ADVANCED TECHNIQUES IN TRAJECTORY DATA ANALYSIS FOR ANOMALY DETECTION AND MAP CONSTRUCTION

Yuejun Guo

Universitat de Girona

DOCTORAL THESIS

Advanced Techniques in Trajectory Data Analysis for Anomaly Detection and Map Construction

Author:
YUEJUN GUO

2020

Universitat de Girona

DOCTORAL THESIS

Advanced Techniques in Trajectory Data Analysis for Anomaly Detection and Map Construction

Author:
YUEJUN GUO

2020

Doctoral Programme in Technology

Advisors:
Dr. ANTON BARDERA REIG
Dra. MARTA FORT MASDEVALL
Tutor:
Dra. IMMACULADA BOADA OLIVERAS

A thesis submitted to Universitat de Girona in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Universitat de Girona

Dr Anton Bardera Reig and Dra. Marta Fort Masdevall, of University of Girona,

WE DECLARE:

That the thesis titles *Advanced Techniques in Trajectory Data Analysis for Anomaly Detection and Map Construction*, presented by Yuejun Guo to obtain a doctoral degree, has been completed under our supervision.

For all intents and purposes, we hereby sign this document.

[illegible] [illegible]

Signature

Girona, 5th December 2019

谨以此书纪念逝去的外公外婆，献给亲爱的父母。

纵然伤心，也不要愁眉不展，因为你不知道谁会爱上你的笑容。

Never frown, even when you are sad because you never know who may be falling in love with your smile.

Nunca frunzas el ceño, incluso cuando estés triste, porque nunca sabes quién puede enamorarse de tu sonrisa.

No arruguis mai el front, ni tan sols quan estiguis trist, perquè no se sap mai qui es pot estar enamorant del teu somriure.

Agraïments

En primer lloc, vull agrair als meus supervisors Anton Bardera i Marta Fort la seva orientació pel meu treball de recerca. Agraeixo a Rodrigo I. Silveira, de la Universitat Politècnica de Catalunya, que ha fet suggeriments útils i interessants sobre la investigació de la construcció de mapes a partir de dades GPS. També vull agrair a tots els professors (Mateu Sbert, Imma Boada, Miquel Feixas, Francesc Castro, Josep Soler, ...) i als companys (Mario, Pau, Adrià, Marc, Xaquín) del Laboratori de Gràfics i Imatge (GILab) la seva ajuda, tant en la vida diària com en el treball. Vull donar les gràcies a tot el personal del departament d'Informàtica, Matemàtica Aplicada i Estadística per la seva amabilitat.

Agraeixo a la meva família (pares, germanes, germà, cosins, ties, oncles, ...) el seu amor i suport incondicional i agraeixo als meus amics (Wen, Robert, Xin, Feng, Silvia, Xiaojuan, Chunchun, Chuan, Liangjian, Dandan, ...) la seva ajuda i companyia en el temps lliure. Especialment vull agrair a Wen, Robert i Xin que passessin temps amb mi i m'animessin quan passava per moments difícils.

Finalment, agraeixo el suport a tots els companys de pis (Angel, Maria, Julia, Sergi, Gerard, Montse, Marika, Quim) amb els quals he viscut, als companys de classe (Silvia, Olga, Hanfei, Sergi, Sergio, Verda, ...), i als professors (Paco i Dankmute) del curs d'Espanyol i Alemany.

Acknowledgements

The work in this thesis has been supported by grants from the Spanish Government (Nr. TIN2016-75866-C3-3-R) and the Catalan Government (Nr. 2017-SGR-1101). I acknowledge the support from Secretaria d'Universitats i Recerca del Departament d'Empresa i Coneixement de la Generalitat de Catalunya and the European Social Fund.

The datasets in this thesis are public. In the section of trajectory anomaly detection, I would like to acknowledge the authors Piciarelli *et al.* for putting the collection of synthetic datasets and trajectory generator, Morris and Trivedi for giving the CROSS and LABOMNI, Lazarević *et al.* for presenting the recorded video trajectory dataset and Chen *et al.* for putting the UCR time series public online, respectively. In the section of map construction from GPS data, I would like to thank Ahmed *et al.* for releasing the four urban datasets: Athens small, Athens large, Chicago and Berlin available online. I would also like to thank Duran *et al.* for providing the four hiking datasets: Delta, Aiguamolls, Garraf and Montseny. The implementation of the Slide tool is available online, and I would like to acknowledge the author Paul Mach for sharing the code. I would also like to thank Ahmed *et al.* for sharing the implementation of different map construction algorithms and evaluation measures, and thank Wang *et al.* for making the code of their map construction method public in GitHub.

Finally, I am also grateful to Dropbox. When I made a mistake of deleting everything at the time to finish, its powerful recovery function saved me and avoided a disaster.

List of Publications

Publications that support this thesis are:

- Yuejun Guo, Anton Bardera, Marta Fort and Rodrigo I. Silveira. Global schematic complete map construction from urban and hiking trajectory data. To submit to International Journal of Geographical Information Science, 2020.
- Yuejun Guo and Anton Bardera. *SHNN-CAD$^+$: An Improvement on SHNN-CAD for Adaptive Online Trajectory Anomaly Detection.* Sensors, vol. 19, no. 1, 2019.
- Anton Bardera, Marta Fort and Yuejun Guo. *Route Graph Construction from GPS Trajectory Data.* Accepted by XVIII Spanish Meeting on Computational Geometry, Girona, July 1-3, 2019.

