Delft University of Technology
Master's Thesis in Embedded Systems

Outlier Detection for Pedestrian Movement

Yujing Gong

TU Delft Delft University of Technology

Outlier Detection for Pedestrian Movement

Master's Thesis in Embedded Systems

Embedded Software Section
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Yujing Gong
Y.Gong-1@student.tudelft.nl

August 14th, 2017

Author
Yujing Gong (Y.Gong-1@student.tudelft.nl)
Title
Outlier Detection for Pedestrian Movement
MSc presentation
August 25th, 2017

Graduation Committee

prof. dr. K. G. Langendoen (chair)	Delft University of Technology
dr. M. A. Zuniga Zamalloa	Delft University of Technology
dr. J. Pouwelse	Delft University of Technology

Daily Advisor

Stef Janssen	Delft University of Technology

Abstract

Fast development of tracking devices has made trajectory outlier detection (TOD) possible and meaningful. Given a set of trajectories T, a TOD algorithm outputs a subset of T, of which trajectories are different from most of the other trajectories in some aspect(s). These trajectories, namely outliers, can indicate important or interesting information and are thus worth noticing. TOD techniques can be used for surveillance security, accident discovery, and many other purposes.

Many of the existing TOD algorithms consider only spatial trajectory outliers. They can detect trajectories that follow an abnormal route or direction. While some existing algorithms are capable of detecting outliers in temporal aspects, like trajectories with abnormal time duration or speed, they have their own weaknesses. For example, they can be computationally expensive, or fail to detect important types of outliers. In this work, we aim to overcome these shortcomings of previous TOD algorithms.

A novel grid-based TOD algorithm is proposed that is capable of detecting temporal-spatial outliers including density, direction, duration, and speed outliers with accuracy as well as fast calculation. The algorithm performs the following three main steps: (i) it calculates density, direction, duration, and speed features of all trajectories in the input set T; (ii) it transforms feature information of trajectories into grid information; (iii) it examines each trajectory grid cell by grid cell. Following these three steps, outlying trajectories are extracted. By conducting experiments on several data sets including both simulated and real ones, the algorithm is shown to be efficient in detecting density, direction, duration, and speed outliers. It outperforms state-of-the-art TOD algorithms in various aspects.

Preface

When I was searching for a topic for my master thesis, I found the cross-faculty project Smart Sensing for Aviation. One aspect of the project is tracking and understanding passenger movements, which aroused my interest - it is a very challenging topic and quite close to real life. Security is of great importance in public places and should always be taken seriously. Obviously, the entire security problem cannot be solved at once. However, we can improve security by detecting problems caused by "abnormal" passengers. Analyzing passenger movements in a public place can not only benefit security, but also bring us other interesting information. Then my research topic was initially proposed: we will do trajectory outlier detection for pedestrian movement, with a special focus on the outliers that are potential security threats for public places.

My master study is ending soon. Here I would like to thank all that helped and supported me. My deepest gratitude goes to my daily advisor, Stef Janssen, for his patience and kindness. He helped me a lot with my research, and writing and presenting skills. Without that, I would not finish my thesis successfully. I also thank my supervisor Koen Langendoen. During our monthly meetings, he often provided with me valuable advice. Thanks also go to Ioannis for his support of some experiments we did, to Marco for his advice during my mid-term presentation, and many other people on "the 9th floor". It was an unforgettable experience doing my master thesis in the Embedded Software group.

I would also thank my family, especially my parents. Were it not for their unconditional support and love, I would not have the opportunity to study in The Netherlands in the first place. Lastly, I also appreciate my friends for their company. Their encouragement and comforts are like a bright light when I am facing difficulties.

Yujing Gong

Delft, The Netherlands
August 4th, 2017

Contents

Chapter 1

Introduction

The increasing availability and development of localization and tracking technologies such as GPS, Wi-Fi, and video cameras make it convenient to generate a myriad of trajectories. Such trajectories can represent the movement of various moving objects like people, vehicles, vessels and animals. It offers researchers the opportunity to do trajectory mining and understand behaviors of moving objects. One area of trajectory mining is *outlier detection* [22], which is to detect trajectories that are different, rare, or possess other unique properties. Outlier detection has been identified as an important technique for detecting critical events in a wide range of data-rich domains where a majority of the data is considered normal and uninteresting [1, 6]. This problem in the trajectory-mining area has been studied for years and can be used in various applications. *Trajectory outlier detection* (TOD) algorithms and models that are applicable in vessel surveillance [7, 12, 13], hurricane tracking [8, 15, 20], animal migratory study [8, 15], taxi fraud detection [2, 14, 21], traffic violation [4, 17], and many other areas have been proposed.

In this thesis, the automated detection of anomalous pedestrian trajectories is central. A study is made on how to develop an efficient method for automated detection of anomalous pedestrian trajectories.

1.1 Problem Statement

In public places like shopping malls, train stations, and airport terminals, large numbers of people move around. Some of these people may behave abnormally due to various reasons: they might be in trouble and need help or they intent to do harm in some way. Under such circumstances, it will benefit both managers and visitors if these abnormal trajectories can be noticed. Since some people with abnormal behaviors will have abnormal trajectories as well, trajectory outlier detection is a way to detect abnormal behaviors

and improve security. However, it is infeasible for a small number of security officers to identify these abnormal trajectories by manually monitoring. This brings the need for automated trajectory anomaly detection.

Different models and algorithms have previously been proposed for trajectory outlier detection. Yet, we argue that these algorithms are not capable of detecting all types of important outliers and suffer from some drawbacks. Some of them take entire trajectories as detecting objects and fail to detect outliers when only small parts of a trajectory are outlying. In fact, since some abnormal behaviors only happen at small parts of an entire trajectory and disappear at the other parts, it is critical to do outlier detection for parts of trajectories. Some algorithms fail to detect duration and speed outliers because they focus much on the spatial domain, like density and direction outliers. As a matter of fact, duration and speed outliers are of great importance because a person who spends an uncommonly long time at some place might be planning something harmful, and a person who runs while the others are walking calmly might have done something bad and is running away. Some algorithms require large memory storage. Yet it is better to have an algorithm with small memory consumption. And lastly, some algorithms are computationally expensive although they are capable of detecting duration and speed outliers. However, it is important to have fast running algorithms because security is time sensitive and requires a quick response.

Hence, the problem of this project is stated as follows:

> Develop a TOD algorithm that is capable of detecting density, direction, duration, and speed outliers with high accuracy as well as fast calculation and small memory consumption. The algorithm should be feasible when only small parts of a trajectory are outlying.

To fulfill this statement, the following sub-questions are defined:

- How well can previous algorithms detect each of these outlier types?
- How can we improve that?

Accordingly, the main goal of this project is to study existing TOD methods and propose new or updated algorithms that outperform existing algorithms in certain aspects. The focus will be on detecting duration and speed outliers, which previous algorithms do badly. While doing so, run-time complexity and memory consumption are taken into account.

1.2 Contributions

During this research, the following contributions were made:

- A novel algorithm is proposed that is able to detect four types of temporal-spatial outliers: density, direction, duration, and speed outliers, with high accuracy as well as fast calculation and small memory usage.
- A precise trajectory-segmentation approach is designed as a pre-process of trajectory data. This can be used for further study on grid-based TOD methods.
- The algorithm is gridbased, making it easy to detect all the four types of outliers when only small parts of a trajectory are outlying.
- The algorithm is shown to be more efficient than state-of-the-art Dynamic Time Warping [18].
- The TOD algorithm has time complexity $O(num_{traj} \cdot N_{max})$, with num_{traj} being the number of trajectories and N_{max} being the maximum number of location points of a trajectory. This is a great advantage compared to the previous algorithms that are capable of detecting all the four types of outliers, which typically have $O(num_{traj}^2 \cdot N_{max})$ or $O(num_{traj} \cdot N_{max}^2)$ complexity.

1.3 Thesis Organization

In Chapter 2 some definitions related to trajectory outlier detection are given. In Chapter 3 some previously existing TOD algorithms are reviewed and compared. Chapter 4 gives introductions of two TOD algorithm designs. In Chapter 5 the experiments we did are introduced, together with the results and analysis of those experiments. Lastly in Chapter 6 conclusions of this work are made, as well as a proposal for future work.

Chapter 2

Definitions

In this chapter, several definitions that are closely relevant to trajectory outlier detection are made. In Section 2.1, the definitions of *trajectory* and *trajectory segment* are given. In Section 2.2, a grid-based method and a trajectory-interpolation approach are introduced. Section 2.3 gives some definitions that are relevant to the relationship between trajectories and grid cells. Section 2.4 introduces some important features of trajectories. And lastly, in Section 2.5 different types of outliers are defined.

2.1 Trajectory

The definitions of *trajectory* and *trajectory segment* are given as follows.

Definition 1 (trajectory): A *trajectory* is defined as an ordered sequence of location points, i.e., *traj*: $q_1 q_2 \dots q_M$. Each location point q_m ($m \in [1, M]$) is represented by a triple (x_m, y_m, t_m), where x_m, y_m, and t_m are the x coordinate, the y coordinate and the time stamp, respectively, of trajectory *traj* at location point q_m. When $a < b$, the inequation $t_a < t_b$ should be satisfied.

Definition 2 (trajectory segment): We assume that there is a line segment between every two consecutive trajectory location points. These line segments are called *trajectory segments* and they indicate the path an object moves along with. If a trajectory segment is connected by two location points q_m and q_{m+1}, it can be referred to as $trajSeg_m : q_m q_{m+1}$ and q_m is its *staring point*.

The trajectory segment defined here has the following attributes.

- The *length* of a trajectory segment $trajSeg_m$ is represented as len_m and can be calculated as $len_m = \sqrt{(x_{m+1} - x_m)^2 + (y_{m+1} - y_m)^2}$.
- If a trajectory has M location points. The number of trajectory segments in it is $M - 1$.

2.2 Grid and Interpolation

In this research, we aim to make trajectory data processing more convenient by applying a grid-based method, which is introduced as follows.

The minimum bounding box of all trajectories is considered as a rectangular area A of size $width_A \times height_A$. The area overlaps with a grid of $size_g \times size_g$ square grid cells. If the number of grid cells in a row and that in a column is n_{g_x} and n_{g_y}, respectively, then $n_{g_x} = ceiling(\frac{width_A}{size_g})$, $n_{g_y} = ceiling(\frac{height_A}{size_g})$. Figure 2.1 shows a visualization of the grid, where a grid cell is represented as $g_{(a,b)}$ $(a \in [0, n_{g_x} - 1],\ b \in [0, n_{g_y} - 1])$. Its four borders are represented by a set $B_{(a,b)} = \{\beta^n_{(a,b)}, \beta^e_{(a,b)}, \beta^s_{(a,b)}, \beta^w_{(a,b)}\}$, where the superscripts n, e, s, and w represent the border on the *east*, *west*, *south*, and *north* side, respectively. In Figure 2.1, $a_{max} = n_{g_x} - 1$, $b_{max} = n_{g_y} - 1$. In this way, the grid cells cover all trajectory location points and trajectory segments and each location point is located inside a grid cell or on the border of a grid cell.

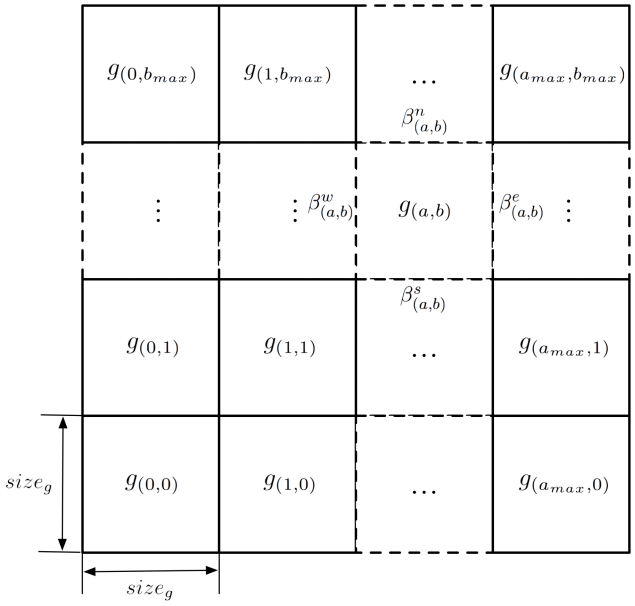


Figure 2.1: A visualization of the grid. $g_{(a,b)}$ $(a \in [0, a_{max}],\ b \in [0, b_{max}])$ is a grid cell and $\beta^e_{(a,b)}, \beta^w_{(a,b)}, \beta^s_{(a,b)}, \beta^n_{(a,b)}$ are its four borders.

When the grid-based method is applied, there exist situations where a trajectory segment passes through some grid cell but has no location point in it. An example is shown in Figure 2.2, in which the trajectory segment $trajSeg_m : q_m q_{m+1}$ passes through the grid cell on the right side but neither of its location points is located inside that grid cell. In order to make things

easier when this kind of situation happens, we interpolate our trajectory data. A location point is inserted whenever a trajectory segment intersects with a grid cell border, like the points p_{n-1}, p_{n+1}, p_{n+2}, and p_{n+3} in Figure 2.2.

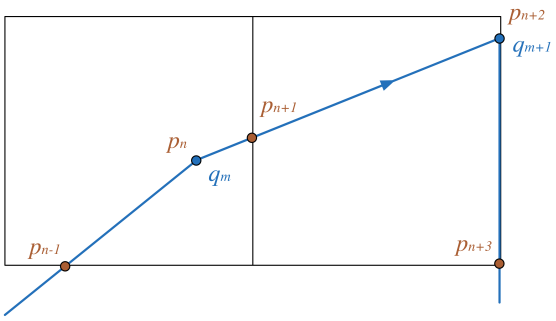


Figure 2.2: An example of interpolation on a trajectory. p_{n-1}, p_{n+1}, and p_{n+3} are newly inserted location points.

After interpolation, trajectory *traj*: $q_1 q_2 \dots q_M$ is transformed to a new sequence of location points, i.e., *traj*: $p_1 p_2 \dots p_N$, where $N \geq M$ and p_n ($n \in [1, N]$) is either an original location point or a newly inserted one. If p_n is a newly inserted location point between two location points q_m and q_{m+1}, t_n is calculated as follows.

$$t_n = t_m + \frac{x_n - x_m}{x_{m+1} - x_m} \cdot (t_{m+1} - t_m)$$

After interpolation, the new location point sequence still conforms to Definition 1. In other words, a *trajectory* is still a *trajectory* after interpolation. The only difference is that it might have more location points in the sequence. Thus a new definition for an interpolated trajectory is not necessary. Since the trajectory location point sequences to be used for outlier detection are the ones after interpolation, from here onwards, *trajectory* refers to the location point sequence after interpolation. Besides, Definition 2 still holds for *trajectory segments* of an interpolated trajectory. The difference is that no trajectory segment will span more than one grid cell.

2.3 Trajectories and Grid Cells

When the grid-based method and the interpolation approach are applied, there exist some relationships between trajectories and grid cells.

Definition 3 (entering point): A trajectory location point that is located on a grid cell border is a *grid cell entering location point*. In this research, grid cell entering location point is also called *entering point* for

short. Accordingly, for a trajectory $traj: p_1p_2 \dots p_N$, p_n ($n \in [1, N]$) is an entering point if x_n or y_n is divisible by $size_g$. Besides, although p_1 might not be located on a grid cell border, we also consider it as an entering point.

Following this definition, observe that all the newly inserted location points in an interpolated trajectory are entering points.

Definition 4 (belong to / in): After interpolation, each trajectory segment of a trajectory *belongs to* a grid cell. If the grid cell trajectory segment $trajSeg_n$ belongs to is referred to as $g_{(a_n,b_n)}$, $g_{(a_n,b_n)}$ is calculated as follows.

$$a_n = floor(\frac{min(x_n, x_{n+1})}{size_g})$$

$$b_n = floor(\frac{min(y_n, y_{n+1})}{size_g})$$

In this case, $trajSeg_n$ is also referred to as $trajSeg_n^{(a_n,b_n)}$ in order to indicate the belonging relationship directly. When $trajSeg_n$ belongs to grid cell $g_{(a_n,b_n)}$, its starting location point, i.e., p_n, also *belongs to* $g_{(a_n,b_n)}$. Moreover, if p_n is located on a border of $g_{(a_n,b_n)}$, it is an *entering point* of $g_{(a_n,b_n)}$ and it is said that $trajSeg_n$ *enters* $g_{(a_n,b_n)}$ at p_n. Specially, p_1 is an *entering point* of $g_{(a_1,b_1)}$ and it is said that $trajSeg_1$ *enters* $g_{(a_1,b_1)}$ at p_1 no matter p_1 is located on a border or not. For convenience, from here onwards, when we say a trajectory segment or a location point is *in* a grid cell, it means that the trajectory segment or location point *belongs to* that a grid cell.

To have a clearer perspective of the belonging relationship defined here, the five situations of a trajectory segment $trajSeg_n$ of a trajectory are listed as follows.

1. Both p_n and p_{n+1} are located inside a grid cell $g_{(a,b)}$;
2. p_n (or p_{n+1}) is located on a border of $g_{(a,b)}$ and p_{n+1} (or p_n) is inside $g_{(a,b)}$;
3. $\beta^1_{(a,b)}$ and $\beta^2_{(a,b)}$ are two borders of grid cell $g_{(a,b)}$, i.e., $\beta^1_{(a,b)}, \beta^2_{(a,b)} \in B_{(a,b)}$, and p_n is located on $\beta^1_{(a,b)}$ and p_{n+1} on $\beta^2_{(a,b)}$;
4. $\beta^e_{(a,b)}$ is the border between grid cell $g_{(a,b)}$ and $g_{(a+1,b)}$, and both p_n and p_{n+1} are located on $\beta^e_{(a,b)}$;
5. $\beta^n_{(a,b)}$ is the border between grid cell $g_{(a,b)}$ and $g_{(a,b+1)}$, and both p_n and p_{n+1} are located on $\beta^n_{(a,b)}$.

In Figure 2.3, $trajSeg_1$, $trajSeg_2$, $trajSeg_3$, $trajSeg_4$, and $trajSeg_5$ are examples of item 1 to 5, respectively. In this figure, $trajSeg_1$ belongs to $g_{(a,b)}$, $trajSeg_2$ belongs to $g_{(a,b+1)}$, $trajSeg_3$ belongs to $g_{(a+1,b+1)}$, and $trajSeg_4$ and $trajSeg_5$ belong to $g_{(a+1,b)}$.

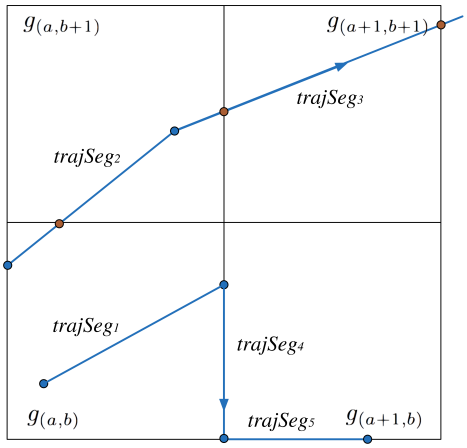


Figure 2.3: Some examples of trajectory segments in different situations.

