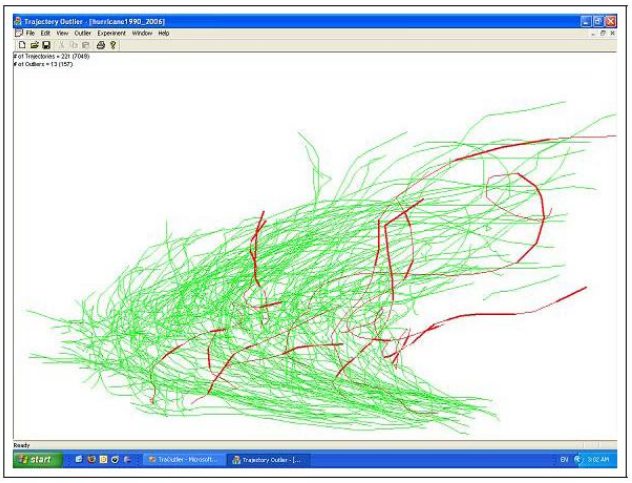


Fig. 9. Trajectory outliers for Hurricane (small).

C. Results for Animal Movement Data

Figure 10 shows the result for the *Elk1993* data set. The parameters are set as follows: $D = 55$, $p = 0.95$, and $F = 0.1$. We observe that three trajectory outliers are detected. The outlying t-partitions are shown to be located in very sparse regions, especially at the protrusions of the movement pattern: the lower-left, lower-right, and upper-right regions. Hence, this result looks reasonable.

Figure 11 shows the result for the *Deer1995* data set. [7] The parameters are set as follows: $D = 80$, $p = 0.95$, and $F = 0.1$. We observe that three trajectory outliers are detected. The outlying t-partitions are shown to be concentrated in a few regions. It turns out that just *one or two* deer have moved around these regions in a random fashion. Hence, this result is verified to be correct. In addition, it is interesting that outliers are detected only in the regions where clusters have not been identified; *i.e.*, this result is almost the complementary set of the clustering result shown by Lee *et al.* [15].

[6] We encourage the interested reader to visit "http://netfiles.uiuc.edu/jaegil/www/icde08" to view large images in this paper.

[7] One might wonder why there are somewhat straight and long line segments. Their endpoints are recorded with a longer interval due to some missing signals. Such line segments are ignored in detecting outlying t-partitions.

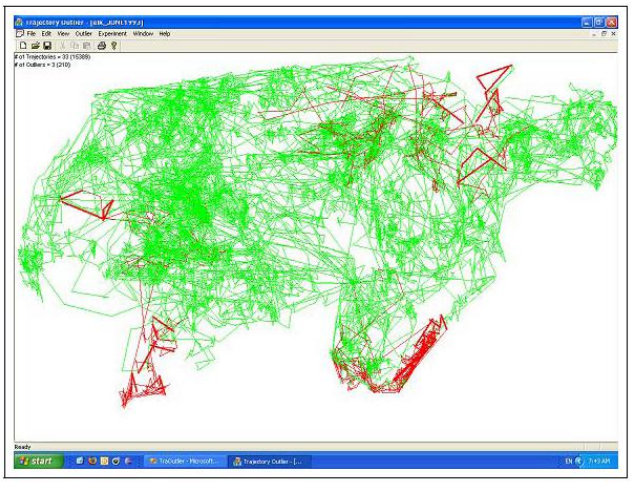


Fig. 10. Trajectory outliers for Elk1993.

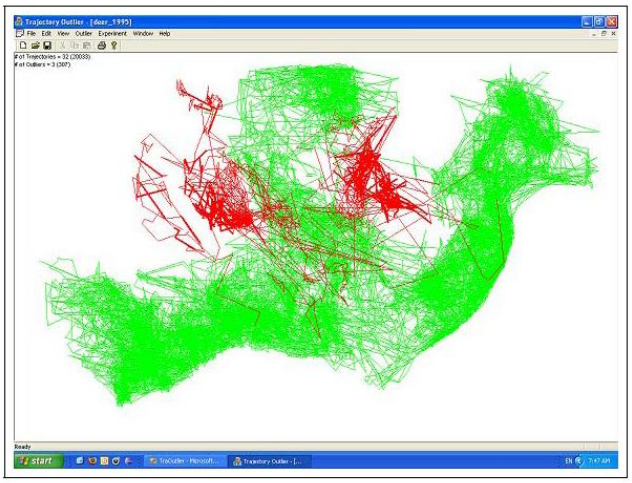


Fig. 11. Trajectory outliers for Deer1995.