Previous publications related with this thesis that have been achieved during phd study are:

- Yuejun Guo, Qing Xu, Peng Li, Mateu Sbert and Yu Yang. *Trajectory Shape Analysis and Anomaly Detection Utilizing Information Theory Tools.* Entropy, vol. 19, no. 7, page 323, 2017.
- Yuejun Guo, Qing Xu and Mateu Sbert. *IBVis: Interactive Visual Analytics for Information Bottleneck Based Trajectory Clustering.* Entropy, vol. 20, no. 3, page 159, 2018.
- Yuejun Guo, Qing Xu, Xiaoxiao Luo, Hao Wei, Hongjuan Bu and Mateu Sbert. *A Group-Based Signal Filtering Approach for Trajectory Abstraction and Restoration.* Neural Computing and Applications, vol. 29, no. 9, pages 371-387, May 2018.

List of abbreviations

***p*-values** probability values.

CAD conformal anomaly detector.

CaD context- aware distance.

DBSCAN density-based spatial clustering of applications with noise.

DH-kNN NCM directed Hausdorff k-nearest neighbour non-conformity measure.

DHD directed Hausdorff distance.

DHD(ω) directed Hausdorff distance with constraint window.

DTW dynamic time warping.

FN false negatives.

FP false positives.

GPS global positioning system.

HMM hidden Markow model.

iBAT isolation-based anomalous trajectory.

KDE kernel density estimation.

kNN k-nearest neighbors.

LCSS longest common subsequence.

MD merge distance.

NCM non-conformity measure.

ROC receiver operating characteristic.

SHNN-CAD sequential Hausdorff nearest-neighbor conformal anomaly detector.

SHNN-CAD^{+} enhanced version of SHNN-CAD.

SNN-CAD similarity based nearest neighbour conformal anomaly detector.

ST-DBSCAN spatial-temporal DBSCAN.

TC1 trace clustering algorithm.

TN true negatives.

TP true positives.

TRAOD trajectory outlier detection algorithm.

List of Figures

List of Tables

Contents

Abstract

With a large amount of trajectory data generated every day, there is a high demand for developing advanced techniques to discover the underlying information instead of dull and heavy manual work. This thesis focuses on the anomaly detection and map construction from GPS data. Anomaly detection aims to identify trajectories that do not follow common behaviors, and map construction deals with a set of trajectory data to generate a route graph that represents the main movement paths hidden in data.

First, to perform online anomaly detection, we study the well-known Sequential Hausdorff Nearest-Neighbor Conformal Anomaly Detector (SHNN-CAD) approach, and propose an enhanced version called SHNN-CAD$^+$. We compute the anomaly threshold adaptively instead of using a predefinition with no prior knowledge. Also, we propose a modified Hausdorff distance that works more accurately and faster. Besides, a re-do strategy makes the detection more flexible and accurate. Extensive experiments on both real and synthetic trajectory data show that SHNN-CAD$^+$ outperforms SHNN-CAD concerning accuracy and runtime.

Second, two common limitations exist in most map construction algorithms: time-consuming and low coverage of GPS data. We present a new, fast and robust three-step framework this is a smart combination of known methods. First, we build an initial density surface by mapping the trajectory data into a regular grid that covers the geographical area. Second, trajectories are adjusted by Slide to align to recompute a more compact density surface. Besides, we propose two solutions to solve the respective defects of Slide. Third, we construct a map by a thinning algorithm and the Douglas-Peucker simplification method. Experimental results on real datasets demonstrate that our framework outperforms the other algorithms concerning the data coverage, artifacts, and quantitative measures.

Third, considering the storage limitation and computational cost dealing with large-scale data, we propose a split-and-merge strategy. We split the geographical area into small regions, then perform the three-step framework in each region. Finally, we merge the generated maps. Furthermore, we propose to use an overlapping zone between adjacent regions to keep the connectivity and consistency of the map. Besides, we utilize the edge weight to visualize the map and remove the wrong edges induced by noise in data. To compute the weight of an edge, we consider both the length and frequency. The experiments on real GPS data demonstrate that the split-and-merge strategy speeds the process and keeps the map quality. The utilization of edge weight achieves promising results in visualizing the map with information on popular routes and in improving the map quality.

Resum

La gran quantitat de dades de trajectòries generades cada dia fa que hi hagi una gran necessitat de desenvolupar tècniques avançades per descobrir la informació subjacent de les dades evitant un treball manual avorrit i pesat. Aquesta tesi es centra en la detecció d'anomalies i la construcció de mapes a partir de dades GPS. La detecció d'anomalies pretén identificar trajectòries que no segueixen comportaments habituals; i la construcció de mapes rep un conjunt de trajectòries i genera un graf de camins que representa els principals moviments amagats en els dades d'entrada.

En primer lloc, per realitzar la detecció d'anomalies en temps real, estudiem el conegut detector formal d'anomalies del veí més proper de Hausdorff (SHNN-CAD) i proposem una versió millorada anomenada SHNN-CAD$^+$. Determinem el llindar d'anomalia de manera adaptativa en lloc de prefixar-lo sense usar coneixement previ. També proposem usar una distància de Hausdorff modificada ja que els resultats són més acurats i ràpids. A més, l'estratègia de refer fa la detecció més flexible i precisa. Extensos experiments usant dades de trajectòries reals i sintètiques mostren que SHNN-CAD$^+$ supera el SHNN-CAD pel que fa a la precisió i el temps d'execució.