In this research, a trajectory outliers detection algorithm is applied on a set of trajectories, which is referred to as $T = \{traj_1, traj_2, \ldots, traj_{num_{traj}}\}$, where num_{traj} is the number of trajectories in the set. Each trajectory $traj_i$ ($i \in [1, num_{traj}]$) has N_i location points and $|S_i| = N_i - 1$ trajectory segments. The set of trajectory segments of $traj_i$ is represented as $S_i = \{trajSeg_1^i, trajSeg_2^i, \ldots, trajSeg_{|S_i|}^i\}$.

Following Definition 3, the ordered sequence of entering points of a trajectory $traj_i$ can be extracted. Such a sequence is represented as E_i and the number of elements in E_i is represented as $|E_i|$, which indicates that there are $|E_i|$ entering points in $traj_i$. Also, following Definition 4, the ordered sequence of grid cells $traj_i$ passes through can be extracted. A sequence G_i is used to represent all the grid cells $traj_i$ passes through. In G_i there are $|G_i|$ elements, indicating that $traj_i$ passes through a total number of $|G_i|$ grid cells. The elements in G_i might not be unique due to the fact that $traj_i$ can enter a grid cell multiple times.

By evaluating all the trajectories in T, the sequence of entering points of a grid cell $g_{(a,b)}$ can be extracted. Such a sequence is referred to as $E^{(a,b)}$ and the number of elements in $E^{(a,b)}$ is referred to as $|E^{(a,b)}|$, which indicates the number of entering points in $g_{(a,b)}$. Besides, the set of trajectory segments in $g_{(a,b)}$ can also be obtained. This set is represented as $S^{(a,b)}$ and the number of elements in it is represented as $|S^{(a,b)}|$. Since a trajectory might enter a grid cell and has more than one trajectory segments in that grid cell before it enters another grid cell, like the situation shown in the grid cell $g_{(a,b+1)}$ in Figure 2.3, the inequation $|S^{(a,b)}| \geq |E^{(a,b)}|$ holds.

Hence, the elements that are in both E_i and $E^{(a,b)}$ compose the set of the entering points of $traj_i$ that are in $g_{(a,b)}$. This set is $E_i^{(a,b)}$ and the number of elements in it is $|E_i^{(a,b)}|$, which indicates the number of times $traj_i$ enters

$g_{(a,b)}$. Also, the elements that are in both set S_i and $S^{(a,b)}$ form the set $S_i^{(a,b)}$, where there are $|S_i^{(a,b)}|$ elements and each element is a trajectory segment that belongs to both $traj_i$ and $g_{(a,b)}$.

2.4 Features of Trajectories

In this section the definitions of some important features of trajectories that can further be used for outlier detection in a set of trajectories are given.

Definition 5 (distance): The *distance* between a trajectory (or trajectory segment) pair is used to compare the two trajectories (or trajectory segments). It is obtained by a distance function or a metric, for example Euclidean distance. A smaller distance indicates a larger similarity, and vice versa. Following the work of Lee et al. [8], we define a distance function for trajectory data. The distance between two trajectory segments $trajSeg_n$ and $trajSeg_m$, i.e., $dis(n,m)$, is calculated by three components: perpendicular distance $d_\perp(n,m)$, parallel distance $d_\parallel(n,m)$ and angular distance $d_\theta(n,m)$, i.e., $dis(n,m) = \omega_\perp d_\perp(n,m) + \omega_\parallel d_\parallel(n,m) + \omega_\theta d_\theta(n,m)$, where the weights $\omega_\perp$, $\omega_\parallel$, and ω_θ are determined depending on applications. $d_\perp(n,m)$, $d_\parallel(n,m)$, and $d_\theta(n,m)$ are calculated as follows and also shown in Figure 2.4.

$$d_\perp(trajSeg_n, trajSeg_m) = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}}$$

$$d_\parallel(trajSeg_n, trajSeg_m) = Min(l_{\parallel 1}, l_{\parallel 2})$$

$$d_\theta(trajSeg_n, trajSeg_m) = \begin{cases} \parallel len_n \parallel \times sin(\theta) & \text{if } 0^\circ \leq \theta \leq 90^\circ \\ \parallel len_m \parallel & \text{if } 90^\circ \leq \theta \leq 180^\circ \end{cases}$$

In Figure 2.4 and the three formulas, p'_n and p'_{n+1} are the projections of

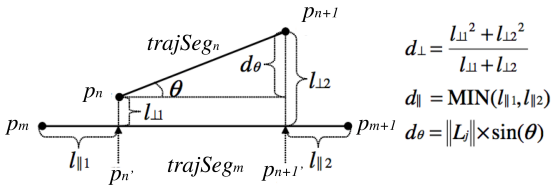


Figure 2.4: The visualization and calculation of the three components of distance [8].

p_n and p_{n+1} on $trajSeg_m$, respectively, $l_{\perp 1}$, $l_{\perp 2}$, $l_{\parallel 1}$, and $l_{\parallel 2}$ are the lengths of line segment $p_n p'_n$, $p_{n+1} p'_{n+1}$, $p_m p'_n$, and $p'_{n+1} p_{m+1}$, respectively, and θ is the angle between $trajSeg_n$ and $trajSeg_m$.

Definition 6 (density): The trajectory *density* of a grid cell $g_{(a,b)}$, which is referred to as $Den^{(a,b)}$, is defined as the number of trajectories that have entering point(s) in $g_{(a,b)}$.

$$Den^{(a,b)} = \sum_{i=1}^{num_{traj}} min(|E_i^{(a,b)}|, 1)$$

Following this definition, observe that when a trajectory $traj_i$ enters grid cell $g_{(a,b)}$ $|E_i^{(a,b)}|$ ($|E_i^{(a,b)}| > 1$) times, $Den^{(a,b)}$ increases only once.

Note that if two trajectories $traj_i$ and $traj_{i+1}$ enter the same grid cell $g_{(a,b)}$, it implies that these two trajectories are close to each other in $g_{(a,b)}$, and their trajectory segments in $g_{(a,b)}$ are neighboring trajectories. Hence, for a trajectory segment $trajSeg_n$ in grid cell $g_{(a_n,b_n)}$, the number of neighboring trajectory segments of $trajSeg_n$ can be indicated by $Den^{(a,b)}$.

Definition 7 (direction): The *direction* of a trajectory segment can be rounded to one of the direction elements in a direction vector $dirVector$, where $dirVector = (d_1, d_2, \ldots, d_{num_{dir}}) = (\theta_1, \theta_2, \ldots, \theta_{num_{dir}})$. In the direction vector, each direction $d_k (k \in [1, num_{dir}])$ is represented by the angle θ_k between itself and direction $d_0 = 0$, as shown in Figure 2.5. If there are num_{dir} ($num_{dir} > 1$) elements in the direction vector, then

$$\theta_k = (k-1) \cdot \frac{2\pi}{num_{dir}},\ k \in [1, num_{dir}]$$

Following this formula, it is easy to notice that $\theta_2 = \frac{2\pi}{num_{dir}}$ is the smallest

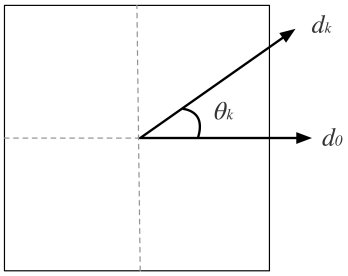


Figure 2.5: A visualization of the relationship between d_k and θ_k.

angle in $dirVector$ that is larger than zero, and the difference between every θ_k and θ_{k+1} ($k \in [1, num_{dir} - 1]$) is θ_2.

Accordingly, the direction of $dirVector$ to which a trajectory segment $trajSeg_n$ belongs, which is referred to as dir_n, can be calculated as follows.

$$dir_n = [\frac{\theta_n}{\theta_2}]$$

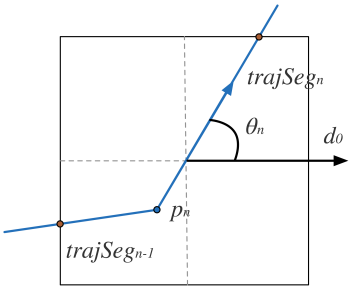


Figure 2.6: A visualization of θ_n, which is the angle from d_0 to $trajSeg_n$

where θ_n ($\theta_n \in [0, 2\pi)$) is the angle from d_0 to $trajSeg_n$, as shown in Figure 2.6.

Note that a trajectory $traj_i$ might have more than one trajectory segments in a grid cell $g_{(a,b)}$, i.e., $|S_i^{(a,b)}| > 1$. In this case, the *general direction* of $traj_i$ in $g_{(a,b)}$ is considered.

Definition 8 (general direction): When a trajectory $traj_i$ enters a grid cell $g_{(a,b)}$ at p_n and then leaves at p_m, there exists a vector p_np_m connected by p_n and p_m, as shown in Figure 2.7. The *general direction* of trajectory $traj_i$ in $g_{(a,b)}$, which is referred to as $Dir_i^{(a,b)}$, is considered as the direction of p_np_m, which is calculated the same way as calculating the direction of a trajectory segment, i.e.,

$$Dir_i^{(a,b)} = [\frac{\theta_{nm}}{\theta_2}]$$

where θ_{nm} is the angle from d_0 to p_np_m, as shown in Figure 2.7.

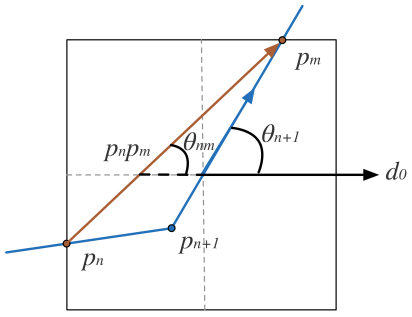


Figure 2.7: A visualization of d_{nm} and d_{n+1}, d_{nm} being the angle from d_0 to p_np_m and d_{n+1} being the angle from d_0 to p_{n+1}.

Following this definition, observe that when a trajectory $traj_i$ enters grid cell $g_{(a,b)}$ $|E_i^{(a,b)}|$ ($|E_i^{(a,b)}| > 1$) times, it has more than one general directions in $g_{(a,b)}$.

Definition 9 (direction frequency): The *direction frequency* of a grid

cell $g_{(a,b)}$ is defined as a vector of num_{dir} elements, i.e., $DirFreq^{(a,b)} = (f_1^{(a,b)}, f_2^{(a,b)}, \ldots, f_{num_{dir}}^{(a,b)})$. If the number of the trajectories passing through grid cell $g_{(a,b)}$ that has a general direction d_k is referred to as $R_{d_k}^{(a,b)}$, f_k ($k \in [1, num_{dir}]$) is the ratio of $R_{d_k}^{(a,b)}$ to $|E^{(a,b)}|$, i.e.,

$$f_k = \frac{R_{d_k}^{(a,b)}}{|E^{(a,b)}|}$$

The equation $\sum_{k=1}^{num_{dir}} f_k = 1$ holds.

Definition 10 (duration): The time a trajectory segment $trajSeg_n : p_n p_{n+1}$ spends within the grid cell it is in is called its *(time) duration* in that grid cell, which is represented as dur_n. If $trajSeg_n$ belongs to $g_{(a,b)}$, the duration of $trajSeg_n$ in $g_{(a,b)}$, which is represented as $dur_n^{(a,b)}$, is defined as the time difference between location point p_n and p_{n+1}, i.e.,

$$dur_n = dur_n^{(a,b)} = t_{n+1} - t_n$$

If $traj_i$ has $n_2 - n_1$ trajectory segments $trajSeg_{n_1}, \cdots, trajSeg_{n_2-1}$ between two consecutive entering points p_{n_1} and p_{n_1} ($n_2 > n_1$), the total duration of these trajectory segments is $sumDur_i^{(a_{n_1}, b_{n_1})} = \sum_{n=n_1}^{n_2} dur_n$, which is also called the *(total) duration* of $traj_i$ in $g_{(a_{n_1}, b_{n_1})}$ when it enters at p_{n_1}.

Definition 11 (average duration): If grid cell $g_{(a,b)}$ has $|E^{(a,b)}|$ entering points and $|S^{(a,b)}|$ trajectory segments, the *average duration* of $g_{(a,b)}$, which is referred to as $\overline{Dur^{(a,b)}}$, is calculated as follows.

$$\overline{Dur^{(a,b)}} = \frac{\sum_{n=1}^{|S^{(a,b)}|} dur_n^{(a,b)}}{|E^{(a,b)}|}$$

Following Definition 10 and 11, observe that when a trajectory $traj_i$ enters grid cell $g_{(a,b)}$ $|E_i^{(a,b)}|$ ($|E_i^{(a,b)}| > 1$) times, they are treated separately.

Definition 12 (speed): The *speed* of a trajectory segment $trajSeg_n : p_n p_{n+1}$, which is referred to as spd_n, is calculated by the distance and the time difference between location point p_n and p_{n+1}, i.e.,

$$spd_n = \frac{len_n}{dur_n}$$

If trajectory segment $trajSeg_n$ belongs to grid cell $g_{(a,b)}$, its speed in $g_{(a,b)}$ can also be referred to as $spd_n^{(a,b)}$.

Definition 13 (average speed): If $traj_i$ has $n_2 - n_1$ trajectory segments $trajSeg_{n_1}, \cdots, trajSeg_{n_2-1}$ between two consecutive entering points p_{n_1} and p_{n_2} ($n_2 > n_1$), the average speed of $traj_i$ in $g_{(a_{n_1}, b_{n_1})}$ when it enters at p_{n_1} is calculated as follow.

$$\overline{Spd_i^{(a_{n_1}, b_{n_1})}} = \frac{\sum_{n=n_1}^{n_2} len_n}{\sum_{n=n_1}^{n_2} dur_n}$$

Note that when a trajectory $traj_i$ enters a grid cell $g_{(a,b)}$ $|E_i^{(a,b)}|$ ($|E_i^{(a,b)}| > 1$) times, they are treated separately, i.e., $traj_i$ has more than one average speed in $g_{(a,b)}$.

In addition to trajectories, the concept average duration is also applicable for grid cells. If grid cell $g_{(a,b)}$ has $|E^{(a,b)}|$ entering points, the *average speed* of $g_{(a,b)}$, which is referred to as $\overline{Spd^{(a,b)}}$, is calculated as follows.

$$\overline{Spd^{(a,b)}} = \frac{\sum_{i=1}^{|E^{(a,b)}|} \overline{Spd_i^{(a,b)}}}{|E^{(a,b)}|}$$

To sum up, following Definition 5 to 13 in this section, distances between trajectory segments can be calculated. Besides, if $traj_i$ is a trajectory that passes through grid cell $g_{(a,b)}$, its general direction, duration, and average speed in $g_{(a,b)}$ can be obtained. These features of $traj_i$ are represented as $Dir_i^{(a,b)}$, $sumDur_i^{(a,b)}$, and $\overline{Spd_i^{(a_n,b_n)}}$, respectively. In addition, for a grid cell $g_{(a,b)}$, its density, direction frequency, average duration, and average speed can also be obtained. These features of $g_{(a,b)}$ are referred to as $Den^{(a,b)}$, $DirFreq^{(a,b)}$, $\overline{Dur^{(a,b)}}$, and $\overline{Spd^{(a,b)}}$, respectively.

2.5 Trajectory Outliers

In this research, we aim to detect trajectory outliers in a trajectory set T in terms of density, direction, duration, and speed. The level of outlier in these four aspects will be measured by four outlying scores (OSs). In this section, how the four OSs are calculated is explained. Besides, the definitions of these types of trajectory outliers are given.

Definition 14 (density outlier): Outlying score is used to indicate the level of outlier of a trajectory. A trajectory $traj_i$ is a *density outlier* if its *density outlying score* $denOS_i$ is larger than λ_{den}, where λ_{den} is an input parameter. $denOS_i$ is calculated by evaluating all the grid cells $traj_i$ passes through, i.e.,

$$denOS_i = \sum_{j=1}^{|E_i|} denos_i^{(a_j,b_j)}$$

where $denos_i^{(a_j,b_j)}$ is the outlying score of $traj_i$ in $g_{(a_j,b_j)}$, $g_{(a_j,b_j)}$ being a grid cell $traj_i$ passes through. If $denOS_i > \lambda_{den}$, $traj_i$ is an outlier. The value of $denos_i^{(a_j,b_j)}$ is calculated as follows.

$$denos_i^{(a_j,b_j)} = \begin{cases} sumDur_i^{(a_j,b_j)}, & \text{if } Den^{(a_j,b_j)} < \tau_{den} \\ 0, & \text{otherwise} \end{cases}$$

where τ_{den} is called the density OS parameter. This formula shows that $denOS_i$ increases by $sumDur_i^{(a_j,b_j)}$ whenever $traj_i$ passes through a grid

cell $g_{(a_j,b_j)}$ of density $Den^{(a_j,b_j)} < \tau_{den}$. This implies that a trajectory that has larger duration in a grid cell $g_{(a_j,b_j)}$ of density $Den^{(a_j,b_j)} < \tau_{den}$ is considered to be more outlying than a trajectory that has smaller duration in $g_{(a_j,b_j)}$.

Definition 15 (direction outlier): A trajectory $traj_i$ is a *direction* outlier if its *direction outlying score* $dirOS_i$ is greater than the direction OS threshold λ_{dir}, where λ_{dir} is an input parameter.

Firstly, for convenience, a function $F(Dir_i^{(a_j,b_j)})$ is defined as follows [4].

$$F(Dir_i^{(a_j,b_j)}) = \frac{1 - \sum_{k=1}^{num_{dir}} f_k^{(a_j,b_j)} \cdot cos(Dir_i^{(a_j,b_j)} - d_k)}{2}$$

$F(Dir_i^{(a_j,b_j)})$ calculates the difference between the general direction of $traj_i$ in $g_{(a_j,b_j)}$ and the direction information of $g_{(a_j,b_j)}$. It has a range of $[0, 1]$;

Then the direction outlying score of $traj_i$ in $g_{(a_j,b_j)}$, which is represented as $diros_i^{(a_j,b_j)}$, can be calculated as follows.

$$diros_i^{(a_j,b_j)} = \begin{cases} sumDur_i^{(a_j,b_j)} \cdot F(Dir_i^{(a_j,b_j)}), & \text{if } F(Dir_i^{(a_j,b_j)}) > \tau_{dir} \\ 0, & \text{otherwise} \end{cases}$$

where τ_{dir} is the direction OS parameter. This formula implies that if a trajectory $traj_i$ has a different enough direction in $g_{(a_j,b_j)}$ compared to the direction information of $g_{(a_j,b_j)}$, it is considered outlying. Besides, if $traj_i$ has larger duration in $g_{(a_j,b_j)}$, it is considered more outlying than when it has smaller duration in $g_{(a_j,b_j)}$.

Then, the direction OS of $traj_i$ can be calculated as follows. If $dirOS_i > \lambda_{dir}$, $traj_i$ is considered as a direction outlier.

$$dirOS_i = \sum_{j=1}^{|E_i|} diros_i^{(a_j,b_j)}$$

Definition 16 (duration outlier): A trajectory $traj_i$ is a *duration* outlier if its *duration outlying scores* $durOS_i$ is greater than some input parameter λ_{dur}. The value of $durOS_i$ is calculated as follows. If $durOS_i > \lambda_{dur}$, $traj_i$ is a duration outlier.

$$durOS_i = \sum_{j=1}^{|E_i|} duros^{(a_j,b_j)}$$

where $duros^{(a_j,b_j)}$ is the duration outlying score of a grid cell $g_{(a_j,b_j)}$ $traj_i$ passes through. The value of $duros_i^{(a_j,b_j)}$ is calculated as follows.

$$duros_i^{(a_j,b_j)} = \begin{cases} sumDur_i^{(a_j,b_j)}, & \text{if } F(sumDur_i^{(a_j,b_j)}) > \tau_{dur} \\ 0, & \text{otherwise} \end{cases}$$

where

$$F(sumDur_i^{(a_j,b_j)}) = \frac{|sumDur_i^{(a_j,b_j)} - \overline{Dur^{(a_j,b_j)}}|}{\overline{Dur^{(a_j,b_j)}}}$$

and τ_{dur} is the duration OS parameter.