D. Efficiency and Accuracy of Pruning

Figure 12 shows the pruning power of the two-level partitioning strategy. Here, *2L-Total* is the ratio of the number of pairs pruned by Rule 1 to the total number of pairs of coarse t-partitions. *2L-False* is the proportion of pairs pruned incorrectly. *Optimal* is obtained by actually calculating the exact number of pairs that can be pruned. This result indicates that a large proportion (64∼88%) of comparisons between coarse t-partitions can be pruned off, and the proportion of incorrect pruning is absolutely negligible (0.1∼1.8%). We note that incorrect pruning is caused by the simplification in Section V-B. More importantly, the bounds derived by Lemma 4 are sufficiently tight in the sense that *2L-Total* is as close as 76∼93% of *Optimal*.

Figure 13 shows the speedup ratio of the two-level partitioning strategy. Here, the speedup ratio is defined as the ratio of the wall clock time of the basic version to that of the performance-enhanced version. Performance is shown to be improved by 27∼74 times. This result indeed demonstrates

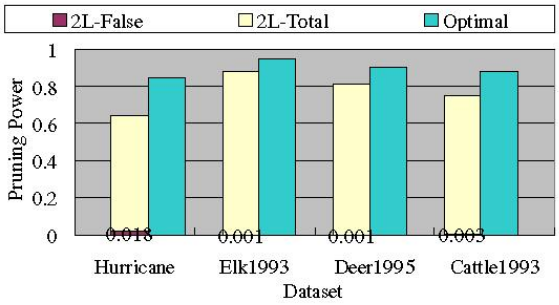


Fig. 12. Pruning power of two-level partitioning.

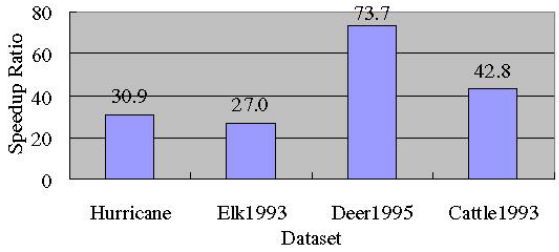


Fig. 13. Speedup ratio of two-level partitioning.

the effectiveness of the two-level partitioning strategy. The speedup ratio becomes higher as the size of the data set increases or as the pruning power gets higher.

E. Effects of Parameter Values

Figure 14 compares the results for the *Hurricane (small)* data set when using different values of the parameter D. Compared with Figure 9, a larger number (19) of trajectory outliers are detected in Figure 14 (a), whereas a smaller number (10) of them are detected in Figure 14 (b). Less-outlying trajectories can be detected using a smaller value of D. As discussed in Section IV-D, the correct parameter value is heavily dependent on the users' purpose: some prefer the former result, but others the latter. We expect that many users have a desired number of trajectory outliers in their minds, so they can adjust the parameter value to achieve it.

Table III shows the number of trajectory outliers (as well as the number of outlying t-partitions) for *Deer1995* while varying the values of the parameters D and p. The number of trajectory outliers increases as the value of D decreases or as the value of p decreases. While varying the value of p, the number of trajectory outliers is varied in a stair-like fashion according to $\lceil (1-p)|\mathcal{I}| \rceil$.

TABLE III
THE NUMBER OF TRAJECTORY OUTLIERS FOR DEER1995 DEPENDING ON THE PARAMETER VALUES.

D ($p = 0.95$)	65	70	75	80	85	90	95
# of trajectories	4	3	3	3	2	1	1
# of t-partitions	662	546	404	307	228	146	87
p ($D = 80$)	0.93	0.94	0.95	0.96	0.97	0.98	0.99
# of trajectories	3	3	3	3	2	2	2
# of t-partitions	307	307	307	307	84	84	84

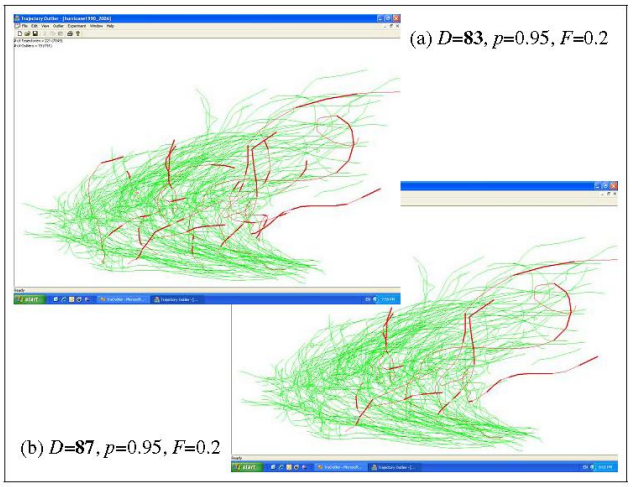


Fig. 14. Comparison of trajectory outliers for Hurricane (small) when using different parameter values.

VII. Conclusions

In this paper, we have proposed a novel framework, the *partition-and-detect* framework, for trajectory outlier detection. Based on this framework, we have developed a trajectory outlier detection algorithm, TRAOD. The main advantage of TRAOD is the detection of outlying sub-trajectories from a trajectory database. The visual inspection results show that TRAOD effectively detects trajectory outliers with outlying t-partitions. Overall, we believe that we have provided a new paradigm in trajectory outlier detection.