En segon lloc, en la majoria dels algorismes de construcció de mapes existeixen dues limitacions comunes: el temps d'execució i la baixa cobertura de les dades de GPS. Presentem una estratègia amb tres passos nova, ràpida i robusta que combina, de forma intel·ligent, mètodes coneguts per extreure un graf que representi tots els camins seguits per les trajectòries. En primer lloc, construïm una superfície de densitat inicial remostrejant cada trajectòria i mapejant-la a una quadrícula regular que cobreix l'àrea d'estudi. En segon lloc, les trajectòries s'ajusten usant Slide per alinear-les perquè defineixin una superfície de densitat més compacta. A més, proposem dues solucions per resoldre alguns dels problemes de l'Slide. En tercer lloc, es recalcula la superfície de densitat a partir de les noves trajectòries ja ajustades; aquesta nova superfície s'utilitza per obtenir el graf dels camins mitjançant un algorisme d'aprimament i el mètode de simplificació Douglas-Peucker. Els resultats experimentals en conjunts de dades reals demostren que el nostre algorisme deixa els altres obsolets pel que fa a la cobertura de les dades inicials, els artefactes i les mesures quantitatives existents.

En tercer lloc, tenint en compte la limitació d'emmagatzematge i el cost computacional relacionat amb dades de GPS a gran escala, proposem una estratègia de divisió i fusió. Dividim l'àrea geogràfica en petites regions i s'aplica l'estratègia amb tres etapes a cada regió. Finalment, fusionem els mapes generats. A més, proposem usar una zona de solapament entre regions adjacents per mantenir la connectivitat i la consistència dels mapes generats. També usem el pes de les arestes per visualitzar el mapa i eliminar les arestes errònies induïdes pel soroll de les dades. Per calcular el pes d'una aresta, considerem la seva longitud i freqüència. Els experiments obtinguts amb dades reals de GPS demostren que l'estratègia de divisió i fusió accelera el procés i manté la qualitat del mapa. L'ús del pes de les arestes obté resultats prometedors en la visualització del mapa amb informació de la popularitat de les rutes i en la millora de la seva qualitat.

Resumen

La gran cantidad de datos de trayectorias generados cada día hace que exista una gran necesidad de desarrollar técnicas avanzadas para descubrir la información subyacente de los datos evitando un trabajo manual aburrido y pesado. Esta tesis se centra en la detección de anomalías y la construcción de mapas a partir de datos GPS. La detección de anomalías identifica trayectorias que no siguen comportamientos habituales; y la construcción de mapas recibe un conjunto de trayectorias y genera un grafo de caminos que representa los principales movimientos escondidos en los datos de entrada.

En primer lugar, para realizar la detección de anomalías en línea, estudiamos el conocido detector formal de anomalías del vecino más cercano de Hausdorff secuencial (SHNN-CAD) y proponemos una versión mejorada llamada SHNN-CAD$^+$. Determinamos el umbral de anomalía de forma adaptativa en lugar de prefijarlo sin usar conocimiento previo. También proponemos una distancia de Hausdorff modificada ya que los resultados son más exactos y rápidos. Además, la estrategia de rehacer usada hace que la detección sea más flexible y precisa. Extensos experimentos muestran que SHNN-CAD$^+$ supera a SHNN-CAD en cuanto a precisión y tiempo de ejecución.

En segundo lugar, en la mayoría de los algoritmos de construcción de mapas existen dos limitaciones comunes: el tiempo de ejecución y la poca cobertura de los datos GPS. Presentamos una estrategia con tres pasos nueva, rápida y robusta que combina, de forma inteligente, métodos conocidos para extraer un grafo que represente todos los caminos seguidos por las trayectorias. Primero, construimos una superficie de densidad inicial remuestreando cada trayectoria y mapeándola en una cuadrícula regular que cubre el área de estudio. Segundo, las trayectorias se ajustan usando Slide para alinearlas con el objetivo de que definan una superficie de densidad más compacta. Tercero, se recalcula la superficie de densidad en base a las nuevas trayectorias ya ajustadas, esta nueva superficie se utiliza para obtener el grafo de caminos con un algoritmo de adelgazamiento y el método de simplificación Douglas-Peucker. Los resultados experimentales en datos reales demuestran que nuestro algoritmo deja los demás obsoletos en relación a la cobertura de los datos iniciales, los artefactos y las medidas cuantitativas existentes.

En tercer lugar, considerando la limitación de almacenamiento y el coste computacional relacionado con datos a gran escala, proponemos una estrategia de división y fusión. Dividimos el área geográfica en pequeñas regiones y les aplicamos la estrategia de tres pasos. Finalmente, fusionamos los mapas generados. Además, proponemos usar una zona superpuesta entre regiones adyacentes para mantener la conectividad y la consistencia de los mapas generados. También utilizamos el peso de las aristas para visualizar el mapa y eliminar las aristas incorrectas inducidas por el ruido en los datos. Para calcular el peso de una arista, se tienen en cuenta su longitud y el número de trayectorias que la definen. Los experimentos obtenidos con datos reales de GPS demuestran que la estrategia de división y fusión acelera el proceso y mantiene la calidad del mapa. El uso del peso de la arista logra resultados prometedores en la visualización del mapa con información de popularidad de rutas y en la mejora de la calidad del mapa.