Definition 17 (speed outlier): A trajectory $traj_i$ is a *speed* outlier if its *speed outlying scores* $spdOS_i$ is greater than some input parameter λ_{spd}. The value of $spdOS_i$ is calculated as follows. If $spdOS_i > \lambda_{spd}$, $traj_i$ is a speed outlier.

$$spdOS_i = \sum_{j=1}^{|E_i|} spdos_i^{(a_j,b_j)}$$

where $spdos_i^{(a_j,b_j)}$ is the speed outlying score of trajectory $traj_i$ in $g_{(a_j,b_j)}$ and is calculated as follows.

$$spdos_i^{(a_j,b_j)} = \begin{cases} sumDur_i^{(a_j,b_j)}, & \text{if } F(\overline{Spd_i^{(a_j,b_j)}}) > \tau_{spd} \\ 0, & \text{otherwise} \end{cases}$$

where

$$F(\overline{Spd_i^{(a_j,b_j)}}) = \frac{|\overline{Spd_i^{(a_j,b_j)}} - \overline{Spd^{(a_j,b_j)}}|}{\overline{Spd^{(a_j,b_j)}}}$$

and τ_{spd} is the speed OS parameter.

Duration and speed outlier

According to Definition 13, the following formula holds.

$$\sum_{n=n_1}^{n_2} dur_n = \frac{\sum_{n=n_1}^{n_2} len_n}{\overline{Spd_i^{(a_{n_1},b_{n_1})}}}$$

This formula shows that there is a correlationship between duration and speed. It can be imagined that if a trajectory has an abnormal duration in a grid cell, probably it also has an abnormal speed in that grid cell. It is hard to determine whether such kind of abnormal behavior is caused by moving speed or time duration since both OSs would increase following Definition 16 and 17. To clearly show the cause of such an abnormal temporal-spatial behavior, in this project we separate duration and speed outliers by categorizing a trajectory/pedestrian's status in a grid cell as *standing*, *crossing*, and *others* (which includes moving around, walking up and down, etc), as shown in Figure 2.8. In the first grid cell of Figure 2.8 there is a standing point indicating that the trajectory remains stationary for a while. In the second grid cell of Figure 2.8, the trajectory does nothing but simply passes across. And in the third grid cell the trajectory moves around.

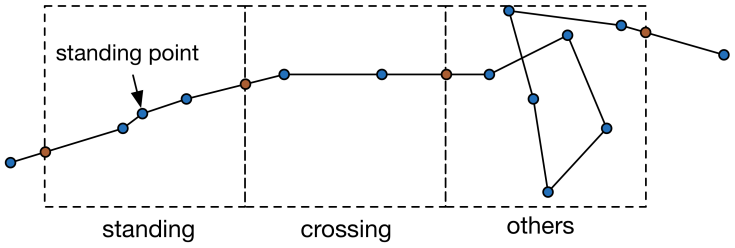


Figure 2.8: A visualization of the the three statuses of a trajectory in a grid cell.

Actually, the standing point in this figure shows an ideal case, where the sampled data always indicate exactly the same location when a pedestrian is stationary. However, in reality this might not be true due to the precision of sensor systems and the implementation of localization algorithms. Sampled data might indicate different locations when a person is actually staying still, like the situation shown in Figure 2.9. In this figure, sampled data indicates that the pedestrian moves up and down at the place he actually stays still. This kind of situation can be solved by a stay point detection algorithm [11], which is able to determine that a trajectory like the one in Figure 2.9 is actually the one shown in the first grid cell of Figure 2.8. A stay point detection algorithm detects stay points automatically from a

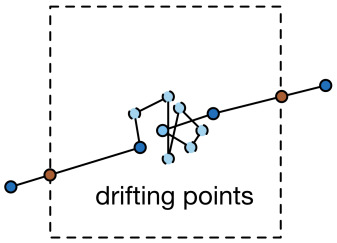


Figure 2.9: A visualization of the real-life situation where there is a pedestrian is stationary for a while.

trajectory by seeking the spatial region where the trajectory spends a period exceeding a certain threshold. For instance, if a trajectory spends more than 10 minutes within a distance of 5 meters, the region is detected as a stay point. Therefore, in this project, focus will not be on this issue and it is assumed that this kind of problem has been solved beforehand. This work only considers the three status shown in Figure 2.8.

These three status can be determined by examining the average speed $\overline{Spd_i^{(a,b)}}$ and the total length $\sum_{n=1}^{|S_i^{(a,b)}|} len_n$ of a trajectory $traj_i$ in a grid

cell $g_{(a,b)}$. If $traj_i$ is standing in $g_{(a,b)}$, it should have a small $\overline{Spd_i^{(a,b)}}$; if it is simply crossing $g_{(a,b)}$, it should have a larger $\overline{Spd_i^{(a,b)}}$ but not a large $\sum_{n=1}^{|S_i^{(a,b)}|} len_n$. The following assumption is made before the definition of the three statuses are given.

Assumption 1: A normal trajectory has a normal speed of spd_{norm}.

Based on this assumption, the following definitions are given.

Definition 18 (standing): If a trajectory $traj_i$ has an average speed $\overline{Spd_i^{(a,b)}}$ smaller than $\rho_{spd} \cdot spd_{norm}$, its status in $g_{(a,b)}$ is determined as *standing*. ρ_{spd} is a parameter that is used to differ between walking slowly and standing still.

Definition 19 (crossing): If a trajectory has a total length $\sum_{n=1}^{|S_i^{(a,b)}|} len_n$ smaller than $\rho_{len} \cdot \sqrt{2} \cdot size_g$, and an average speed $\overline{Spd_i^{(a,b)}}$ larger than $\rho_{spd} \cdot spd_{norm}$ in grid cell $g_{(a,b)}$, its status in $g_{(a,b)}$ is determined as *crossing*. ρ_{len} is a parameter that is used differ between crossing and moving around in a grid cell.

Definition 20 (others): If a trajectory has neither standing nor crossing status in a grid cell $g_{(a,b)}$, its status is *others*.

Now that the three statuses are defined, if a trajectory has standing status in a grid cell, only duration OS is considered when examine that grid cell. If a trajectory has crossing status in a grid cell, only speed OS is considered. And if a trajectory is not within the former two cases, both OSs should be considered. Accordingly, the formulas of $duros_i^{(a_j,b_j)}$ and $spdos_i^{(a_j,b_j)}$ in 16 and 17 are updated as follows.

$$duros_i^{(a_j,b_j)} = \begin{cases} 0, & \text{if } F(sumDur_i^{(a_j,b_j)}) < \tau_{dur} \\ & \text{or } traj_i \text{ has crossing status in } g_{(a_j,b_j)} \\ sumDur_i^{(a_j,b_j)}, & \text{otherwise} \end{cases}$$

$$spdos_i^{(a_j,b_j)} = \begin{cases} 0, & \text{if } F(\overline{Spd_i^{(a_j,b_j)}}) < \tau_{spd} \\ & \text{or } traj_i \text{ has standing status in } g_{(a_j,b_j)} \\ sumDur_i^{(a_j,b_j)}, & \text{otherwise} \end{cases}$$

Chapter 3

Related Work

The improvements in tracking facilities have made it possible to collect huge amounts of trajectory data of moving objects. Nowadays, there is much work in literature regarding the issue of trajectory outlier detection. These TOD techniques can be used for surveillance security, behavior analysis, and many other purposes. Previously in Section 2.5 four types of outliers are defined. Some of them can be detected by certain TOD algorithms. In this chapter, we categorize relevant previous TOD algorithms into four groups: (i) grid-based methods, (ii) distance-based methods, (iii) classification and clustering-based methods, and (iv) the methods that are based on trajectory similarity measurement. In the following sections, we review these TOD approaches and analyze which types of outliers they can or cannot detect.

3.1 Grid-based

The first group contains grid-based methods. The basic idea of these methods is to find connections between trajectories and grid cells and thus be able to detect outliers based on grid. These methods often have small computational complexity. In this section three grid-based techniques are introduced. The first two are similar in the way of detecting outliers while the third one has a different detecting technique compared to the former two.

3.1.1 Density-based

Density-based approaches have long been used in the outlier detection area. Some ideas of the density-based outlier detection approaches that deal with other data types instead of trajectory data can also be extended to the TOD area. Traditional density-based TOD methods rely on the notion of LOF (local outlier factor). This requires a distance function for trajectory comparison and k-nearest neighbor (k-NN) search. The calculation of distance and the search for nearest neighbors can be computationally expensive.

TOP-EYE [4] is an example of density-based TOD methods. In TOP-EYE, the notion of density is integrated with grid cell, which is similar as Definition 6. Following this definition, TOP-EYE detects density-based outliers similarly as what Definition 14 does. In this way, TOP-EYE avoids a large amount of distance computation and k-NN search for trajectories. More specifically, in this method, after a single evaluation of all trajectories in set T, the density of each grid cell can be obtained. Then the density outlying score of a trajectory can be calculated by examining the densities of all the grid cells it passes. After that, the larger the outlying score is, the higher its level of outlier is. Using this grid-based method, when a new trajectory is introduced, there is no need to evaluate all the trajectories another time to decide whether this trajectory is outlying or not. However, when a method is grid-based, the performance of outlier detection is related to the size of a grid cell. Although having a relatively large cell size reduces computation, it decreases accuracy. On the contrary, having a relatively small cell size increases accuracy, but also increases computational complexity. And in extreme cases, having a too large or too small cell size will not be able to do outlier detection correctly. There is a certain tradeoff between efficiency and accuracy so the cell size needs to be chosen carefully. Besides, if these density-based approaches are not combined with other approaches and no temporal information is taken into account, they can only detect density-based outliers.

3.1.2 Direction-based

Some other TOD algorithms are designed to detect direction-based outliers. Direction-based TOD approaches are able to detect trajectories that have abnormal directions in one or more areas. To determine whether a trajectory has an abnormal direction in some area, it is compared with its adjacent trajectories. In addition to being a density-based algorithm, TOP-EYE [4] is also direction-based. In this algorithm, direction is defined similarly as in Definition 7, i.e., the direction of a trajectory segment is rounded to one of the elements in the direction vector. Since it is a grid-based method, the direction information, which is similar as the direction frequency defined in Definition 9, of each grid cell should be obtained for outlier detection. After that, the direction outlying score of each trajectory is calculated by evaluating all the grid cells it passes. When a trajectory is found out to have a very different direction compared to the direction information of a grid cell it passes, its outlying score increases. After evaluating all the grid cells a trajectory passes, the trajectory will be determined to have a higher outlier level if it has a larger outlying score, and vise versa. As mentioned before in Section 3.1.1, this grid-based method has some strengths and also some weaknesses. Besides, in terms of being direction-based, TOP-EYE has some problems when a trajectory passes some grid cell where no other trajectory

passes because it has no adjacent trajectory to compare with. What TOP-EYE does to solve this problem is to combine a direction-based method with a density-base method, which also makes it possible to detect more types of outliers.

3.1.3 Isolation-based

Outlying trajectories can be considered as trajectories that isolate from most of the other trajectories in the data set in terms of some chosen attribute. Zhang et al. [21] proposed an Isolation-Based Anomalous Trajectory (iBAT) detection method, which exploits the property that anomalies are susceptible to a mechanism called isolation. iBAT is grid-based. It exploits anomalous trajectories' intrinsic properties of being "few and different", i.e., anomalous trajectories are the minority in the data set and should be different in some way compared to the others. Another similar method is called iBOAT [2], which is also grid-based and is an improvement over iBAT. Instead of labeling a whole trajectory as an outlier, iBOAT is able to determine which parts of a trajectory are anomalous. The basic idea of iBAT and iBOAT is to find out the trajectories that pass one or more grid cells where no other trajectory passes. In this way, no distance and density calculation is needed. They firstly need to get the sequence of grid cells each trajectory $traj_i$ passes, which is E_i, by evaluating all the trajectories in set T. Then, a sub-set of T, which is T', is randomly chosen and these methods calculate the number of iterations it needs to isolate a test trajectory $traj_t$ from the trajectories in T'. In each iteration, a grid cell g that $traj_t$ passes is randomly chosen. Examination is made to see if there exist other trajectories in T' that also pass g. The trajectories that do not pass g will be deleted from T'. This is the *isolation* process. After several iterations, a trajectory is isolated when it finds out that it passes a grid cell where no other trajectories pass. In this way, a smaller number of iterations indicates a more possible outlying trajectory because it shows that the trajectory is easily isolated from the others, and vice versa. These two methods are good because they are able to detect some density outliers with efficiency. However, iBAT and iBOAT cannot detect duration or speed outliers because they do not consider temporal information. Besides, they are not able to detect direction-based outliers because they only consider whether a trajectory passes some grid cell but have no concern of the direction of that trajectory.

The core steps of iBAT are shown in Algorithm 1, which is based on the basic idea introduced above.

3.2 Distance-based

Distance-based outlier detection is another popular approach in the outlier detection area. Most existing metrics used for distance-based techniques

Algorithm 1 iBAT

Inputs: T - a set of trajectories
$traj_t \in T$ - test trajectory
m - number of running trials
ψ - sub-sample size

procedure
  let n be a vector of m zeros (n_i's are zeros)
  for i=1 to m **do**
    $T' \leftarrow$ randomly sample ψ trajectories from T
    repeat
      $n_i \leftarrow n_i + 1$
      randomly choose a grid cell g $traj_t$ passes
      $T' \leftarrow$ select the trajectories that also pass g from T'
    until T' is empty
  compute $traj_t$'s outlying score according to n and ψ.

are defined based upon the concepts of local neighborhood or kNN. The first notion of distance-based outlier was originally proposed by Knorr et al. [5]. In our case, that notion can be expresses as: a trajectory $traj_i$ in set T is a distance-based outlier if at least fraction λ_{dis} trajectories in T lie greater than distance τ_{dis} from $traj_i$, where λ_{dis} and τ_{dis} are two user-supplied parameters. The basic idea is to find out the trajectories that have very large distance to most of the other trajectories in set T. In order to obtain this information, a distance function should be defined.

Knorr et al. proposed a distance-based TOD method, in which each trajectory is summarized by an attribute vector. The attribute vector is defined as a vector of four elements: start location point that includes x and y coordinates, end location point that includes x and y coordinates, direction information that includes maximum, minimum, and average direction, and speed information that includes maximum, minimum, and average speed. Summarizing trajectories in this attribute vector fashion provides a multidimensional space in which trajectories can be compared. In this method, two distance functions are used to compare trajectories. The first one is used to calculate the difference between each attribute of every two trajectories, which is a euclidean distance function; the second one is used to sum up the differences between trajectories when all the four attributes are taken into account, which is a weighted sum function. With these tools, distance between every trajectory pair can be obtained. After that, following the notion of distance-based outlier mentioned before, outliers can be determined. Some experimental results by Knorr et al. show that this method is able to successfully detect some density, direction, duration, and speed outliers. However, not all of these types of outliers can be detected because the at-

tribute vector they choose stores limited information of an entire trajectory.

Another example of distance-based approaches is TRAOD [8]. The distance function TRAOD defines is similar as the one in Definition 5. With this distance function, a trajectory segment $trajSeg_n$ is determined to be outlying if more than λ_{dis} trajectory segments in set S have a distance larger than τ_{dis} from $trajSeg_n$. And a trajectory $traj_i$ is outlying if the total length of its outlying trajectory segments is larger than a certain threshold. TRAOD, however, suffers from high computational complexity because it requires a large amount of distance computation and k-NN search for all trajectory segments. Another weakness of TRAOD is that, when a new trajectory is introduced, a large amount of computation should be made to obtain the distances of its trajectory segments to all the trajectory segments that were previously in set S. This is also computationally expensive. And as the number of trajectory grows, the memory consumption will become large as well. Besides, since TRAOD does not take temporal information into consideration, it cannot detect important duration and speed-based outliers.

3.3 Classification and Clustering-based

The third group includes classification and clustering-base TOD algorithms. These methods are extended from traditional data-mining approaches. Same as distance-based methods, they require distance functions. Besides, they often need an attribute vector for distance measurement. By doing classification or clustering trajectories, similar trajectories will be grouped and outliers can be detected.

3.3.1 Classification-based

Classification is very common in the data mining area. It relies on a training data set, and with good-quality training data it often leads to high accuracy. As mentioned before, classification-based approaches require an attribute vector and a distance function. Classification of trajectories is done by a classifier and based on the attribute vector and the distance function. In simple outlier detection, there can just be two classes: Class normal and Class abnormal.

ROAM [13], proposed by Li et al., is an example of classification-based TOD methods. In ROAM, each trajectory is represented by a sequence of motif expressions, associated with the values related to time and location. Typical examples of motifs are straight line, right turn, left turn, u-turn, and loop. After evaluating a trajectory motif sequence, information like the average speed of a motif, the maximum speed of the trajectory, or the start location of a motif can be extracted. If each of these higher-level information

is considered as a motif attribute, a trajectory can be represented by a sequence of motif-oriented attribute vectors, where the first element indicates the motif and the following elements are the values of each motif attribute. In this way, each trajectory is put into a multi-attribute space and can be compared using a distance function. In ROAM, a motif expression contains the starting time, ending time, start location, end location, and distance traveled. From this information, attributes such as density, direction, duration, and speed can be obtained. ROAM is able to determine whether a trajectory motif has a different direction or speed, or occurs at a different time or location compared to the other trajectories in the data set, and thus is able to determine various types of outliers, including density, direction, duration, and speed outliers.

Although classification-based methods are able to detect many types of outliers when all related information is considered in the attribute vector, they also have some weaknesses. An essential task for effective classification is generating discriminative attributes [9]. When attributes are extracted from entire trajectories, these methods have limited classification capability when discriminative attributes appear at parts of trajectories. On the other hand, when attributes are extracted from trajectory segments, computation becomes more complex. Another weakness is the necessity of choosing a classifier. Classifier performance depends greatly on the characteristics of the data to be classified. There is no single classifier that works best on all given problems. When choosing a classifier, one needs to consider at least the following three problems. How many training examples are needed? What should the dimension of the attribute space be? How many categories should there be? Thus good training and test data sets are necessary. The more possible situations the training data sets include, the better the performance would be. However, more training examples increases complexity. Another weakness is the need of a proper attribute vector. The more useful information the attribute vector includes, the more types of outliers these methods are able to detect. However, notice that the more elements there are in the attribute vector, the higher dimension the attribute space is, and thus increases the higher the computational complexity.

3.3.2 Clustering-based

Unlike classification-based methods, clustering performs automated grouping without the aid of training data sets. A clustering-based TOD approach is not a direct way to detect outliers. The main purpose of a clustering algorithm is to find similarity rather than to find differences. Outliers can be found as a by-product of clustering algorithms. Also, clustering is a well-established research area and there have been abundant clustering algorithms that users can choose from for performing clustering and then detecting outliers.