This work is just the first step, and there are many challenging issues. First, we consider here only the spatial information. In applications, temporal information is often associated with spatial information. Since the movement pattern of an object is closely related to its speed, more interesting results can be obtained if we exploit this temporal information as well. Second, we are using a hybrid of the distance-based and density-based approaches. It is worthwhile to compare the result of our approach with that of the purely density-based approach. We are currently investigating into the detailed issues as a further study.

Acknowledgement

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

References

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

[2] V. Barnett and T. Lewis, *Outliers in Statistical Data*. John Wiley & Sons, 1994.

[3] E. M. Knorr and R. T. Ng, "Algorithms for mining distance-based outliers in large datasets," in *Proc. 24th Int'l Conf. on Very Large Data Bases*, New York City, New York, Aug. 1998, pp. 392–403.

[4] ——, "Finding intensional knowledge of distance-based outliers," in *Proc. 25th Int'l Conf. on Very Large Data Bases*, Edinburgh, Scotland, Sept. 1999, pp. 211–222.

[5] E. M. Knorr, R. T. Ng, and V. Tucakov, "Distance-based outliers: Algorithms and applications," *VLDB Journal*, vol. 8, no. 3, pp. 237–253, Feb. 2000.

[6] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *Proc. 2000 ACM SIGMOD Int'l Conf. on Management of Data*, Dallas, Texas, May 2000, pp. 427–438.

[7] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. 2000 ACM SIGMOD Int'l Conf. on Management of Data*, Dallas, Texas, May 2000, pp. 93–104.

[8] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos, "LOCI: Fast outlier detection using the local correlation integral," in *Proc. 19th Int'l Conf. on Data Engineering*, Bangalore, India, Mar. 2003, pp. 315–326.

[9] C. C. Aggarwal and P. S. Yu, "Outlier detection for high dimensional data," in *Proc. 2001 ACM SIGMOD Int'l Conf. on Management of Data*, Santa Barbara, California, May 2001, pp. 37–46.

[10] L. E. Carr and R. L. Elsberry, "Monsoonal interactions leading to sudden tropical cyclone track changes," *Monthly Weather Review*, vol. 123, no. 2, pp. 265–290, Feb. 1995.

[11] W. Jin, A. K. H. Tung, and J. Han, "Mining top-n local outliers in large databases," in *Proc. 7th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, San Francisco, California, Aug. 2001, pp. 293–298.

[12] X. Li, J. Han, S. Kim, and H. Gonzalez, "ROAM: Rule- and motif-based anomaly detection in massive moving object data sets," in *Proc. 7th SIAM Int'l Conf. on Data Mining*, Minneapolis, Minnesota, Apr. 2007.

[13] J. Chen, M. K. H. Leung, and Y. Gao, "Noisy logo recognition using line segment hausdorff distance," *Pattern Recognition*, vol. 36, no. 4, pp. 943–955, Apr. 2003.

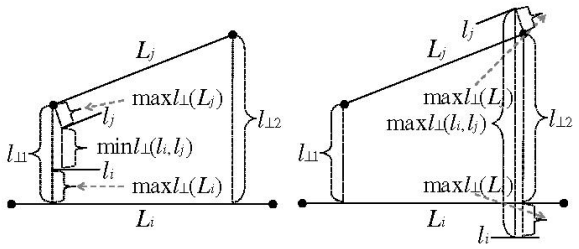
[14] H. Cao, O. Wolfson, and G. Trajcevski, "Spatio-temporal data reduction with deterministic error bounds," *VLDB Journal*, vol. 15, no. 3, pp. 211–228, Sept. 2006.

[15] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: A partition-and-group framework," in *Proc. 2007 ACM SIGMOD Int'l Conf. on Management of Data*, Beijing, China, June 2007, pp. 593–604, The extended version is available from http://www.cs.uiuc.edu/research/techreports.php?report=UIUCDCS-R-2007-2828.

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

Appendix: Proof of Lemma 1

Let l_i and l_j be any fine t-partitions in L_i and L_j, respectively. $d_\perp(l_i, l_j)$ denotes the perpendicular distance between l_i and l_j. First, the minimum of $l_\perp(l_i, l_j)$ is achieved as in Figure 15 (a). In this case, Eq. (10) $\leq minl_\perp(l_i, l_j) \leq d_\perp(l_i, l_j)$. Second, the maximum of $l_\perp(l_i, l_j)$ is achieved as in Figure 15 (b). In this case, $d_\perp(l_i, l_j) \leq maxl_\perp(l_i, l_j) \leq$ Eq. (11).



(a) Lower bound. (b) Upper bound.

Fig. 15. Derivation of the bounds for $d_\perp$.