CHAPTER 1

Introduction

Contents

1.1 Motivation

Thanks to the advanced location-aware sensors and global positioning system (GPS) devices, a large number of trajectory data from continuously moving objects, like people, vehicles, hurricanes, and aircraft, are generated every day. Generally, a trajectory is stored as a finite sequence of sample points, and each sample point is represented by its attributes, such as location, timestamp, and speed. Discovering knowledge from trajectory data contributes to many application domains [Kong 2018], including video surveillance [Haritaoglu 2000, Majecka 2009], airspace monitoring [Gariel 2011], landfall forecasts [Powell 2001], animals' migratory analysis [Lee 2008], network intrusion detection [Görnitz 2013], road network generation [Karagiorgou 2012] and so on. As a result, many research works related to trajectory analysis have been conducted on different topics, such as trajectory clustering, classification, anomaly detection, trajectory indexing and retrieval, and map construction [Zheng 2015a]. In this thesis, we focus on the anomaly detection and map construction from trajectory data. Note that anomaly detection can be taken as a pre-processing for map construction to eliminate the outliers.

Abnormal trajectory also refers to anomaly, outlier, anomalous trajectory, suspicious trajectory, and outlying trajectory. The existence of an abnormal trajectory may degrade the analysis performance, which calls for the anomaly detection. Besides, the detected outliers can help to identify abnormal events for many applications, such as video surveillance, intelligent transportation, and animal migration [Yu 2018]. Typically, an outlier is defined as "an observation (or a set of observations) which appears to be inconsistent with the remainder of that set of data" [Barnett 1974], namely it is significantly different (concerning some similarity metric) from the trajectory patterns that frequently occur. The anomaly detection of trajectory data can be included in the clustering procedure by clustering-based algorithms or in a single process by non-clustering-based algorithms. In the first case, the abnormal trajectory is identified along with or after performing the clustering algorithms. In the second case, given the trajectory data, the outlier is recognized by a complete detection approach. For both cases, the challenge is to estimate a fine anomaly threshold or the specific parameters related to this threshold to achieve a highly accurate detection [Keogh 2004]. Moreover, the clustering-based category usually needs a large amount of historical data to train and then to obtain the patterns.

A road map provides the basic geographic information especially the network of streets of the study area, which forms the basis of route planning [Davies 2006]. The traditional way of making route maps is based on expensive ground surveys, remotely sensed images and labor-intensive post-processing, all of which still face technological challenges [Li 2016, Zheng 2018, Deng 2018, Huang 2018]. As an alternative, extracting road map from massive GPS data is becoming an increasingly hot and important topic named map construction in the field of trajectory data analysis. In literature, the concept of map construction is also known as trace graph construction [Edelkamp 2003], map generation [Davies 2006, Guo 2007], graph generation [Cao 2009], street network construction [Ahmed 2012], road network generation or construction [Karagiorgou 2012, Karagiorgou 2013] and map inference [Biagioni 2012b, Liu 2012a]. With any name, the goal is to extract a geometric graph from GPS trajectories (or traces) to represent main movement routes that are followed by these data. Acquiring a road map from massive GPS trajectory data helps to understand the mobility of users and provides a foundation to geography navigation and recommendation system. The generated maps can also be used to detect and update the changes in existing road maps, and to customize maps for travelers [Biagioni 2012b].

A range of studies have been reported and discussed [Biagioni 2012a, Ahmed 2015d] which can be divided into three categories: incremental track insertion, intersection linking, and point clustering. Each category has its characteristics and limitations. The incremental track insertion based algorithms use the greedy strategy to iteratively add a trajectory to update the initial empty map, which is computation-light but the process is irreversible and may converge to the local optimum. The intersection linking category aims to find the intersection points each of which connects more than two edges. The common method is to perform a clustering algorithm on all the track points to obtain the cluster centers as the intersections, which is time-consuming due to the need for computing the distance between a large number of pairwise points. Besides, determining the number of clusters (number of intersections) is not straightforward with no prior knowledge of data. The point clustering category either applies clustering on points to produce vertices of the route graph or estimates the density distribution to obtain the route structure. The kernel density estimation (KDE) is mostly utilized on all the points to do the density computation. However, since KDE estimates the density at one specified point by visiting all the points, it is quite heavy work. Furthermore, most algorithms ignore roads followed by low frequency of trajectories for being unimportant [Cao 2009, Wang 2015]. Here, low frequency means to a small number. This kind of road possibly does not appear in the standard map but is also valuable. For example, the tracks from outdoor activities like hiking may include unknown roads that users are interested in. What's more, lacking of up-to-date information can make the standard map improper for performance evaluation. In turn, the inferred maps can be helpful to update the standard maps and to detect errors [Biagioni 2012b]. In this thesis, we aim at developing a fast algorithm to extract a road map that records all the paths appearing in GPS data.

Almost all the map construction approaches require determining different parameters to generate a high-quality map from GPS data. In general, for the data-dependent parameters, extensive experiments have to be carried out. This task gets more difficult when there are several unrelated parameters of an algorithm or the data volume is big. Also, storage memory will be a problem when dealing with a large amount of data. For instance, in density-based algorithms, the density distribution is an important input. However, if the geographical area is very big and the grid of the area is fine, the recording would be unreachable since the storage of computers is limited. Besides, storing the distance from a quite large number of pairwise points is necessary for the approaches utilizing the clustering algorithms. Thus, considering the limitations of computational cost and storage of the computer performance, we propose to perform the map construction more efficiently with a split-and-merge strategy.