Lee et al. [10] also proposed a trajectory clustering framework called TRACLUS, which shares many characteristics with the popular clustering algorithm DBSCAN [3]. DBSCAN is a density-based clustering algorithm: given a set of data, it groups together objects that are close to each other and marks as outliers those whose nearest neighbors are too far away. It is one of the most common clustering algorithms. In both DBSCAN and TRACLUS, distance calculation between objects, in our case trajectories or trajectory segments, should be made in order to find nearest neighbors. As to TRACLUS, a distance function is defined, which is similar to the one in TRAOD and our Definition 5. Using this function, distances between trajectory segments can be calculated and the number of nearest neighbors of each trajectory segment can be obtained. If a trajectory segment has more than a certain number of neighboring trajectory segments, all of its neighboring trajectory segments and itself will be grouped into the same cluster. Thus, by evaluating all trajectory segments and calculating distances between each trajectory segment pair, TRACLUS is able to cluster trajectory segments of small distances. And those who are left out can be considered as outliers. According to its behavior, TRACLUS is able to detect density-based outliers. However, in TRACLUS, the distance function proposed is not a metric since it does not obey the triangle inequality rule. This makes direct application of traditional spatial indexes difficult and is a problem that needs to be solved. In a clustering-based method, if attributes like speed are taken into account, it is able to detect more types of outliers. In TRACLUS and some other clustering algorithms, when clusters are formed already and a new trajectory is introduced, only the distances between the new trajectory and the core of each cluster are calculated. This saves much computation and is obviously an advantage. In addition, compared to classification-based approaches, one does not need to know the number of categories when doing clustering, and there is no need for training or test data sets. But clustering-based methods also have some same weaknesses as in classification-based methods. They, as well, need to define a method to calculate distances between trajectories or trajectory segments, which includes choosing the attributes that matter and choosing a proper distance function. Besides, these clustering-based methods have high computational cost because they need to firstly build clusters. Building clusters, as introduced above, is not easy work and requires a larger number of distance measurements and also the process of finding nearest neighbors.

3.4 Trajectory Similarity

Since an outlying trajectory can be considered as a trajectory that is different from the rest, it can also be detected by measuring similarity between trajectories. One common measure of trajectory similarity is dynamic time

warping (DTW) [18]. The basic idea behind DTW is to find out the *warping path* W between two trajectories that has minimum *warping cost*. The DTW similarity value is that corresponding warping cost. For two trajectories $traj_i$ and $traj_j$, with N_i and N_j location points, an $N_i \times N_j$ matrix M_{ij} can be created where each element $M_{ij}(m,n)$ $(m \in [1,N_i], n \in [1,N_j])$ represents the distance between location points p_m^i and p_n^j. The calculation of distance between location points requires a chosen distance function, for example, Euclidean distance. Then a warping path W can be created by starting at a matrix element $M_{ij}(m,n)$ with $m = 1$ and $n \in [1,N_j]$, and increasing either m or n or both by 1 each step until reaching a matrix element $M_{ij}(m,m)$ with $m = N_i$ and $n \in [1,N_j]$. For example, a path beginning at $M_{ij}(1,1)$ can move to one of $M_{ij}(1,2)$, $M_{ij}(2,1)$ and $M_{ij}(2,2)$. If w_l represents a matrix element $M_{ij}(m,n)_l$, then a warping path W can be represented as the sequence of matrix elements, i.e., $W = w_1 w_2 \dots w_z$. Obviously there exist many possible warping paths. Dynamic programming is one way to find the optimal one. After that, the warping cost can be calculated. This cost value is the DTW similarity between the two trajectories. A common warping cost function is the sum of all of the distances calculated along the warping path. Hence, if a trajectory has small values of similarity with most of the trajectories in the data set, it is probably an outlier.

Using DTW, if the x coordinate, y coordinate, and time stamp of each location point are all taken into account when calculating distances between location points, various types of outliers can be detected, including density, direction, duration, and speed-based outliers. However, not all of those four types of outliers can be detected. Besides, it is easy to notice that the similarity measurement of every trajectory pair is computationally expensive, not to mention the computational complexity when there is a large number of trajectories as input data set. Another weakness of DTW is that noise can greatly affect the performance because every location point of both trajectories must have at least one match when searching for warping paths.

3.5 Summary

From the eight types of TOD methods introduced in the previous sections of this chapter, we get seven representative algorithms, which are TOP-EYE, TRAOD, the one by Knorr et al.,iBAT, ROAM, TRACLUS, and DTW. A summary of these algorithms is given in Table 3.1. This table lists the four types of outliers defined in Section 2.5. It shows what types of outliers each of these TOD methods is able to detect.

TOP-EYE is a combination of density and direction-based method. It is able to detect both direction and density-based outliers. TRAOD and TRACLUS can detect density and direction-based outliers because the dis-

tance function they use includes also the direction information (angular distance) of trajectories. As to some other distance-based methods, like the one proposed by Knorr et al., they are capable of detecting all the four types of outliers if all necessary information is included in the attribute vector of trajectories. This is also true for classification-based methods like ROAM and clustering-based methods like TRACLUS. The sixth method, iBAT, isolates trajectories by examining whether they pass some grid cells that no other trajectories pass. It is able to detect density-based outliers. Lastly, DTW is used to measure similarity between trajectories and thus be able to detect trajectories that are quite different from the rest. If time is taken into account, it is able to detect all the four types of outliers.

Table 3.1: TOD algorithms and different types of outliers

Approaches		Outliers			
		density-based	direction-based	speed-based	duration-based
Dir & Den	TOP-EYE	√	√		
Distance	TRAOD	√	√		
	Knorr	√	√	√	√
Isolation	iBAT	√			
Classification	ROAM	√	√	√	√
Clustering	TRACLUS	√	√	√	√
Similarity	DTW	√	√	√	√

Chapter 4

Algorithm Design

In this chapter two TOD algorithms are introduced. Section 4.1 introduces the grid-based TOD algorithm we propose, namely the OS (Outlying Score) algorithm. Section 4.2 introduces another TOD algorithm design that is based on dynamic time warping (DTW). Some of the notations used in this chapter were previously defined in Chapter 2.

4.1 The OS Algorithm

In this section, the OS algorithm is introduced. Algorithm 2 shows a full version of the algorithm. It is composed of four major steps. During the first step, trajectories are interpolated. In the second step, grid feature information is obtained. Afterwards, the outlying scores (OSs) of all trajectories are calculated. Lastly, trajectories are evaluated by their OSs and determined whether each of them is an outlier. In Section 4.1.1 to 4.1.4, the four major steps are introduced in detail. In Section 4.1.5 the parameters that are necessary for the OS algorithm are summarized.

Algorithm 2 The OS algorithm

Inputs: T - a set of trajectories
$size_g$ - the size of a grid cell
num_{dir} - the number of elements in the direction vector
τ - threshold parameters
ω - weights of features
λ_{os} - outlier threshold

Outputs: OT - a set of outlying trajectories

1: $interpT \leftarrow interpolate(T, size_g)$;
2: $[E, G, Den, dirFreq, \overline{Dur}, \overline{Spd}] \leftarrow calcFeatures(interpT, size_g, num_{dir})$;
3: $OS \leftarrow calculateOS(E, G, Den, dirFreq, \overline{Dur}, \overline{Spd}, \tau, \rho)$;
4: $OT \leftarrow determineOutliers(OS, \lambda_{os}, \omega)$;

Algorithm 3 interpolate

Inputs: T - a set of trajectories
$size_g$ - the size of a grid cell
Outputs: (n_{g_x}, n_{g_y}) - the number of grid cells in a row and a column
$interpT$ - a set of interpolated trajectories

$A = [0, w_A, 0, h_A] \leftarrow$ the minimum bounding box of all trajectories;
$n_{g_x} \leftarrow ceiling(\frac{w_A}{size_g})$; $n_{g_y} \leftarrow ceiling(\frac{h_A}{size_g})$;
for $traj_i \in T$ **do**
    initialize $traj'$ as an empty set;
    for $p_n^i \in traj_i$ **do**
        $(x_n^i, y_n^i, t_n^i) \leftarrow p_n^i$, $(x_{n+1}^i, y_{n+1}^i, t_{n+1}^i) \leftarrow p_{n+1}^i$;
        add $(\frac{x_n^i}{size_g}, \frac{y_n^i}{size_g}, t_n^i)$ to $traj'$;
        if $\lfloor \frac{x_n^i}{size_g} \rfloor \neq \lfloor \frac{x_{n+1}^i}{size_g} \rfloor$ or $\lfloor \frac{y_n^i}{size_g} \rfloor \neq \lfloor \frac{y_{n+1}^i}{size_g} \rfloor$ **then**
            $p_1 \leftarrow (\frac{x_n^i}{size_g}, \frac{y_n^i}{size_g})$, $p_2 \leftarrow (\frac{x_{n+1}^i}{size_g}, \frac{y_{n+1}^i}{size_g})$;
            $p_1^{i\prime}, \cdots, p_m^{i\prime} \leftarrow$ intersection points of p_1p_2 and cell borders;
            for $j \in [1, m]$ **do**
                add $p_j^{i\prime} = (\frac{x_j^{i\prime}}{size_g}, \frac{y_j^{i\prime}}{size_g}, t_n^i + \frac{t_{n+1}^i - t_n^i}{m+1} \cdot j)$ to $traj'$;
            end
        end
    end
    add $traj'$ to $interpT$;
end

4.1.1 Interpolation

Algorithm 3 shows the first major step of the OS algorithm, which is the trajectory-interpolation process. In this process, with input $size_g$, which is the size of a grid cell, each trajectory in the input set T is examined and the minimum bounding box of all trajectories is extracted. Then the number of grid cells in a row and a column is calculated. Therefore, it is possible and also convenient to refer to each grid cell with an integer tuple (a, b). The grid coordinate system was previously introduced in Section 2.2. Also, previously in Section 2.2 the interpolation approach was introduced in detail. During this process, each trajectory in the input set T is examined and location points are inserted whenever the trajectory reaches a cell border. These location points are referred to as *entering points*. During this process, the x and y coordinates of a trajectory are amplified $size_g$ times compared to the original ones because they are mapped to the grid coordinate system.

4.1.2 Calculate Grid Features

After interpolation, the feature information of grid cells is calculated. Algorithm 4 shows how this process works. In this design the following features are taken into account for outlier detection. These were previously defined in Section 2.4.

- *density*; indicates the number of trajectories passing through a grid cell;
- *direction*; indicates the common direction trajectories take when they pass a grid cell;
- *duration*; indicates the general length of time trajectories spent within a grid cell;
- *speed*; indicates the common speed trajectories have when passing through a grid cell;

If only density and direction are taken into consideration, our design will be similar as TOP-EYE [4]. However, all these four features are important for trajectory outlier detection since many outlying behaviors are related to duration and speed.

This feature-calculation process is an important part of the OS algorithm. It consists of three steps. During the first step, for each trajectory $traj_i$, its sequence of entering points E_i and the grid cells it passes through, i.e., G_i, are extracted. It is easy to obtain E_i and G_i for $traj_i$ following Definition 4. Algorithm 5 shows this process in detail. It has a run time of $O(num_{traj} \cdot N_{max})$, where num_{traj} is the number of trajectories in the input set and N_{max} equals to the largest number of location points N_i ($i \in [1 : num_{traj}]$) of trajectories. The second step is shown in Algorithm 6. During this step, for each trajectory, which grid cells it passes through, and its general direction, total duration, and average speed in these grid cells are calculated. These information will update grid feature matrices Den, dir, dur, and spd. This process is straightforward following Definition 6 to 12. It has a run time of $O(num_{traj} \cdot |E|_{max})$, where $|E|_{max}$ equals to the largest number of entering points $|E|_i$ ($i \in [1 : num_{traj}]$) of trajectories. The last step is shown in Algorithm 7. In this step feature matrices dir, dur, and spd are normalized. The direction frequency $dirFreq$ (Definition 9), average duration $\overline{Dur}$ (Definition 11), and average speed $\overline{Spd}$ (Definition 13) of each grid cell are calculated. During this feature-calculation process, all historical trajectory data are transformed to four matrices that store grid features, i.e., Den, $dirFreq$, $\overline{Dur}$, and $\overline{Spd}$. These matrices will be further used for outlier detection. In other words, the algorithm does not need to store all historical trajectory data for further comparison, greatly reducing memory consumption.

Algorithm 4 calcFeatures

Inputs: $interpT$ - a set of interpolated trajectories
num_{dir} - the number of elements in the direction vector
(n_{g_x}, n_{g_y}) - the number of grid cells in a row and a column
Outputs: E - entering points of trajectories
G - grid cells trajectories pass through
Den - grid density matrix
$dirFreq$ - grid direction matrix
$\overline{Dur}$ - grid duration matrix
$\overline{Spd}$ - grid speed matrix

initialize Den, dur, and spd as $n_{g_x} \times n_{g_y}$ zero matrices;
initialize dir as $n_{g_x} \times n_{g_y} \times num_{dir}$ zero matrices;
/* Step 1. get sequence E_i and G_i for each $traj_i$ */
for $traj_i \in interpT$ **do**
    $[E_i, G_i] \leftarrow calcEG(traj_i)$;
end for
/* Step 2. calculate feature information for grid cells */
for $i \in [1, num_{traj}]$ **do**
    $[Den, dir, dur, spd] \leftarrow calcFeatureMatrices(E_i, G_i, Den, dir, dur, spd, num_{dir})$;
end for
/* Step 3. get $dirFreq$, $\overline{Dur}$, and $\overline{Spd}$ */
for $a \in [1, n_{g_x}]$ **do**
    for $b \in [1, n_{g_y}]$ **do**
        $[Den, dirFreq, \overline{Dur}, \overline{Spd}] \leftarrow normFeatureMatrices(a, b, Den, dir, dur, spd, num_{dir})$;
    end for
end for

4.1.3 Calculate Outlying Scores

During the previous steps, grid features are obtained as four matrices. Besides, the sequence E_i and G_i for each trajectory $traj_i$ are extracted. These variables are the inputs in this calculation step. During this process, the outlying scores of trajectories are calculated. Algorithm 8 shows this process. It has a run time of $O(num_{traj} \cdot |E|_{max})$.

For each trajectory $traj_i$, all the grid cells it passes are examined. Four OSs are used to measure the outlier level of $traj_i$. As defined in Definitions 14 to 17, if a grid cell $g_{(a,b)}$ has a low density, which indicates that it is not common to be in that grid, the density OS of $traj_i$ increases. If $traj_i$ has a different direction compared to that of $g_{(a,b)}$, its direction OS increases. If $traj_i$ has a *standing* or *others* status in $g_{(a,b)}$, and has a different duration compared to $g_{(a,b)}$, its duration OS increases. And similarly, if $traj_i$ has a *crossing* or *others* status in $g_{(a,b)}$, and has a different speed compared to $g_{(a,b)}$, its speed OS increases. The definitions of the three statuses *standing*,

Algorithm 5 calcEG

Inputs: $traj_i$ - a trajectory
Outputs: E_i - the sequence of entering points of $traj_i$
G_i - the grid cells $traj_i$ passes through

for $p_n^i \in traj_i$ **do**
  $(x_n^i, y_n^i, t_n^i) \leftarrow p_n^i$, $(x_{n+1}^i, y_{n+1}^i, t_{n+1}^i) \leftarrow p_{n+1}^i$;
  $cnt \leftarrow 0$;
  if p_n^i is an entering point **then**
    $cnt \leftarrow cnt + 1$;
    add (x_n^i, y_n^i, t_n^i) to $E_i(cnt)$;
    if $cnt > 1$ **then**
      add $trajLen$ to $E_i(cnt - 1)$;
    end if
    $trajLen \leftarrow length(p_n^i p_{n+1}^i)$;
    $a_n^i \leftarrow min(x_n^i, x_{n+1}^i)$, $b_n \leftarrow min(y_n^i, y_{n+1}^i)$;
    add (a_n^i, b_n^i) to $G_i(cnt)$;
  else
    $trajLen \leftarrow trajLen + length(p_n^i p_{n+1}^i)$;
  end if
end for

others, and *others* were given in Section 2.5. Following Definitions 14 to 17, there are four necessary parameters τ_{den}, τ_{dir}, τ_{dur}, and τ_{spd}. These threshold parameters are the elements of input τ. Imagine if a grid cell of size $size_g$ is small enough, most probably only the *standing* and *crossing* statuses exist. On the other hand, if $size_g$ is large enough, all three status are possible. Ideally, with a proper choice of $size_g$, only the former two cases are expected, i.e., a trajectory should either have *standing* or *crossing* status in a grid cell. In this case, only one of duration OS and speed OS will increase when evaluating a grid cell of a trajectory.

It is easily noticed that the increase of each OS is related to the time a trajectory spends within a grid cell. This is based on the assumption that the outlier level of a trajectory is related to the time it spends being outlying. Since a duration outlier usually spends more time in a grid cell, a normalization factor $(1 + \tau_{dur})$ is used when increasing its OS. On the other hand, due to the fact that a speed outlier usually spends less time in a grid cell, the increase of speed OS is normalized by a factor $\frac{1}{1+\tau_{spd}}$.

4.1.4 Determine Outliers

Since there exist four OSs based on different features, a question that needs to be answered is how to combine these OSs and make the final decision of outliers. One way is to determine outliers **type by type** and choose

Algorithm 6 calcFeatureMatrices

Inputs: E_i - the sequence of entering points of $traj_i$
G_i - the sequence of grid cells $traj_i$ passes through
num_{dir} - the number of elements in the direction vector
Den - grid density matrix
dir - grid direction matrix
dur - grid duration matrix
spd - grid speed matrix

Outputs: Den - grid density matrix
dir - grid direction matrix
dur - grid duration matrix
spd - grid speed matrix

1: **for** $j \in [1, |E_i| - 1]$ **do**
2: $(x^i_j, y^i_j, t^i_j, trajLen^i_j) \leftarrow E_i(j)$;
3: $(x^i_{j+1}, y^i_{j+1}, t^i_{j+1}, trajLen^i_{j+1}) \leftarrow E_i(j+1)$;
4: $p_1 \leftarrow (x^i_j, y^i_j)$, $p_2 \leftarrow (x^i_{j+1}, y^i_{j+1})$;
5: $angle \leftarrow$ the angle from $(0,0)$ to vector $\overrightarrow{p_1p_2}$;
6: $dir^i_j \leftarrow round(num_{dir} \cdot \frac{angle}{2\pi})$;
7: $dur^i_j \leftarrow t^i_{j+1} - t^i_j$;
8: $spd^i_j \leftarrow \frac{trajLen^i_j}{dur^i_j}$;
9: $Den(G_i(j)) \leftarrow Den(G_i(j)) + 1$;
10: $dir(G_i(j), dir^i_j) \leftarrow dir(G_i(j), dir^i_j) + 1$;
11: $dur(G_i(j)) \leftarrow dur(G_i(j)) + dur^i_j$;
12: $spd(G_i(j)) \leftarrow spd(G_i(j)) + spd^i_j$;
13: **end for**

the union of all those outliers as the output set OT. In this way, following Definition 14 to 17, four outlier thresholds, i.e., λ_{den}, λ_{dir}, λ_{dur}, and λ_{spd}, are necessary. Another solution would be a weighted sum function as follows.

$$sumOS_i = \omega \cdot OS_i$$

where $OS_i = (OSDen_i, OSDir_i, OSDur_i, OSSpd_i)$, and the weight vector $\omega = (\omega_{den}, \omega_{dir}, \omega_{dur}, \omega_{spd})$ is set by users. The sum of all elements in ω should be 1, i.e., $\omega_{den} + \omega_{dir} + \omega_{dur} + \omega_{spd} = 1$. With this solution only one threshold λ_{os} is needed. This solution is shown in Algorithm 9. Since this solution can be easily transformed to the type-by-type method by setting the corresponding weight to 1 and the other three to 0, the type-by-type method will not be shown in detail. With a weighted sum function, different types of outliers can be treated differently. Users can set different weights for these OSs according to applications and their own needs. This is an advantage of this algorithm as, for instance, in some applications density outliers are big

Algorithm 7 normFeatureMatrices

Inputs: num_{dir} - the number of elements in the direction vector
(a, b) - the coordinates of a grid cell
Den - grid density matrix
dir - grid direction matrix
dur - grid duration matrix
spd - grid speed matrix

Outputs: Den - grid density matrix
$dirFreq$ - normalized grid direction matrix
$\overline{Dur}$ - normalized grid duration matrix
$\overline{Spd}$ - normalized grid speed matrix

1: $num \leftarrow Den(a, b)$;
2: **for** $d \in [1, num_{dir}]$ **do**
3: $\quad dirFreq(a, b, d) \leftarrow \frac{dir(a,b,d)}{num}$;
4: **end for**
5: $\overline{Dur}(a, b) \leftarrow \frac{dur(a,b)}{num}$;
6: $\overline{Spd}(a, b) \leftarrow \frac{spd(a,b)}{num}$;

concerns while in other places duration outliers are a more serious problem. Also, this design can be easily transformed to an algorithm that is similar to TOP-EYE when ω_{den} and ω_{dir} are set to 0.5 and ω_{dur} and ω_{spd} are set to 0.