The generated maps are usually visualized to show the road structure without more information like the road popularity. Assigning weight to roads for vivid visualization is useful. For this reason, we propose to utilize the edge weight to facilitate the map display. Furthermore, due to the noise in data, the generated map is always not the same as the groundtruth. The edge weight can be helpful to remove wrong connections and improve the map quality.

1.2 Objectives

The goal of this thesis is to analyze trajectory data via developing advanced techniques to detect anomalies and to generate road maps. To fulfill this aim, the following objectives are considered:

1. Detect trajectory outliers online

   Anomaly detection is an important part of trajectory analysis. Clustering-based approaches usually produce the outliers as incidental findings, which pays more attention to learn patterns from trajectory data. Generally, to obtain a comprehensive set of patterns needs a large amount of training data, resulting in high computational cost. Besides, with the increase of testing data, the patterns should be updated, otherwise, the detection is less confident. However, when and how to update is still not clear [Laxhammar 2014b], and there is a time delay. Online anomaly detection requires a fast and accurate manner, thus the non-clustering based algorithm would be a better option. To be suitable for different types of trajectory data, the algorithm should be parameter-light. Furthermore, the anomaly threshold or related parameters need to be data-adaptive, avoiding the influence of user experience.

2. Generate a route map with a high coverage of GPS data rapidly

Extracting a road map from GPS data is widely studied in the literature, while there are two common limitations: time-consuming and ignorance of infrequent trajectories. To solve these issues, we focus on developing a fast and robust algorithm to perform the route map generation rapidly and to output a precise summary graph that presents almost all the paths followed by GPS data.

3. Deal with large-scale data and improve the visualization in map construction

   First, in the case of generating a route map from a large-scale GPS data related to a large geographical area and long-time collection, not so much attention is paid in literature but is valuable in practical applications. Taking into account both the storage limitation and computational cost problems, we aim to deal with this objective through a split-and-merge strategy. Then the key problem will be how to keep the consistency and connectivity at the splitting boundaries. Second, in general, the generated map is presented with single-color lines to show the roads. We propose to visualize the map utilizing the information of how many trajectories cross through the roads to give insights into the popular zones. Besides, this information can be used to remove the wrong edges.

1.3 Thesis Outline

The dissertation is organized into six chapters. Following the introduction, the background is presented. Then we describe techniques related to the three objectives, respectively, along with the experimental results and discussion in each chapter. Finally, the conclusion and future work are presented.

- Chapter 2: **Background**

  In this chapter, some common basic concepts about trajectory analysis are firstly presented to give a preliminary insight on trajectory data, the distance measure, and the clustering technique. Next, an overview of previous work related to anomaly detection and map construction is given. As the performance evaluation is a key work in algorithm design, we also include the validation methods.

- Chapter 3: **Adaptive Online Trajectory Anomaly Detection**

  Based on a recent anomaly detection algorithm called Sequential Hausdorff Nearest-Neighbor Conformal Anomaly Detector (SHNN-CAD), we propose to enhance the performance in this chapter. SHNN-CAD is a parameter-light algorithm which is also able to do the online detection by dealing with the raw trajectory data directly. According to several observed limitations, different improvement strategies are introduced and discussed with extensive experiments.

- Chapter 4: **Map Construction From GPS Trajectory Data by a Three-Step Framework**

  This chapter deals with the problem of map construction from GPS trajectory data. To solve the common issues, data coverage, and computational cost, we present a three-step framework that is a smart combination of well-known algorithms. We use the slide tool to make the trajectory data distribute more densely, and a thinning algorithm to obtain the skeleton.

- Chapter 5: **Improvements on the Three-Step Map Construction Framework**

  Based on the three-step framework, this chapter presents two ways to improve the performance. First, given a large volume of GPS data, the storage limitation and computational cost would be the main problems to construct a route map effectively. To solve this, this chapter shows a split-and-merge strategy that splits the geographical area into small regions, then computes the route maps separately, and finally merges the small maps to obtain a complete map for the whole area. Second, computing the edge weight and utilizing it to visualize the route graph gives insights into popular and rare roads. Besides, the edge weight can help to filter wrong edges caused by noise in data. In this chapter, the edge weight is computed with two factors: edge frequency and edge length.

- Chapter 6: **Conclusions**

  This chapter provides the concluding remarks and the future work of the thesis with the related publications.

CHAPTER 2

Background

Contents

2.1 Introduction

In this chapter, we present an overview of previous work related to this thesis. First, Section 2.2 introduces the basic concepts of trajectory analysis that are commonly used in anomaly detection and map construction. Second, Section 2.3 provides an overview of the anomaly detection algorithms. The last Section 2.4 is devoted to introduce the research work on map construction.

2.2 Basic Concepts

In this section, some basic concepts related to the topics of anomaly detection and map construction are reviewed. As both topics are dealing with trajectory data, Section 2.2.1 briefly introduces the concept of trajectory data. Section 2.2.2 presents some widely used distance measures for quantifying the similarity between points or trajectories. Some anomaly detection methods first cluster trajectories to obtain patterns for further identifying outliers, and map construction algorithms use clustering of points to generate nodes in the route graph. Thus, Section 2.2.3 outlines the typical clustering algorithms for mining patterns in trajectory data.

2.2.1 Trajectory Data

In a recent survey, Kong *et al.* [Kong 2018] classified the trajectory data into two categories, explicit trajectory data and implicit trajectory data, and clearly explained each category and the application of trajectory data. In brief, the explicit trajectory data have clear time and location information stored in the point, while the implicit trajectory data, such as sensor-based data and network-based data, has quite weak spatiotemporal continuity. In practice, with basic data processing operations, the implicit trajectory data can be converted to explicit trajectory data, for example, extracting trajectories from surveillance videos [Majecka 2009]. Thus, the commonly used trajectory data refers to explicit trajectory data.

The widely accepted definition of "trajectory" is that a trajectory is a trace (a track, a path) generated by a moving object in geographical space, usually recorded as a sequence of multi-dimensional points [Zheng 2015b, Mazimpaka 2016]. In literature, trajectory data is also known as tracking data [Karagiorgou 2012, Ahmed 2015c] and trajectories are called traces [Davies 2006, Cao 2009, Biagioni 2012b]. Each point usually includes the location information and some additional information, such as timestamp, time duration, height, speed and so on. The location of GPS data is in the format of longitude and latitude in a geographic coordinate system, and the data extracted from implicit trajectory data are in the format of x and y in a Cartesian coordinate system. Via a simple transformation operation, the values in geographic coordinate system can be mapped to the Cartesian coordinate system for simplification and using the general methods, for instance, the Euclidean distance measure. Figure 2.1 shows an example of recording a trajectory in different formats.

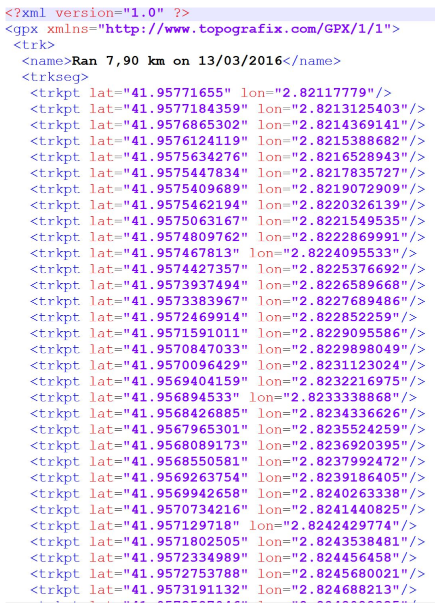
2.2.2 Distance Measure

First of all, we define some notations for this section. A trajectory P is a sequence of points $\{p_1, p_2, \ldots, p_m\}$. Each point is denoted by x, y-coordinates, such as $p_i = \left(p_{i,x}, p_{i,y}\right)$. Let $Q = \{q_1, q_2, \ldots, q_n\}$ be a trajectory with n points. The Euclidean distance between two points, p_i and q_j, respectively from P and Q is

$$dist\left(p_i, q_j\right) = \sqrt{\left(p_{i,x} - q_{j,x}\right)^2 + \left(p_{i,y} - q_{j,y}\right)^2} \tag{2.1}$$

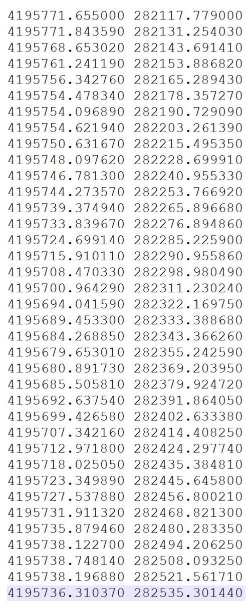
where $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$. In literature, the length of a trajectory usually has two meanings: number of points and sum length of pairwise points. In this section, we use the second meaning. Namely, the length of trajectory P is $l\left(P\right) = \sum_{i=1}^{m} dist\left(p_i, p_{i+1}\right)$.

Quantifying the similarity (or the distance) between points or trajectories is a fundamental task in many trajectory analysis approaches, such as clustering, anomaly detec-



```
<?xml version="1.0" ?>
<gpx xmlns="http://www.topografix.com/GPX/1/1">
 <trk>
  <name>Ran 7,90 km on 13/03/2016</name>
  <trkseg>
   <trkpt lat="41.95771655" lon="2.82117779"/>
   <trkpt lat="41.9577184359" lon="2.8213125403"/>
   <trkpt lat="41.9576865302" lon="2.8214369141"/>
   <trkpt lat="41.9576124119" lon="2.8215388682"/>
   <trkpt lat="41.9575634276" lon="2.8216528943"/>
   <trkpt lat="41.9575447834" lon="2.8217835727"/>
   <trkpt lat="41.9575409689" lon="2.8219072909"/>
   <trkpt lat="41.9575462194" lon="2.8220326139"/>
   <trkpt lat="41.9575063167" lon="2.8221549535"/>
   <trkpt lat="41.9574809762" lon="2.8222869991"/>
   <trkpt lat="41.957467813" lon="2.8224095533"/>
   <trkpt lat="41.9574427357" lon="2.8225376692"/>
   <trkpt lat="41.9573937494" lon="2.8226589668"/>
   <trkpt lat="41.9573383967" lon="2.8227689486"/>
   <trkpt lat="41.9572469914" lon="2.822852259"/>
   <trkpt lat="41.9571591011" lon="2.8229095586"/>
   <trkpt lat="41.9570847033" lon="2.8229898049"/>
   <trkpt lat="41.9570096429" lon="2.8231123024"/>
   <trkpt lat="41.9569404159" lon="2.8232216975"/>
   <trkpt lat="41.956894533" lon="2.8233338868"/>
   <trkpt lat="41.9568426885" lon="2.8234336626"/>
   <trkpt lat="41.9567965301" lon="2.8235524259"/>
   <trkpt lat="41.9568089173" lon="2.8236920395"/>
   <trkpt lat="41.9568550581" lon="2.8237992472"/>
   <trkpt lat="41.9569263754" lon="2.8239186405"/>
   <trkpt lat="41.9569942658" lon="2.8240263338"/>
   <trkpt lat="41.9570734216" lon="2.8241440825"/>
   <trkpt lat="41.957129718" lon="2.8242429774"/>
   <trkpt lat="41.9571802505" lon="2.8243538481"/>
   <trkpt lat="41.9572334989" lon="2.824456458"/>
   <trkpt lat="41.9572753788" lon="2.8245680021"/>
   <trkpt lat="41.9573191132" lon="2.824688213"/>
```