After summing up all the four OSs by a weighted sum function, the algorithm sorts all trajectories by $sumOS$ in descending order. As shown in Algorithm 9, this design provides three methods to determine outliers by $sumOS$. With the **first method**, λ_{os} is set to an actual $sumOS$ value and is used to differentiate between abnormal and normal trajectories directly. This method is shown in Figure 4.1(a). In this figure, the trajectories with $sumOS$ larger than λ_{os} are classified as outliers. For the **second method**, λ_{os} is set to a slope value. When all $sumOS$ values are sorted in descending order, it is expected that there is an obvious distinction between the $sumOS$ values of abnormal trajectories and those of normal ones, like the case shown in Figure 4.1(b). In such a case, the threshold can be set to the point where the slopes of two consecutive lines change greatly. In Figure 4.1(b), the threshold is set to point A since there is a great difference between the two slopes at that point. In Chapter 5 some experiments that apply this method are explained in detail. However, in real situations, this method might not work for larger data sets. For a larger data set, it is easy to have two (abnormal) trajectories with very similar $sumOS$ values. In such situations, it is hard to determine the correct threshold. As shown in Figure 4.1(c), although the slope changes the most at point A, it is not proper to set the threshold at that point because the trajectories that have

Algorithm 8 calculateOS

Inputs: num_{traj} - the number of input trajectories
num_{dir} - the number of elements in the direction vector
E - entering points of trajectories
G - grid cells trajectories pass through
Den - grid density matrix
$dirFreq$ - grid direction matrix
$\overline{Dur}$ - grid duration matrix
$\overline{Spd}$ - grid speed matrix
τ - threshold parameter

Outputs: OS - outlying scores of trajectories

for $i \in [1, num_{traj}]$ **do**
  initialize $denOS_i$, $dirOS_i$, $durOS_i$, and $spdOS_i$ as zeros;
  for $j \in [1, |E_i| - 1]$ **do**
    $(x_j^i, y_j^i, t_j^i, trajLen_j^i) \leftarrow E_i(j)$;
    $(x_{j+1}^i, y_{j+1}^i, t_{j+1}^i, trajLen_{j+1}^i) \leftarrow E_i(j+1)$;
    $trajDur = t_{j+1}^i - t_j^i$;
    if $Den(G_i(j)) < \tau_{den}$ **then**
      $denOS_i \leftarrow denOS_i + trajDur$;
    end
    $p_1 \leftarrow (x_j^i, y_j^i)$, $p_2 \leftarrow (x_{j+1}^i, y_{j+1}^i)$;
    $angle \leftarrow$ the angle from $(0, 0)$ to vector $\overrightarrow{p_1 p_2}$;
    $dir_j^i \leftarrow round(num_{dir} \cdot \frac{angle}{2\pi})$;
    $dirDiff \leftarrow 1 - \sum_{k=1}^{num_{dir}} f_k(G_i(j)) \cdot cos(dir_j^i - d_k)$;
    if $dirDiff > \tau_{dir}$ **then**
      $dirOS_i \leftarrow dirOS_i + dirDiff \cdot trajDur$;
    end
    if $traj_i$ has *standing* or *others* status in $G_i(j)$ **then**
      if $\frac{|trajDur - \overline{Dur}(G_i(j))|}{\overline{Dur}(G_i(j))} > \tau_{dur}$ **then**
        $durOS_i \leftarrow durOS_i + \frac{trajDur}{1+\tau_{dur}}$;
      end
    else if $traj_i$ has *crossing* or *others* status in $G_i(j)$ **then**
      if $\frac{|(\frac{trajLen_j^i}{trajDur}) - \overline{Spd}(G_i(j))|}{\overline{Spd}(G_i(j))} > \tau_{spd}$ **then**
        $spdOS_i \leftarrow spdOS_i + trajDur \cdot (1 + \tau_{spd})$;
      end
    end
  end
  $OS_i \leftarrow (denOS_i, dirOS_i, durOS_i, spdOS_i)$;
end

Table 4.1: Parameters of the OS algorithm

Parameter	Remark
$size_g$	the size of a grid cell
num_{dir}	the number of directions in the direction vector
$\tau = (\tau_{den}, \tau_{dir}, \tau_{dur}, \tau_{spd})$	determine when an OS should increase
$\omega = (\omega_{den}, \omega_{dir}, \omega_{dur}, \omega_{spd})$	weights of OSs
λ_{os}	differentiate abnormal trajectories from normal ones

slightly smaller $sumOS$ values than $traj_A$ are also possible outliers. To avoid such kind of problems, this design provides a **third method** that uses a cumulative distribution function (CDF) to determine outliers automatically. Firstly it obtains the probability density function (PDF) $f_{sumOS}(x)$ of the $sumOS$ values of all trajectories. Some example PDF graphs are shown in Chapter 5. A PDF graph of $sumOS$ values is used to specify the probability of a random $sumOS$ value falling within a particular range of values. With a PDF graph, it is easy to obtain the corresponding CDF $F_{sumOS}(x)$. Therefore, users can set the outlier threshold λ_{os} to a percentage number. If the user would like to pay more attention to the top $\eta\%$ trajectories that have the largest $sumOS$ values, λ_{os} can be set to $\eta\%$. Then, by solving the equation $F_{sumOS}(x) = 1 - \lambda_{os}$, the trajectories with $sumOS$ values that are larger than or equal to x are determined as outliers. The advantage of the second and third method is that users do not need to know the exact values of $sumOS$ or what these values mean in essence. These three methods of determining outliers can be easily applied to the type-by-type method. Users only needs to change $sumOS$ to a certain type of OS and λ_{os} to one of λ_{den}, λ_{dir}, λ_{dur}, and λ_{spd}.

4.1.5 Parameters

The parameters that need to be set for the OS algorithm are summarized in Table 4.1.

The first important parameter is the size of a grid cell $size_g$. As mentioned in Subsection 4.1.3, having a relatively small cell size can solve the co-relation problem of duration and speed OSs. However, one should also notice that having a small cell size not only increases computational complexity but also makes the outlier detection susceptible to noisy data. Generally, the size of the minimum bounding box of all trajectories, the precision of localization devices, and how precise the user want the detection to be should be taken into account when choosing a value for $size_g$. In Chapter 5 some examples of this parameter setting are shown.

Next, users need to set the number of elements in the direction vector, i.e., num_{dir}. If num_{dir} is too small, for example, $num_{dir} = 4$, a trajectory that follows a direction that is less than $\frac{\pi}{4}$ different from the normal ones

Algorithm 9 determineOutliers

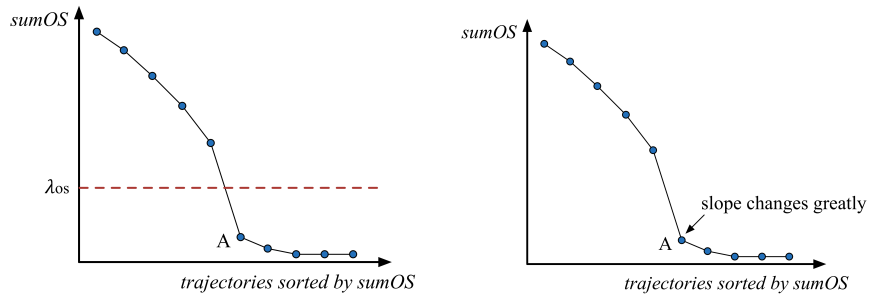
Inputs: num_{traj} - the number of input trajectories
OS - outlying scores of trajectories
ω - weight parameter
λ_{os} - outlier threshold

Outputs: OT - a set of outlying trajectories

```
1: for i ∈ [1, num_traj] do
2:     sumOS_i ← ω_den·denOS_i + ω_dir·dirOS_i + ω_dur·durOS_i + ω_spd·spdOS_i;
3: end for
4: sort trajectories by sumOS in descending order;
5: choose a method from Method #1, #2, and #3;
6: /* Method #1 */
7: for i ∈ [1, num_traj] do
8:     if sumOS_i > λ_os then
9:         add traj_i to OT;
10:    end if
11: end for
12: /* Method #2 */
13: for i ∈ [1, num_traj − 1] do
14:    add traj_i to OT;
15:    slope_i ← sumOS_i − sumOS_{i+1};
16:    if i > 1 then
17:        slope_{i−1} ← sumOS_{i−1} − sumOS_i;
18:        if |slope_{i−1} − slope_i| > λ_os then
19:            return;
20:        end if
21:    end if
22: end for
23: /* Method #3 */
24: obtain the probability density function f_sumOS(x) of sumOS;
25: obtain the corresponding cumulative distribution function F_sumOS(x);
26: solve equation F_sumOS(x) = 1 − λ_os;
27: for 1 ∈ [1, num_traj] do
28:    if sumOS_i ≥ x then
29:        add traj_i to OT;
30:    end if
31: end for
```

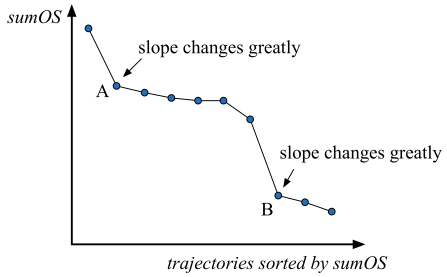
will not be detected as its direction will be rounded to the normal direction. In this design, following TOP-EYE [4], num_{dir} is set to 8 by default.

The third parameter is a vector of four elements. These elements are used to separate abnormal trajectories from normal ones in different aspects. The



(a) Method 1. determine outliers by a threshold of an actual *sumOS* value.

(b) Method 2. determine outliers by the difference between two consecutive slopes of the *sumOS* graph.



(c) An example where the slope threshold cannot be set correctly.

Figure 4.1: Methods and problems of determining outliers.

increase of each OS is related to parameter τ_{den}, τ_{dir}, τ_{dur}, or τ_{spd}. These parameters indicate the level where a trajectory is considered outlying in each of the four aspects. The settings of these parameters should depend on applications. For example, when the algorithm is applied in a library during exam weeks, where the flowrate of people is large, τ_{den} should be larger than when the algorithm is applied during holiday time. Also it should be noticed that τ_{dir} has a range of [0,1] since the function $F(Dir^{(a,b)})$ in Definition 15 has a range of [0,1].

The fourth parameter, weight ω, is also a vector of four elements. As mentioned in Section 4.1.4, the elements in ω are tuned by users according to applications and their own needs. There might be situations where, for example, the values of duration OS are still much larger than the other three after normalization. In such kind of situations, ω_{dur} can be set as a smaller value to degrade the influence of duration OS on *sumOS*.

Lastly, the fifth parameter λ_{os} is used as the threshold of *sumOS* that separates abnormal trajectories from normal ones. λ_{os} can be set to different types of values according to user preference. If users would like to detect outliers type by type, four thresholds λ_{den}, λ_{dir}, λ_{dur}, and λ_{spd} are

necessary. In this case λ_{os} is a vector of four elements. If users would like to apply a weighted sum function, when applying the first method introduced in Subsection 4.1.4, λ_{os} is set to an actual $sumOS$ value. Since the $sumOS_i$ value of a trajectory $traj_i$ is closely related to the time of $traj_i$ spends being outlying, if the user wants all trajectories that spend more than $t_{outlying}$ time duration being outlying to be detected as outliers, λ_{os} can be roughly set to $t_{outlying}$. For the second method, λ_{os} is set to a slope value. For this method, it is easy to tune a proper λ_{os} automatically when there is an obvious distinction between abnormal and normal trajectories. However, in real-life situations, due to the great diversity of trajectory data, this might not happen. In such situations, users can choose the third method where λ_{os} is set to a percentage number.

In Definition 18 and 19, another two parameters ρ_{spd} and ρ_{len} were mentioned. In this design these two parameters have their default setting. $\rho_{spd} = 0.5$ since it is assumed that a normal pedestrian will not have a walking speed that is smaller than a half of spd_{norm}. If a trajectory simply passes across a grid cell, it should have a length that is smaller than $\sqrt{2} \cdot size_g$ if it moves straight. ρ_{len} is set as 1.2 since it is assumed that a trajectory that simply passes across a grid cell should not have a length that is larger than $1.2\sqrt{2} \cdot size_g$ in that grid cell.

4.2 Dynamic Time Warping

Another TOD algorithm that is based on Dynamic Time Warping (DTW) is designed as a comparison of the OS algorithm.

4.2.1 The Algorithm

As introduced in Section 3.4, DTW relies on matching location points of trajectories. The DTW similarity value is the total of all of the distances calculated along the optimal warping path. The method introduced by Toohey et al. [18] matches location points by calculating Euclidean distances in the spatial domain, i.e., only the x and y coordinates of location points are considered. This method performs well with trajectories of different lengths and even widely varying sampling rates. However, with this method, some expected direction, duration and speed outliers will not be detected if they have the same shape as the other normal ones. Therefore, in our version, location points are matched by calculating the Euclidean distance in both the temporal and the spatial domains, i.e., the time stamps of location points are also taken into account. When doing so, it should be noticed that now the sampling rate has an influence on trajectory similarity. The sum of all Euclidean distances for each trajectory is used to determine outliers. The run time of this algorithm is $O(num_{traj} \cdot N_{max}^2)$. This indicates that the DTW algorithm is more computationally expensive than the OS algorithm.

Algorithm 10 The DTW algorithm

Inputs: T - a set of trajectories
λ_{dtw} - threshold parameter
Outputs: OT - a set of outlying trajectories

for $i \in [1, num_{traj}]$ **do**
  for $j \in [i, num_{traj}]$ **do**
    /* Step 1. generate distance matrix */
    for $m \in [1, N_i]$ **do**
      for $n \in [1, N_j]$ **do**
        $M_{ij}(m, n) \leftarrow distance(p_m, p_n)$;
      end for
    end for
    /* Step 2. calculate warping cost */
    $m \leftarrow 1$;
    $n \leftarrow$ index of $min(M_{ij}(1, :))$;
    $cost_{ij} \leftarrow 0$;
    while $m \leq N_i$ **do**
      $[cost_{ij}, m, n] \leftarrow calcCost(cost_{ij}, m, n, M_{ij})$;
    end while
    $sumDTW_i \leftarrow sumDTW_i + cost_{ij}$;
    $sumDTW_j \leftarrow sumDTW_j + cost_{ij}$;
  end for
end for
/* Step 3. determine outliers */
$OT \leftarrow determineOutliers(sumDTW, \lambda_{dtw})$;

Algorithm 10 shows the DTW algorithm. The algorithm has three major steps. During the first step, for each trajectory pair, a distance matrix is generated. In this matrix, each element is the distance of a location-point pair. In the second step, for each distance matrix, the warping path that has the smallest sum of distances is found. This warping path should pass every location point at least one time. The sum of distances is the DTW similarity value of that trajectory pair. In this design, the optimal warping path is found by greedy programming, as shown in Algorithm 11. The third step is the same as Algorithm 9. In this function, all the DTW similarity values of each trajectory $traj_i$ are summed up as $sumDTW_i$. Afterwards, outliers are determined by $sumDTW$ values and a threshold λ_{dtw}.

Algorithm 11 calcCost

Inputs: $cost_{ij}$ - the warping cost of trajectory pair $traj_i$ and $traj_j$
m - a point index of $traj_i$
n - a point index of $traj_j$
M_{ij} - the distance matrix of $traj_i$ and $traj_j$

Outputs: $cost_{ij}$ - the warping cost of trajectory pair $traj_i$ and $traj_j$
m - a point index of $traj_i$
n - a point index of $traj_j$

$minDis = min(M_{ij}(m+1,n), M_{ij}(m,n+1), M_{ij}(m+1,n+1))$
$cost_{ij} \leftarrow cost_{ij} + minDis$;
if $minDis == M_{ij}(m+1,n)$ **then**
    $m \leftarrow m+1$;
else if $minDis == M_{ij}(m,n+1)$ **then**
    $n \leftarrow n+1$;
else
    $m \leftarrow m+1$;
    $n \leftarrow n+1$;
end if

Table 4.2: Parameters of the DTW algorithm

Parameter	Remark
λ_{dtw}	differentiate abnormal trajectories from normal ones

4.2.2 Parameters

In this algorithm, only an outlier threshold parameter is needed, as shown in Table 4.2. Similarly as how λ_{os} is set, a threshold λ_{dtw} is set to determine outliers and can be an actual $sumDTW$ value, a slope value, or a percentage number.

Chapter 5

Experiments

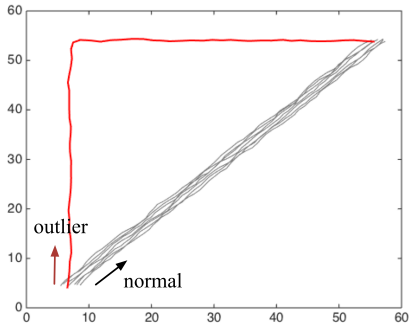
This chapter describes the experiments we have done to test and compare different TOD algorithms. Section 5.1 introduces the first experiment, in which there are six small data sets of simulated trajectories. Section 5.2 introduces the second experiment, which is conducted on a real-life trajectory data set gathered by an omni-directional camera in a lab [16]. In Section 5.3 an experiment on another real-life trajectory data set gathered by smart phones is introduced. This is a trajectory data set of people moving around in a relatively large outdoor area in The Netherlands [19].

5.1 Simulated Trajectories

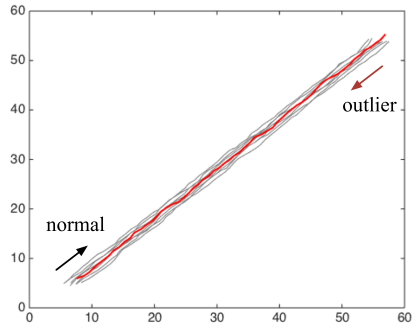
In this experiment, the OS and DTW algorithms are applied on a couple of simulated trajectory data sets. The setup, experimental result, and analysis of this experiment will be introduced in this section.