(a) Longitude and latitude



```
4195771.655000 282117.779000
4195771.843590 282131.254030
4195768.653020 282143.691410
4195761.241190 282153.886820
4195756.342760 282165.289430
4195754.478340 282178.357270
4195754.096890 282190.729090
4195754.621940 282203.261390
4195750.631670 282215.495350
4195748.097620 282228.699910
4195746.781300 282240.955330
4195744.273570 282253.766920
4195739.374940 282265.896680
4195733.839670 282276.894860
4195724.699140 282285.225900
4195715.910110 282290.955860
4195708.470330 282298.980490
4195700.964290 282311.230240
4195694.041590 282322.169750
4195689.453300 282333.388680
4195684.268850 282343.366260
4195679.653010 282355.242590
4195680.891730 282369.203950
4195685.505810 282379.924720
4195692.637540 282391.864050
4195699.426580 282402.633380
4195707.342160 282414.408250
4195712.971800 282424.297740
4195718.025050 282435.384810
4195723.349890 282445.645800
4195727.537880 282456.800210
4195731.911320 282468.821300
4195735.879460 282480.283350
4195738.122700 282494.206250
4195738.748140 282508.093250
4195738.196880 282521.561710
4195736.310370 282535.301440
```

(b) x,y-coordinates

Figure 2.1: Example of recording a trajectory where the points are in the format of (a) longitude and latitude, (b) x,y-coordinates.

tion and map construction. Extensive distance measures for this purpose have been proposed and studied [Ding 2008, Yuan 2017a]. The representative distance measures are Euclidean distance, Hausdorff distance, Frèchet distance, dynamic time warping (DTW) and longest common subsequence (LCSS) [Zhang 2006, Ding 2008, Morris 2009a]. Euclidean distance is the simplest and fast measure to quantify the distance between two trajectories [Faloutsos 1994, Hu 2007, Guo 2015, Luo 2015, Guo 2018c], and is widely applied as a prior measure by the other distance measures, such as Hausdorff distance, to obtain the distance. Given two trajectories P and Q. If $m = n$, the Euclidean distance between P and Q is the sum of the distances between corresponding points:

$$D_e(P,Q) = \sum_{i=1}^{m} dist(p_i, q_i) \tag{2.2}$$

The strict condition that the Euclidean distance requires the two trajectories to have the same number of sample points makes it inapplicable for most practical data, and also because of this this distance measure is very sensitive to noise [Ding 2008]. Hausdorff distance overcomes this problem by only taking the largest distance of all the points from a trajectory to their corresponding closest points from another trajectory [Alt 1992, Laxhammar 2011, Laxhammar 2014b], which is clear in the definition:

$$D_h(P,Q) = \max\left(\max_{p_i \in P}\left(\min_{q_j \in Q} dist\left(p_i, q_j\right)\right), \max_{q_j \in Q}\left(\min_{p_i \in P} dist\left(p_i, q_j\right)\right)\right) \tag{2.3}$$

where

$$\overrightarrow{dh}(P,Q) = \max_{p_i \in P} \left(\min_{q_j \in Q} dist\left(p_i, q_j\right) \right) \tag{2.4}$$

is known as the directed Hausdorff distance from P to Q. The shortcoming of Hausdorff distance is that it ignores the order of sample points, as a result it fails to account for the direction information implied by trajectory data of moving objects. In addition, it compares the distance between every pair of points respectively of two trajectories, leading to a high computational cost.

To solve the above weaknesses of Hausdorff distance, Atev *et al.* [Atev 2006] modified it by adding neighborhood windows to search the best matching points:

$$h_\alpha(P,Q) = \overset{\alpha}{\underset{p_i \in P}{\mathrm{ord}}} \left(\min_{q_j \in N(C(p_i))} dist\left(p_i, q_j\right) \right) \tag{2.5}$$

$N\left(C\left(p_i\right)\right)$ represents the points from Q that are regarded as the neighborhood of a point p_i in trajectory P, which enables a limited search space. $\overset{\alpha}{\underset{p_i \in P}{\mathrm{ord}}} f(.)$ denotes the value of $f(.)$ that is bigger than α percent of all the other $f(.)$ values over P.

Similar with Hausdorff distance, the Frèchet distance is shape-based which was designed to measure the similarity between curves [Eiter 1994]. Let L be a coupling between P and Q which is a sequence of distinct point pairs from two trajectories. L is denoted by $\left(p_{a_1}, q_{b_1}\right), \left(p_{a_2}, q_{b_2}\right), \ldots, \left(p_{a_s}, q_{b_s}\right)$ where $a_1 = 1$, $b_1 = 1$, $a_s = m$ and $b_s = n$. A big difference with Hausdorff distance is that the Frèchet distance does not backtrack from one point to the others which means that it takes into account the temporal order of points, so a strict rule is that for any i, $a_{i+1} = a_i$ or $a_{i+1} = a_i + 1$ and $b_{i+1} = b_i$ or $b_{i+1} = b_i + 1$. The length of L is the largest distance between pairwise points and is computed by

$$||L|| = \max_{i=1,2,\ldots,s} dist\left(p_{a_i}, q_{b_i}\right) \tag{2.6}$$

The discrete Frèchet distance between two trajectories P and Q is defined as:

$$\delta_{dF}(P,Q) = \min(||L||). \tag{2.7}$$

The Frèchet distance can deal with trajectories that have different number of points. Both Hausdorff and discrete Frèchet distance measures are sensitive to noise considering that only one distance between pairwise points is taken.