5.1.1 Setup

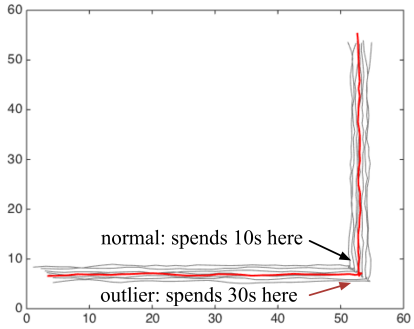
Four small sets of simulated trajectories are generated as benchmarks of the TOD algorithms. Each of these data sets involves one of the four types of outliers defined in Section 2.5. The goal of this experiment is to test whether the algorithms are capable of detecting these four types of outliers correctly in simple cases. In each of the small data sets eight trajectories are generated, one of which is a certain type of outlier. It is assumed that these trajectories are within an area of 60 meters by 60 meters and $spd_{norm} = 1.4\ meters/seconds$. In the first set, there is one density outlier. Similarly, the second, third, and fourth data set have a direction, duration, and speed outlier, respectively. These sets of trajectories are shown in Figure 5.1. In Figure 5.1(a) a density outlier is shown, which follows a different route compared to the other seven normal ones. Figure 5.1(b) shows a direction outlier that follows an opposite direction compared to the other seven



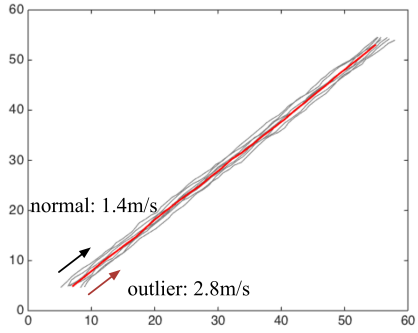
(a) One density outlier $traj_a$.



(b) One direction outlier $traj_b$.



(c) One duration outlier $traj_c$.

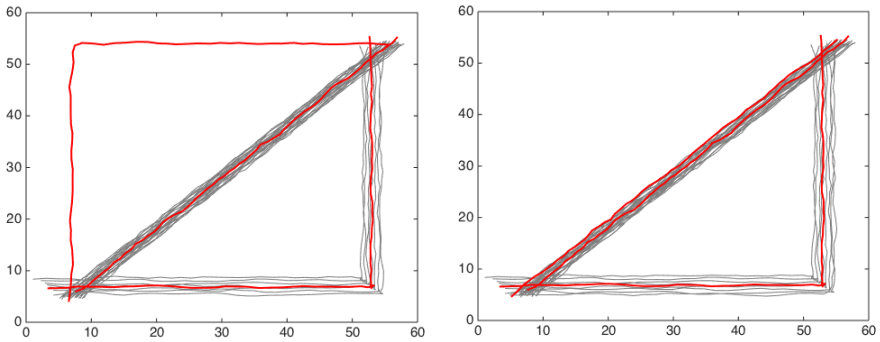


(d) One speed outlier $traj_d$.

Figure 5.1: A visualization of simulated trajectory data set I, II, III, and IV.

trajectories. In Figure 5.1(c) there is a duration outlier that follows the same path but spends more time at the lower right corner. And In Figure 5.1(d) there exists a speed outlier that has an average speed of 2 times faster than the others. For convenience, these four sets are referred to as set I-Den, II-Dir, III-Dur, and IV-Spd, respectively, and the corresponding outlying trajectories in each of them are referred to as $traj_a$, $traj_b$, $traj_c$, and $traj_d$.

Furthermore, to test whether the algorithms work when there exist more than one type of outliers in a single data set, the four small data sets are combined to generate the fifth data set named set V. This set accordingly has 32 trajectories, as shown in Figure 5.2(a). In this figure, 23 trajectories move diagonally and 9 move across a corner. It is obvious that $traj_a$, $traj_b$, $traj_c$, and $traj_d$ in this combined data set are still outlying. Lastly, another data set, set VI, is generated similarly as set V. The only difference is that $traj_a$ in the fifth data set is rotated for 180 degrees around the center of the 60m×60m area. This rotated trajectory $traj'_a$ follows the same direction as



(a) Trajectory set V. (b) Trajectory set VI.

Figure 5.2: A visualization of simulated trajectory data set V and VI. In each of them there are 32 trajectories and four of them are outliers.

$traj_c$. This data set is shown in Figure 5.2(b). In this figure the rotation of trajectory $traj_a$ is still an outlier. While the other 8 trajectories that follow the same path as $traj'_a$ all stop for a while at the lower-right corner, $traj'_a$ simply goes across that place. Thus, $traj'_a$ is an expected speed outlier.

The parameter setting of the experiments on the simulated trajectories is shown in Table 5.1. The size of a grid cell is set to 5 meters, which is a reasonable value considering that the size of the whole area is $60m \times 60m$. The cell size can also be set to other values. An experiment on the cell size will be introduced in Subsection 5.1.3. When generating data sets I to VI, it was assumed that a person is 0.5 meter in width and the distance between two people is at least 0.5 meter. Therefore, if there exists a group of people who pass through a grid cell ($size_g = 5m$) together, the number of people in that group is at most 5. In these data sets, normal people always move in groups while the density outliers always move alone. Therefore, for these experiments, if τ_{den} is set to a value that is larger than 5, normal trajectories will also have large density OSs. And since the value of τ_{den} should be at least 2 as no trajectory will pass through a grid cell with density less than 1, we choose 3 as the value of τ_{den}. According to Definition 15, the difference between the general direction of a trajectory and the direction of a grid cell is within a range of $[0, 1]$. Thus, τ_{dir} is set to the average of 0 and 1, which is 0.5. τ_{dur} is set to 0.5 since we would like to detect the trajectories that spend more than 1.5 times the normal duration at some area. And τ_{spd} is also set to 0.5 because we would like to detect the trajectories that move with speeds that are more than 1.5 times the normal speed. The weights of the four features are set to be equal since in these experiments all four types of outliers are treated equally. Lastly, if the first method of determining outliers by $sumOS$ is chosen and it is assumed that a trajectory that spends more than 1 minute being outlying is an outlier, λ_{os} is roughly set to 1. If

the second method is chosen, λ_{os} should be set to a slope value. With this method, we are capable of determining a proper λ_{os} value $\lambda_{os} = 3$ that works for data sets I to V, but this does not work for data set VI. The detailed explanation will be shown in Subsection 5.1.2. If it is chosen to apply the third method and set λ_{os} to a percentage number, λ_{os} is set to 12.5% since we would like to detect the one outlier from every eight trajectories. As to the setting of λ_{dtw}, since an individual $sumDTW$ value does not reflect the outlier level of a trajectory, it is hard to determine a proper value for $sumDTW$ if the first method is chosen. It makes more sense to apply the second and third method to determine outliers for the DTW algorithm. For the second method, when λ_{dtw} is set to 40, it works for data sets I to IV. However, it is impossible to determine the correct outliers in sets V and VI by the second method for this algorithm. The reasons for that will be explained in Subsection 5.1.2. For the third method, λ_{dtw} is also set to 12.5%. This works for data sets I to V.

Table 5.1: Parameter setting of the experiments on data set I to VI

Parameter	Value
$size_g$	5 meters
dir_{num}	8
$(\tau_{den}, \tau_{dir}, \tau_{dur}, \tau_{spd})$	(3,0.5,0.5,0.5)
$(w_{den}, w_{dir}, w_{dur}, w_{spd})$	(0.25,0.25,0.25,0.25)
λ_{os}	1 for Method 1; 12.5% for Method 3
λ_{dtw}	12.5% for Method 3

5.1.2 Results and Discussion

In this subsection, firstly an analysis of the experiments on sets I-Den, II-Dir, III-Dur, and IV-Spd will be presented, followed by an analysis of the experiments on sets V and VI. The three methods that are used to determine outliers by $sumOS$ or $sumDTW$ will also be analyzed for each sub-experiment.

Sets I, II, III, and IV

Table 5.2 and 5.3 show the $sumOS$ and $sumDTW$ values of all the trajectories in data sets I-Den, II-Dir, III-Dur, and IV-Spd. It can be seen that the expected outlying trajectory in each of these sets, which are trajectory $traj_a$, $traj_b$, $traj_c$, and $traj_d$, have much larger $sumOS$ and $sumDTW$ values than the other trajectories in the same set. Thus, both the OS and DTW algorithm are capable of detecting them as outliers. Since the normal trajectories in each of these sets have similar $sumOS$ and $sumDTW$ values while the outliers have larger $sumOS$ and $sumDTW$ values, all the three

Table 5.2: The *sumOS* values of all trajectories in data sets I to IV.

Trajectory	*sumOS*			
	Set I-Den	Set II-Dir	Set III-Dur	Set IV-Spd
$traj_1$	0	0	0	0
$traj_2$	0	0	0	0
$traj_3$	0	0	0	0
$traj_4$	0	0	0	0
$traj_5$	0	0	0	0
$traj_6$	0	0	0	0
$traj_7$	0	0	0	0
$traj_o$ (o is a, b, c, or d)	13.05	10.31	5.63	10.86

Table 5.3: The *sumDTW* values of all trajectories in data sets I to IV.

Trajectory	*sumDTW*			
	Set I-Den	Set II-Dir	Set III-Dur	Set IV-Spd
$traj_1$	159.29	260.76	70.24	127.68
$traj_2$	151.86	249.09	58.17	119.29
$traj_3$	147.82	248.56	51.67	111.30
$traj_4$	152.47	245.77	48.78	110.70
$traj_5$	159.14	249.20	52.56	109.42
$traj_6$	163.70	251.73	59.64	113.97
$traj_7$	174.12	265.47	71.03	123.19
$traj_o$ (o is a, b, c, or d)	620.02	1517.58	161.79	593.38

methods that are used to determine outliers by *sumOS* or *sumDTW* are applicable on these data sets.

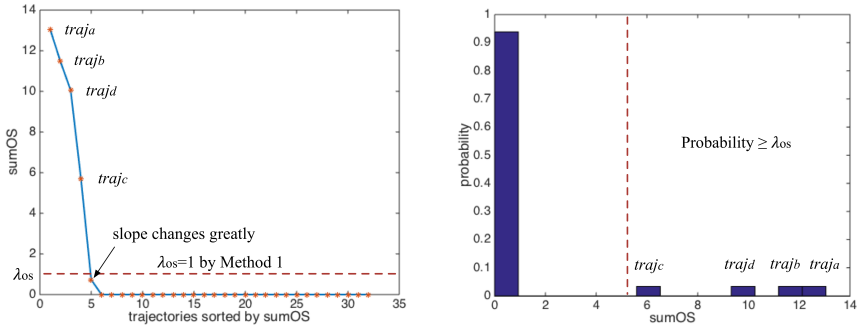
In set I-Den, $traj_a$ has a nonzero $sumOS_a$ because most of the grid cells it passes through have a small density. $sumOS_a$ is actually the density OS. In set II-Dir, the value of $sumOS_b$ comes from the fact that $traj_b$ follows an uncommon direction in all the grid cells it passes through. It is actually the direction OS. In set III-Dur, $traj_c$ has a larger *sumOS* value because it has a different time duration in one of the grid cells it passes through. This value is in fact the duration OS. And lastly in set IV-Spd, $traj_d$ has a nonzero value of *sumOS* because it has an uncommon speed in all the grid cells it passes through. $sumOS_d$ is actually the speed OS of $traj_d$. Since the four outlying trajectories $traj_a$, $traj_b$, $traj_c$, and $traj_d$ are outlying in different aspects and they spend different time duration being outlying, they have different *sumOS* values. If $traj_a$, $traj_b$, and $traj_c$ move with normal speeds and $traj_d$ has a speed that is twice the normal speed, it can be inferred from Figure 5.1 that: (i) $traj_a$ spends the most time being outlying, (ii) $traj_b$ and $traj_c$ spends less outlying time duration, and (iii) $traj_d$ spends the least time duration being outlying. Since *sumOS* is related to the time

duration a trajectory spends being outlying, $traj_a$ has the largest $sumOS$ value, $traj_b$ and $traj_d$ have smaller $sumOS$ values, and $traj_c$ has the smallest $sumOS$. The three methods

As to the $sumDTW$ values of these trajectories, since all the normal trajectories in each set have similar movements, they have similar warping costs when matching location points, and thus similar $sumDTW$ values. With regards to the outlying trajectories, it is easy to infer that the warping cost of $traj_a$ should be larger than the others since it takes a completely different route. For $traj_b$, since it follows an opposite direction compared to the others, the Euclidean distances between its location points and their matching location points is large. In addition, when a location point of $traj_b$ finds its optimal matching point in the spatial domain, the difference between their time stamps will be large. On the contrary, when it finds its optimal matching point in the temporal domain, the spatial distance between it and its matching point will become large. Thus, the distances between the trajectories in set II are the largest among all the four data sets. For $traj_c$, it has a larger $sumDTW$ because it has more location points at the lower-right corner. More location points results in a larger $sumDTW$ value since all location points should have at least one matching point of the other trajectories. Furthermore, since all the 8 trajectories in set III take the same route and direction, except that $traj_c$ spends more time at the lower-right corner, the trajectories in this set have smaller distances between each other compared to the other sets. Lastly, since $traj_d$ has a faster speed than the others, it has fewer location points as all trajectories are generated by the same sampling rate. However, the Euclidean distances between its location points and their matching location points is large. Therefore, $traj_d$ has a large $sumDTW$ value as well.

Sets V and VI

Figure 5.3 shows the result of the OS algorithm applied on data set V. In Figure 5.3(a) $traj_a$, $traj_b$, $traj_c$, and $traj_d$ have much larger $sumOS$ values than the other trajectories in set V. Same as in sets I, II, III, and IV, the nonzero $sumOS$ values of these four outlying trajectories in this set come from density OS, direction OS, duration OS, and speed OS, respectively. It can be seen from this figure that there is a fifth trajectory that has a nonzero $sumOS$ value. It was determined that this trajectory takes the route that is the same as $traj_c$. Its nonzero $sumOS$ value is due to the fact that it spends a little more time than the others at the lower-right corner. Since this different is small enough, the value of $sumOS$ is also small. This trajectory is not detected as an outlier. For this figure, it can be noticed that if the first method is applied and $\lambda_{os} = 1$, all outliers can be correctly detected. Since the slope changes greatly at the point after $traj_c$, when the second method is chosen, the algorithm is also capable of detecting the



(a) The $sumOS$ values of the trajectories in set V sorted in descending order.

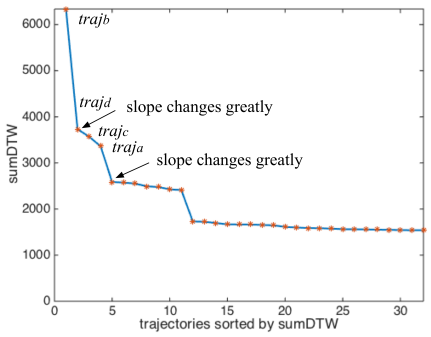
(b) PDF graph of $sumOS$.

Figure 5.3: Experimental results of the OS algorithm applied on set V.

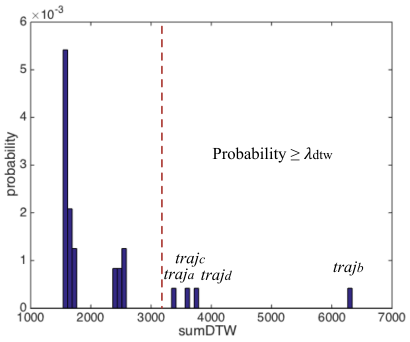
correct outliers. In Figure 5.3(b) the PDF of $sumOS$ is shown. In this figure there are 14 classes, each class with a width of 0.93. When the third method is chosen and λ_{os} is set to 12.5%, the separation between abnormal and normal trajectories is made at $sumOS = 5.59$. The trajectories that have $sumOS$ values larger than 5.59, i.e., $traj_a$, $traj_b$, $traj_c$, and $traj_d$, are thus detected as outliers.

In terms of the DTW algorithm, Figure 5.4(a) shows the $sumDTW$ values of all trajectories. It can be seen from this figure that if the first method is applied, with a proper threshold, for example $\lambda_{dtw} = 3000$, $traj_a$ to $traj_d$ can all be correctly detected as outliers. In this figure the 7 trajectories after $traj_c$ have very similar $sumDTW$ values and the last 21 trajectories also have very similar $sumDTW$ values. It was determined that the 7 trajectories are the ones that follow the same path as $traj_c$. Although these trajectories have small warping costs with the trajectories that take the same route, since more trajectories move diagonally, they have larger warping costs with more trajectories. Thus, they have larger $sumDTW$ values. On the other hand, since the last 21 trajectories move diagonally, they have smaller warping costs to more trajectories. Therefore, they have smaller $sumDTW$ values. It is easily noticed that the slope changes greatly at point $traj_d$ and also the point after $traj_a$. By manual checking, the difference between the two slopes at point $traj_d$ is the largest. Therefore, the DTW algorithm is not capable of detecting all correct outliers with the second method. Figure 5.4(b) show the PDF graph of $sumDTW$. In this PDF graph there are 100 classes, each with a width of 74.98. When setting λ_{dtw} to 12.5%, the separation between abnormal and normal trajectories is made at $sumDTW = 3339$. Thus, $traj_a$ to $traj_d$ can all be correctly detected by the third method.

However, when it comes to data set VI, the DTW algorithm has bad performance in terms of detecting $traj'_a$ while the OS algorithm still works

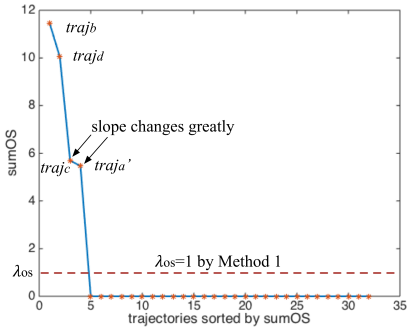


(a) The *sumDTW* values of the trajectories in set V sorted in descending order.

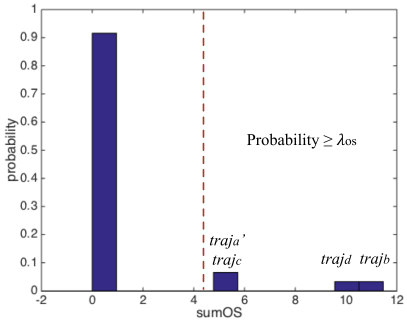


(b) PDF graph of *sumDTW*.

Figure 5.4: Experimental results of the DTW algorithm applied on set V.



(a) The *sumOS* values of the trajectories in set VI sorted in descending order.

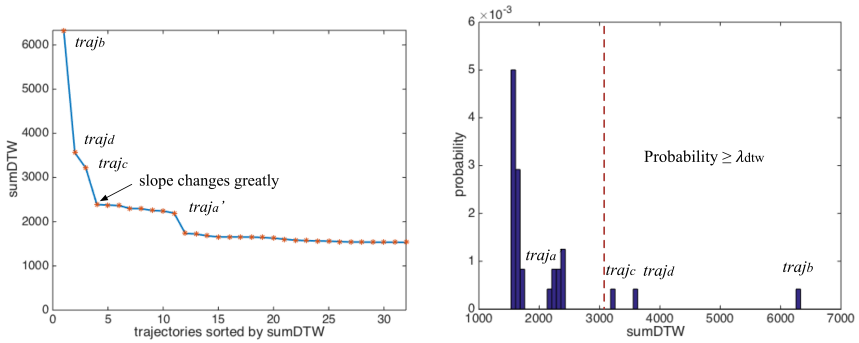


(b) PDF graph of *sumOS*.

Figure 5.5: Experimental results of the OS algorithm applied on set VI.

well. As shown in Figure 5.5(a), $traj'_a$, $traj_b$, $traj_c$, and $traj_d$ have much larger *sumOS* values than the other trajectories in set VI. They can all be detected by the OS algorithm with $\lambda_{os} = 1$ by the first method. However, it can be noticed that the slope changes greatly at both point $traj_c$ and $traj'_a$. These two changes are larger than the change at the point after $traj'_a$. Therefore, if the second method is chosen, the determine-outlier process of the OS algorithm does not work for this data set. Figure 5.5(b) shows the PDF graph of *sumOS*. In this figure there are 12 classes, each class with a width of 0.96. With the third method of determining outliers, if λ_{os} is set to 12.5%, the separation between abnormal and normal trajectories is made at $sumOS = 4.78$. The trajectories that have *sumOS* values larger than this value, i.e., $traj'_a$, $traj_b$, $traj_c$, and $traj_d$, are all detected as outliers.

Figure 5.6(a) shows the *sumDTW* values of all trajectories. In this figure,



(a) The *sumDTW* values of the trajectories in set VI ranked in descending order.

(b) PDF graph of *sumDTW*.

Figure 5.6: Experimental results of the DTW algorithm applied on set VI.

Table 5.4: Comparison of the three methods of determining outliers

	Method #1 (actual sum value)		Method #2 (slope)		Method #3 (PDF)	
	OS	DTW	OS	DTW	OS	DTW
set I to IV	√	√	√	√	√	√
set V	√	√	√		√	√
set VI	√				√	

although $traj_b$, $traj_c$, and $traj_d$ have the top three *sumDTW* values, $traj'_a$ ranked 11th in this set. Therefore, the DTW algorithm is not capable of detecting $traj'_a$ as an outlier. The reason for that is many trajectories follow the same path as $traj'_a$, making it easy to match the location points of $traj'_a$ to those of the other trajectories. It may seem strange that $traj_c$ can be detected while $traj'_a$ cannot. The reason is that $traj_c$ has more location points at the lower-right corner because it spends more time at that place. Since all location points should have at least one matching point of the other trajectories, more location points results in a larger warping cost and thus a larger *sumDTW* value. The PDF graph is also shown in Figure 5.6(b). With $\lambda_{dtw} = 12.5\%$, the algorithm only detects $traj_b$, $traj_c$, and $traj_d$ as outliers.

Table 3.1 shows a summary of the performance of the three methods applied on data sets I to VI. A tick (√) indicates that a method is applicable on the a certain data set. From this table, it is easily noticed that Method 1 and 3 works for all data sets. Since a percentage number is easier to understand and setting Method 3 is a more automatic way of determining outliers, users can choose to tune thresholds with this method.

Table 5.5: Run time (seconds) of the OS and DTW algorithm on the simulated data

Data set	OS	DTW
I	0.34	36.49
V	1.25	674.54

Run Time

The computation time of the OS and DTW algorithms over 100 runs are recorded in Table 5.5. The algorithms are run by MATLAB on a MacBook Pro with 2.7 GHz Intel Core i5 and 8GB main memory. Since data sets I, II, III, and IV have the same number of trajectories and these trajectories have similar numbers of location points, only the run times of the experiments on data set I are shown. For the same reason, only the run times of the experiments on data set V are shown as a representative of data sets V and VI. It can be seen from this table that the OS algorithm has much smaller computation times on these data sets than the DTW algorithm does.

To sum up, from these experimental results, the conclusion can be drawn that both the OS and DTW algorithms are capable of detecting density, direction, duration, and speed outliers in some situations. However, due to the fact that the DTW algorithm does a comparison of all the trajectories in the input set, it fails to detect some temporal-spatial outliers if the outliers behave abnormally at only small areas. On the other hand, since the OS algorithm does comparison of neighboring trajectories, it has a high efficiency in such cases.

5.1.3 The Size of Grid Cells

To study the influence of the size of a grid cell on the outlier detection by the OS algorithm, a small experiment is conducted on data set V. In this experiment, parameter $size_g$ is changed. Since the setting of τ should be consistent with $size_g$, some elements in τ are also changed as $size_g$ changes. The values of $size_g$ and τ as well as the detection results are shown in Table 5.6. In this table *Fail* indicates that with the corresponding parameters the algorithm is not capable of detecting the correct outliers, and *Succeed* indicates that the algorithm is capable of detecting the correct outliers.

As mentioned in Subsection 5.1.1, τ_{den} is at least 2, and if τ_{den} is larger than 5, normal trajectories in set V will have non-zero density OSs. Therefore, the choice of τ_{den} is from 2 to 5. Following the experiments introduced in this section, $\tau_{den} = 3$ works for all six data sets with $size_g = 5m$. Actually, experimental results show that $\tau_{den} \in [2, 5]$ give the same results for sets I to VI with $size_g = 5m$. Therefore, we arbitrarily set τ_{den} to 3 when $size_g = 5m$. When the cell size is smaller, τ_{den} is also set to a smaller value

Table 5.6: The influence of the size of grid cell on the outlier detection by the OS algorithm on set V.

$size_g$	τ	Performance
3m	(2,0.5,0.5,0.5)	Fail
4m	(2,0.5,0.5,0.5)	Succeed
5m	(3,0.5,0.5,0.5)	Succeed
10m	(3,0.5,0.5,0.5)	Succeed
30m	(3,0.5,0.5,0.5)	Succeed
35m	(3,0.5,0.5,0.5)	Fail

since it is assumed that a smaller area will have a smaller number of trajectories passing through it. Experimental results also show the algorithm does not work when $size_g$ is set to 3m or 4m and $\tau_{den} = 3$. Thus, when $size_g$ is set to 3m or 4m, τ_{den} is set to a smaller value. In fact, the detection of density outliers is influenced the most when the cell size is decreasing. When the cell size is too small, more normal trajectories will be detected as density outliers. On the other hand, the detection of all four types of outliers is affected when the cell size is increasing. When the cell size is large enough, hardly any outlier can be detected. When the cell size becomes larger, the trajectories within a same grid cell probably have large distances with each other. In such a case, the algorithm not only compares a trajectory with the ones that are adjacent to it, but also the ones that actually lie far away from it. Therefore the algorithm will not perform well. For this data set, when cell size is too small, the algorithm fails to detect the density outlier even though τ_{den} is also set to a smaller value. Thus the algorithm performs badly. On the other hand, when the cell size is too large, it fails to detect the duration outlier. Since the duration outlier $traj_c$ only has an outlying behavior in a very small area, its duration in a grid cell will not be considered abnormal if the cell size is large enough.

Therefore, a proper choice of cell size is necessary. The choice of a cell size usually depends on applications. Users should take the minimum bounding box of all trajectories, computation time, and detection precision into account when setting this parameter. Among the four types of outliers, density outliers are the most obvious ones and can probably be determined by a manual checking on historical data. Usually when there exist density outliers in the input trajectory set, users can tune $size_g$ by doing trail experiments to detect the density outliers. $size_g$ should be set to a value where the expected density outliers can be correctly detected and no normal trajectories are falsely detected as outliers. This gives the lower and upper bounds for $size_g$. Afterwards, users can choose a value from the certain range by additionally taking computation time and detection precision into account. If there is no density outlier in the input trajectory set, the cell size

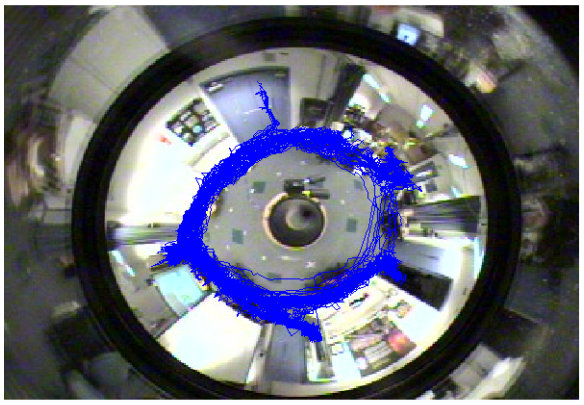


Figure 5.7: Trajectories in a lab captured by an omni-directional camera [16].

should be set to a value where normal trajectories are not falsely detected as outliers. This gives the lower bound for $size_g$. Afterwards, users can take both computation time and detection precision into account and set a proper $size_g$.

5.2 Trajectories in a Lab

The second experiment is conducted on a real-life trajectory data set [16], as shown in Figure 5.7. In this set there are 209 trajectories. It is a data set of humans walking through a lab captured using an omni-directional camera. For convenience, we refer to this data set as LabOmni. In the following parts of this section, the setup, the experimental results, and analysis of this experiment will be introduced.

5.2.1 Setup

The parameters set for this experiment are shown in Table 5.7. In this experiment, the size of a grid cell is set to 10 pixels. Since we do not know how large this lab is, considering that the image captured by the camera is approximately $150pixel \times 150pixel$, a cell size of 10 pixel is considered reasonable. In this data set the number of trajectories is larger than the former experiment, and we also want to detect the trajectories that exist in some low-density areas, thus τ_{den} is set to 8. It is also found out that in this lab visitors usually spend more time at different places, thus duration OS is easy to increase with a small τ_{dur}. In this experiment, τ_{dur} is set to a value that is larger than in Subsection 5.1.1: $\tau_{dur} = 1.5$. Also, in this lab visitors have various speeds. τ_{spd} in this experiment is set a little

Table 5.7: Parameter setting of the experiments on the LabOmni data.

Parameter	Value
$size_g$	10 pixels
dir_{num}	8
$(\tau_{den}, \tau_{dir}, \tau_{dur}, \tau_{spd})$	(8,0.5,1.5,0.7)
$(w_{den}, w_{dir}, w_{dur}, w_{spd})$	(0.25,0.25,0.25,0.25)
λ_{os}	18% for Method 3
λ_{dtw}	18% for Method 3

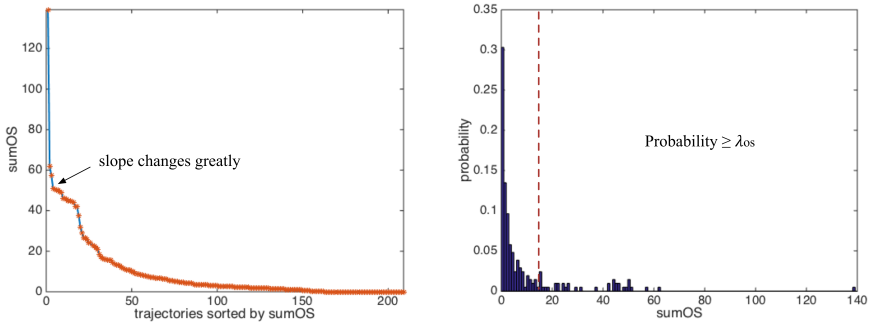
higher than Subsection 5.1.1. If the first method is chosen and it is assumed that a trajectory that spends 10 time stamps being outlying is an outlier, λ_{os} can be set to 10. If the second method is chosen, it is not possible to make a correct distinction between abnormal and normal trajectories. The reasoning will be explained in Subsection 5.2.2. When applying the third method, at first λ_{os} was set to be the same as in Subsection 5.1.1, which is 12.5%. However, it was determined that some obvious density outliers are not detected. Thus, λ_{os} was tuned to a larger value 18%.

5.2.2 Results and Discussion

Figure 5.8 shows the results of the $sumOS$ values of the trajectories in set LabOmni. The trajectories are sorted by their $sumOS$ values in descending order. In Figure 5.8(a), the point where slope changes greatly is not a proper threshold since many of the trajectories after this point are also possible outliers. For this data set, it is more proper to apply the third method for determining outliers, where λ_{os} is set to a percentage number. Figure 5.8(b) shows the PDF graph of $sumOS$. In this figure there are 140 classes, each with a width of 0.99. When λ_{os} is set to 18%, the distinction between abnormal and normal trajectories is formed at $sumOS = 12.92$. The top 39 trajectories that have larger $sumOS$ values than 12.92 are detected as outliers. It can be seen from this figure that the division is made in an area where trajectories have similar $sumOS$ values, i.e., there is no obvious distinction between abnormal and normal trajectories. For such kinds of data set, users should try different values for λ_{os} to make sure that a proper separation between abnormal and normal trajectories is made.

Figure 5.9 shows the $sumDTW$ values by the DTW algorithm. For comparison, λ_{dtw} is also set to 18% and the third method for determining outliers is applied. Figure 5.9(b) shows the PDF graph of $sumDTW$. In this graph there are 250 classes, each with a width of 942. When $\lambda_{dtw} = 18\%$, the separation between abnormal and normal trajectories is made at $sumDTW = 44,406$. Thus 41 trajectories are detected as outliers.

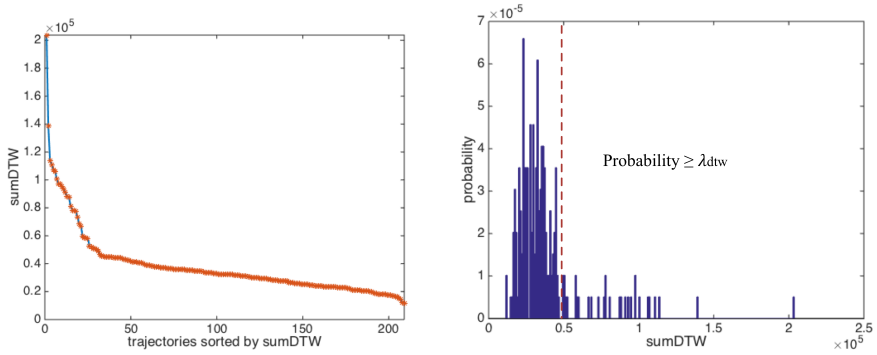
The 39 trajectories detected by the OS and the 41 detected by the DTW



(a) The *sumOS* values of the trajectories in data set LabOmni.

(b) PDF graph of *sumOS*.

Figure 5.8: Experimental results of the OS algorithm applied on set LabOmni.

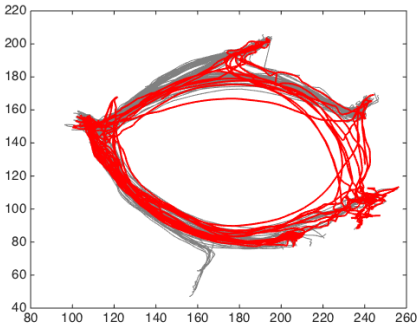


(a) The *sumDTW* values of the trajectories in data set LabOmni.

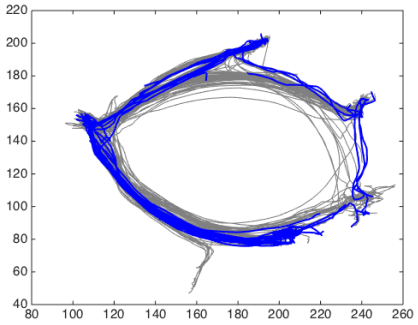
(b) PDF graph of *sumDTW*.

Figure 5.9: Experimental results of the DTW algorithm applied on set LabOmni.

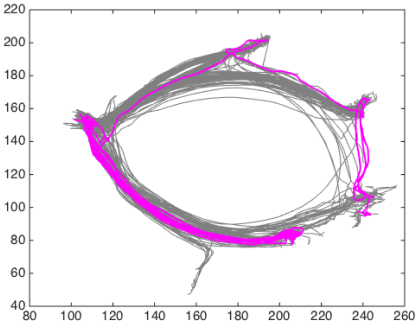
algorithm are shown in Figure 5.10(a) and 5.10(b), respectively. The intersection of these two sets of trajectories is shown in Figure 5.10(c). By checking the 39 trajectories in Figure 5.10(a) in detail, it was determined that they truly have outlying movements in one or more aspects from density, direction, duration, and speed. From these figures, it can be noticed that the DTW algorithm fails to detect some obvious density and duration outliers while the OS algorithm does well. By manual checking, the number of these outliers is 15. In this larger data set, each trajectory needs to be matched to all the other 208 trajectories. In other words, they are compared with not only the trajectories that are adjacent to them, but also those that have completely different routes. Because of the great diversity of move-



(a) The outliers that are detected by the OS algorithm.



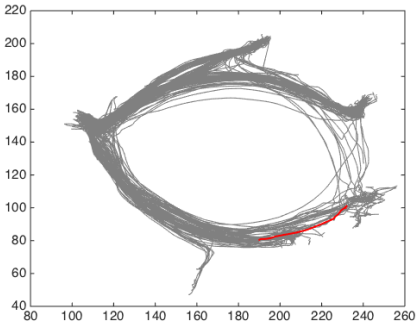
(b) The outliers that are detected by the DTW algorithm.



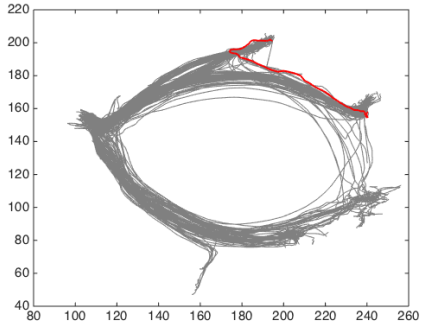
(c) The outliers that can be detected by both the OS and DTW algorithm.

Figure 5.10: A comparison of the outliers that are detected by the OS algorithm and those that are detected by the DTW algorithm.

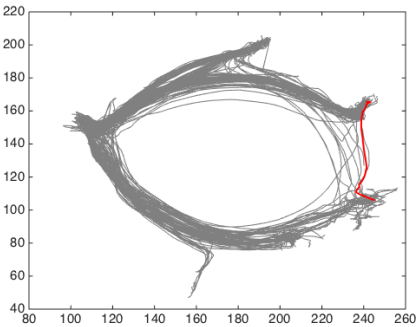
ments in this data set, it is hard to find out which trajectories are different or abnormal by matching location points and accumulating warping costs. By looking into the top 41 trajectories with larger $sumDTW$ values, it was determined that 19 of them are normal in all density, direction, duration and speed aspects. Figure 5.11 shows some representative trajectories that are detected by the DTW algorithm. The trajectories in Figure 5.11(a) and 5.11(b) are short and located at relatively sparse areas. This leads to larger warping costs when matching their location points to those of the others. Thus, they are classified as false positive (FP) outliers. The trajectory in Figure 5.11(c) is correctly detected as an outlier because it is truly an outlier that appears at a low-density area. Regarding the trajectory in Figure 5.11(d), it can be detected by the DTW algorithm because it spends more time around its right endpoint. This brings a larger warping cost when matching location points.



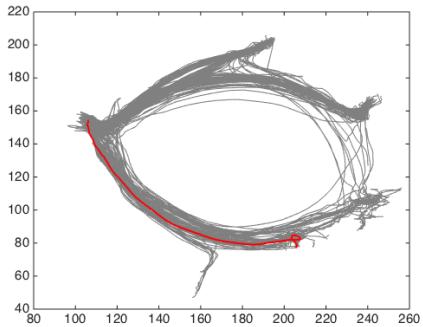
(a) A representative FP outlier detected by the DTW algorithm.



(b) Another representative FP outlier detected by the DTW algorithm.



(c) A representative true density outlier detected by the DTW algorithm.



(d) A representative true duration outlier detected by the DTW algorithm.

Figure 5.11: Some representative outliers detected by the DTW algorithm.

Above all, since the OS algorithm does comparison of neighboring trajectories while the DTW algorithm does that of the entire trajectory set, the OS algorithm is not affected by the distribution of trajectories. In other words, as long as a trajectory is not a density outlier, it is only compared with its adjacent trajectories. Besides, the detection of the OS algorithm is not affected by the length of trajectories. As to the DTW algorithm, although it is able to detect some density and duration outliers that are similar as the ones in Figure 5.11(c) and 5.11(d), it fails to detect some other density, duration, and speed outliers.

The computation times of the OS and DTW algorithms shown in Table 5.8 also indicates that the OS algorithm has a much smaller computation time than the DTW algorithm. The run-time values in this table are obtained over 10 runs.

Table 5.8: Run time (second) of the OS and DTW algorithm on LabOmni

Data set	OS	DTW
LabOmni	0.61	29,373.36

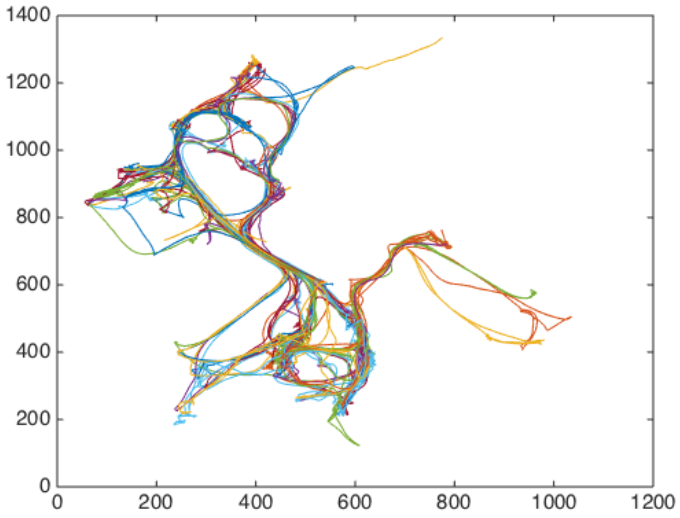


Figure 5.12: Trajectories of the Mysteryland event.

5.3 Trajectories at an Outdoor Event

In this section, the last experiment is introduced. It is an experiment on another real-life trajectory data set. The trajectories in this data set are from a large-scale two-day dance event called Mysteryland [19]. During this event, there were various performances in different zones. The whole event was within an area of approximately 1km by 1km.

A subset of this data set is extracted as our experimental data, as shown in Figure 5.12. The reason why we decide not to use the entire data set as the input trajectory data set lies in the following aspect. It was of great chance that people would move according to their preferences on different performances. And since the performances took place at different time periods of a whole day, the movements of these trajectories are time-dependent. Therefore, it makes no sense to do comparison of the trajectories that appear at different time periods. The subset consists of 34 trajectories that appear between 14pm and 18pm of the first day. For convenience, this data set is referred to as the ML data set.

Table 5.9: Parameter setting of the experiments on the ML data set.

Parameter	Value
$size_g$	10 meters
dir_{num}	8
$(\tau_{den}, \tau_{dir}, \tau_{dur}, \tau_{spd})$	(2,0.5,3,0.5)
$(w_{den}, w_{dir}, w_{dur}, w_{spd})$	(0.25,0.25,0.25,0.25)
λ_{os}	10% for Method 3
λ_{dtw}	10% for Method 3

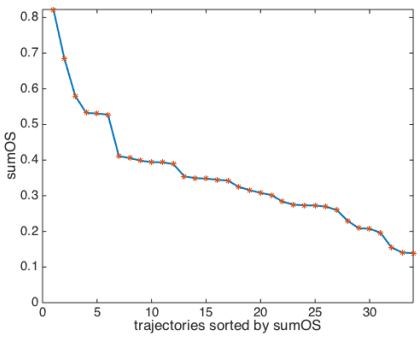
5.3.1 Setup

The parameter set for this experiment is shown in Table 5.9. Since the trajectories in this data set are within a large area, the size of a grid cell is set to be larger than in Section 5.1: $size_g$ is set to 10m. However, there are only 34 trajectories within the target time period, which is a small number of historical data considering that the whole area is large, τ_{den} is set to a small number. It is also assumed that in the Mysteryland area visitors would spend much time staying still when they were enjoying a performance. Thus, τ_{dur} is set to a larger value than in Section 5.1. Lastly, the third method for determining outliers is chosen and λ_{os} and λ_{dtw} are both set to 10%.

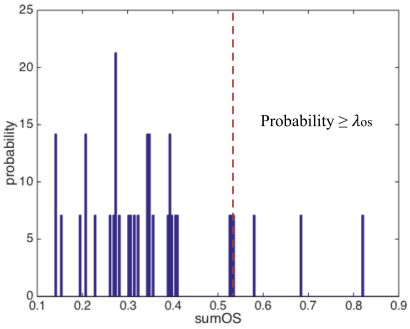
5.3.2 Results and Discussion

Figure 5.13 shows the results of the OS algorithm applied on the ML data set. In Figure 5.13(b) the PDF graph has 180 classes, each with a width of 0.004. The separation between abnormal and normal trajectories is made at $sumOS = 0.53$ with $\lambda_{os} = 10\%$. In this way 4 trajectories are detected as outliers. Similarly, Figure 5.13 shows the results of the DTW algorithm. In the PDF graph there are 300 classes, each with a width of 0.1. When λ_{dtw} is set to 10%, the separation between abnormal and normal trajectories is made at $sumDTW = 9.9$. Also, 4 trajectories are detected as outliers.

The two algorithms show quite different results on this data set. Only one trajectory is detected by both the algorithms. The trajectories that have the largest and third largest $sumOS$ are shown in Figure 5.15(a) and 5.15(b), respectively. By analyzing these trajectories manually, it was determined that these trajectories pass many areas with very low density. They should be detected as density outliers. However, the DTW algorithm fails to do so. The trajectory that has the second largest $sumOS$ is shown in 5.15(c). This trajectory has a large direction OS. Thus it might be a direction outlier. The trajectory shown in Figure 5.16(a) is the only trajectory that is detected by both the algorithms. By manual checking, it is truly a duration outlier. The trajectory that has the highest $sumDTW$ value is shown in Figure 5.16(b). This trajectory is thought of as a false positive (FP) outlier by manual

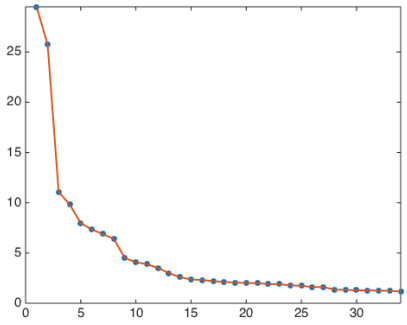


(a) The *sumOS* values of the trajectories in data set ML.

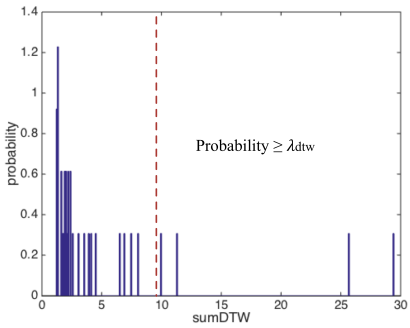


(b) PDF graph of *sumOS*.

Figure 5.13: Experimental results of the OS algorithm applied on set ML.



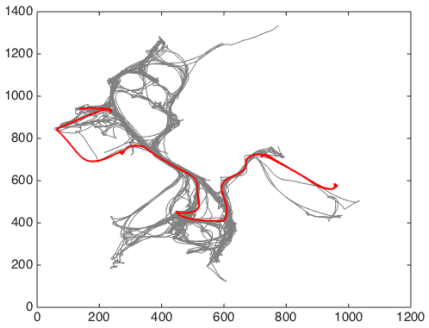
(a) The *sumDTW* values of the trajectories in data set ML.



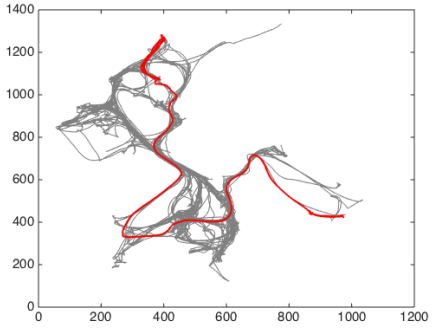
(b) PDF graph of *sumDTW*.

Figure 5.14: Experimental results of the DTW algorithm applied on set ML.

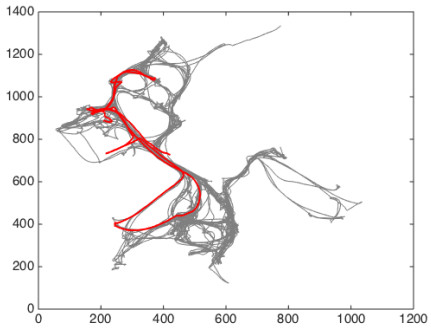
checking. The reason why the DTW algorithm determines this trajectory as an outlier is that it appears only at the upper area. This brings larger warping costs when matching its location points to the others. For the same reason, the trajectory shown in Figure 5.16(c) is another FP outlier detected by the DTW algorithm. The three trajectories in Figure 5.16 have the same property: they appear only in the upper or lower area. And since the DTW algorithm does comparison of all trajectories in the input set, they are easily detected as outliers. The trajectory in Figure 5.16(a) is not FP as it is a duration outlier. Therefore, for this data set, the OS algorithm shows great advantages since it does comparison of neighboring trajectories when examining each trajectory.



(a) A density outlier that is detected by both algorithms.



(b) A density outlier that is only detected by the OS algorithm.



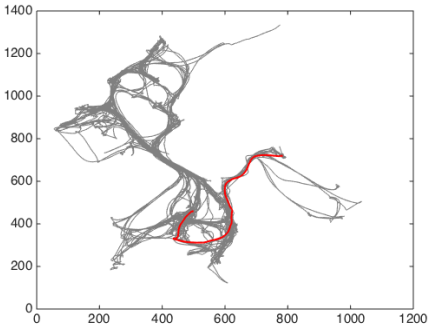
(c) Another direction outlier that is only detected by the OS algorithm.

Figure 5.15: A comparison of the trajectories that are detected by the OS algorithm and those that are detected by the DTW algorithm.

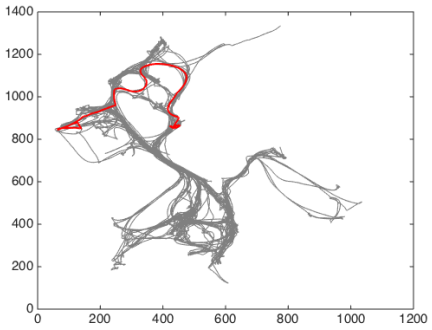
Table 5.10: Run time (seconds) of the OS and DTW algorithm on ML data

Data set	OS	DTW
ML	0.10	27,510.91

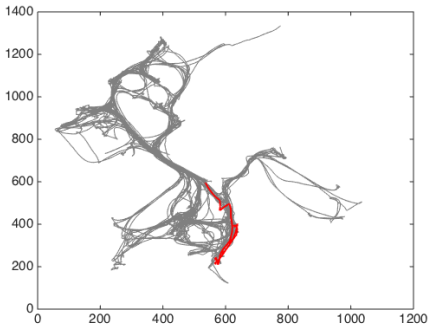
The run time of this experiment is shown in Table 5.10. The values in this table are obtained over 10 runs. Same as in the first two experiment, the run time of the DTW algorithm is much larger than that of the OS algorithm.



(a) A duration outlier that is detected by both algorithms.



(b) A FP outlier that is detected by the DTW algorithm.



(c) A FP outlier that is detected by the DTW algorithm.

Figure 5.16: Some representative trajectories detected by the DTW algorithm.

5.4 Discussion

In this section several experiments on both generated and real-life trajectory data are reported. Generally, both the OS and DTW algorithms are capable of detecting (some) density, direction, duration, and speed outliers. When the trajectory input set is more comprehensive, the OS algorithm performs better. According to the above experiments and analysis, the major advantages of the OS algorithm over the DTW algorithm are: (i) it has locality advantage, in other words, it is capable of detecting outlying behaviors in small areas, (ii) and it has a low run-time complexity. One drawback of the OS algorithm is the necessity of setting several parameters. Without proper values of those parameters the algorithm will not work well. Thus, users need to understand the meaning of each parameter. Another important aspect of the OS algorithm is the need for valid and meaningful

historical trajectory data. It can be assumed that more useful historical data means more accurate grid feature information, and thus more accurate outlier detection.

Although no experiment is conducted on the other existing TOD algorithms introduced in Chapter 3, some theoretical analysis is made here. It is easy to infer that iBAT can only detect the density outliers in these data sets, and TOP-EYE, TRAOD, and TRACLUS will fail to detect the duration and speed outliers. The algorithm by Knorr may fail to detect the outliers that only have outlying behaviors in small areas since it considers each trajectory as a whole. With the framework provided by ROAM, if the attribute vector includes all the four feature information, it is possible to detect many of the four types of outliers with a proper classifier. However, it may have a large computational complexity.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this work, a novel trajectory outlier detection (TOD) algorithm is proposed, which is motivated by the fact that outlying trajectories are possibly caused by harmful behaviors and may bring security threats. The algorithm is gridbased and has a similar detecting strategy as TOP-EYE [4]. However, TOP-EYE is not capable of detecting any duration or speed outliers. A trajectory interpolation approach that provides much convenience for grid-based outlier detection is designed. After interpolation, every location point of a trajectory can be mapped to a grid cell. Then, density, direction, duration, and speed features of trajectories can be extracted. With such a method, it is easy to compare a trajectory with its neighboring trajectories in all the four aspects with efficiency. Four corresponding outlying scores (OSs) are used to measure the outlier level of trajectories. For each of the four aspects, when a trajectory was determined to be different from most of its neighbors in a grid cell, its OS increases. Lastly, a weighted sum function is used to combine the four OSs into a single value $sumOS$. The weight of each OS can be adjusted according to application and user preference. Afterwards, the trajectories with larger $sumOS$ values will be detected as outliers. The design also provides an automated strategy for determining outliers by $sumOS$. During this automatic process, firstly, all $sumOS$ values are smoothed by a smoothing function and sorted in descending order. Then, the tangent values at each point of the smoothed $sumOS$ curve are calculated. Since the tangent values are also expected to be in descending order, the outlier threshold can be set as a tangent value. When a tangent value on the $sumOS$ curve reaches the threshold, the outlier-determining process is finished. The trajectories that have larger or the same smoothed $sumOS$ values than/as the threshold will be determined as outliers.

For reference, another TOD algorithm based on DTW (Dynanmic Time Warping) was designed. Some experiments on both simulated and real data sets were conducted with the two algorithms. Based on the experimental results, we conclude that the OS algorithm performs better in many situations, especially in terms of detecting duration and speed outliers and the outliers that exist within small areas. Since the OS algorithm compares of the trajectories that are adjacent to each other while the DTW algorithm compares all input trajectories, the DTW algorithm fails to detect some density and duration outliers because some other trajectories with small length or located at relatively sparse areas will have larger DTW values. As such these trajectories will be falsely classified as outliers (false positives). However, as the feature information of grid cells is extracted from historical data, more meaningful historical data means more accurate feature information. Therefore, when applying the OS algorithm, enough historical trajectory data is a necessity. Besides, more parameters are needed, including a proper choice of cell size, parameters for increasing OSs, and an outlier threshold.

Theoretical analysis is also made on other relevant TOD algorithms. Based on either the theoretical or the experimental analysis of these TOD algorithms, we conclude that the OS algorithm we propose has advantages over the other TOD algorithm. Compared to previous TOD algorithms, the algorithm has a low computational complexity, a small memory consumption, and is capable of detecting the four types of temporal-spatial outliers and the outliers that appear within small areas.

6.2 Future Work

In the future, this work can be broaden in several aspects. Firstly of all, one could think of conducting experiments on other state-of-the-art TOD algorithms and compare the experimental results with the ones of the OS algorithm. Secondly, one could exploit other information such as the structure of public places for anomalous trajectories detection. Furthermore, the proposed algorithm can be extended to a practical trajectory outlier detection system. Lastly, since detecting anomalous trajectories when the trajectory is ongoing is also important for security, it would be interesting to develop a TOD algorithm that is able to deal with ongoing trajectories.

Bibliography

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[2] Chao Chen, Daqing Zhang, Pablo Samuel Castro, Nan Li, Lin Sun, Shijian Li, and Zonghui Wang. iboat: Isolation-based online anomalous trajectory detection. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):806–818, 2013.

[3] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[4] Yong Ge, Hui Xiong, Zhi-hua Zhou, Hasan Ozdemir, Jannite Yu, and Kuo Chu Lee. Top-eye: Top-k evolving trajectory outlier detection. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1733–1736. ACM, 2010.

[5] Edwin M Knorr, Raymond T Ng, and Vladimir Tucakov. Distance-based outliers: algorithms and applications. *The VLDB JournalThe International Journal on Very Large Data Bases*, 8(3-4):237–253, 2000.

[6] Longin Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. Outlier detection with kernel density functions. *Machine Learning and Data Mining in Pattern Recognition*, pages 61–75, 2007.

[7] Rikard Laxhammar. *Anomaly detection in trajectory data for surveillance applications*. PhD thesis, Örebro universitet, 2011.

[8] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. Trajectory outlier detection: A partition-and-detect framework. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 140–149. IEEE, 2008.

[9] Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez. Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering. *Proceedings of the VLDB Endowment*, 1(1):1081–1094, 2008.

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

[11] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 34. ACM, 2008.

[12] Xiaolei Li, Jiawei Han, and Sangkyum Kim. Motion-alert: automatic anomaly detection in massive moving objects. In *International Conference on Intelligence and Security Informatics*, pages 166–177. Springer, 2006.

[13] Xiaolei Li, Jiawei Han, Sangkyum Kim, and Hector Gonzalez. Roam: Rule-and motif-based anomaly detection in massive moving object data sets. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 273–284. SIAM, 2007.

[14] Siyuan Liu, Lionel M Ni, and Ramayya Krishnan. Fraud detection from taxis' driving behaviors. *IEEE Transactions on Vehicular Technology*, 63(1):464–472, 2014.

[15] Elio Masciari. Trajectory outlier detection using an analytical approach. In *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*, pages 377–384. IEEE, 2011.

[16] Brendan Morris and Mohan Trivedi. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 312–319. IEEE, 2009.

[17] Claudio Piciarelli, Christian Micheloni, and Gian Luca Foresti. Trajectory-based anomalous event detection. *IEEE Transactions on Circuits and Systems for video Technology*, 18(11):1544–1554, 2008.

[18] Kevin Toohey and Matt Duckham. Trajectory similarity measures. *SIGSPATIAL Special*, 7(1):43–50, 2015.

[19] Tim van Oijen. Real-time data collection at mysteryland. `https://www.tudelft.nl/en/ceg/research/stories-of-science/real-time-data-collection-at-mysteryland/`, September 2016.

[20] Guan Yuan, Shixiong Xia, Lei Zhang, Yong Zhou, and Cheng Ji. Trajectory outlier detection algorithm based on structural features. *Journal of Computational Information Systems*, 7(11):4137–4144, 2011.

[21] Daqing Zhang, Nan Li, Zhi-Hua Zhou, Chao Chen, Lin Sun, and Shijian Li. ibat: detecting anomalous taxi trajectories from gps traces. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 99–108. ACM, 2011.

[22] Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):29, 2015.