DTW calculates the distance between two trajectories by using non-linear temporal alignment [Berndt 1994, Izakian 2016]. The distance between P and Q is defined as [Berndt 1994]:

$$D_{dtw}(P,Q) = \begin{cases} 0 & m = n = 0 \\ \infty & m = 0 \; or \; n = 0 \\ dist\left(p_1, q_1\right) + \min\{D_{dtw}\left(Rest\left(P\right), Rest\left(Q\right)\right), & \\ \qquad D_{dtw}\left(Rest\left(P\right), Q\right), D_{dtw}\left(P, Rest\left(Q\right)\right)\} & otherwise \end{cases} \tag{2.8}$$

where $Rest\,(P)$ and $Rest\,(Q)$ indicate the rest pats of P and Q by removing p_1 and q_1, respectively. DTW provides better alignment but it may produce pathological result. For instance, a sample point from one trajectory can be mapped to a large subset of points from another trajectory. Enforcing a temporal constraint on the wrapping window size can improve the accuracy, but it may also avoid the correct wrapping, too.

LCSS finds the longest common subsequence from two trajectories [Gariel 2011, Choong 2017], thus does not require the trajectory data to have equal number of points.

$$D_{lcss}\,(P,Q) = \begin{cases} 0 & m = 0 \; or \; n = 0 \\ D_{lcss}\,(Rest(P), Rest(Q)) + 1 & \begin{array}{l}|p_{1,x} - q_{1,x}| \le \varepsilon \; and \\ \quad |p_{1,y} - q_{1,y}| \le \varepsilon\end{array} \\ \max\left\{D_{lcss}\,(Rest\,(P)\,, Q)\,, D_{lcss}\,(P, Rest\,(Q))\right\} & otherwise \end{cases} \tag{2.9}$$

where ε is the matching threshold.

Hu *et al.* [Hu 2007] proposed to measure the distance between trajectories by averaging the Euclidean distance between every two sample points from corresponding trajectories (called HU distance in [Morris 2009a]):

$$D_{hu} = \frac{1}{m}\sum_{i=1}^{m} dist\,(p_i, q_i) \tag{2.10}$$

However, this distance measure has a limitation that the trajectory data should have equal number of points. Afterwards, the similarity between these two trajectories is given by

$$Sim_{hu} = \exp\left[\frac{-D_{hu}}{2\sigma^2}\right] \tag{2.11}$$

where σ controls the decrease of similarity as the distance increases.

Piciarelli *et al.* [Piciarelli 2006] put forward a distance measure (called PF distance in [Morris 2009a]) between a trajectory and a cluster to deal with online clustering. This distance takes into account the time shift which influences many typical distance measures. Given a trajectory P and a cluster $C = \{c_1, c_2, \ldots, c_s\}$, the distance is defined as

$$D_{pf}\,(P,C) = \frac{1}{m}\sum_{i=1}^{m} \min_{j=1}^{s}\left(\frac{\text{dist}\left(p_i, c_j\right)}{\sqrt{\sigma_j^2}}\right) \tag{2.12}$$

where σ is the local variance of the cluster at a certain time, j is limited to several values within a window related to i to handle the time shift. In addition, the cluster is represented as a list of vector which combines the information of all the trajectories contained to avoid memorizing a large number of data and to improve the online performance.

Lee *et al.* [Lee 2008] adapted the angle, parallel and perpendicular distances that are widely used in the field of pattern recognition to fit for trajectory data. The proposed distance measure is a weighted average of these three distances for flexible usage in different applications. Benefiting from this distance measure, Hu *et al.* [Hu 2018] computed the distance between sub-trajectories to extract different spatial features.

Ismail and Vigneron [Ismail 2015] presented the Merge Distance (MD) to measure the similarity between trajectories, especially between GPS data. Given two trajectories P and Q, MD aims to find the *shortest supertrajectory*, $s(P,Q)$, which has the shortest length to connect all the sample points of P and Q [Li 2018]. The distance is given by

$$MD(P,Q) = \frac{2\ell(s(P,Q))}{\ell(P) + \ell(Q)} - 1 \tag{2.13}$$

This expression ensures that $MD(P,Q) = 0$ when P is equal to Q, and $MD(P,Q)$ is large when P and Q is very different, and vice versa.

San Román *et al.* [Román 2018] proposed a context-aware distance (CaD) to measure the distance between trajectories. The distance is designed to imply a weighted average of the differences in angle, the Euclidean distance and the number of sample points in each trajectory. In an extreme condition that two trajectories are parallel, if the Euclidean distance has the maximum importance, then the obtained CaD equals to the Euclidean distance; if the angle has the maximum importance, then CaD is zero as there is no difference concerning direction. However, in another extreme condition where two trajectories are vertical, the Euclidean distance is ignored by CaD regardless of weights. CaD does not consider this case.

There are also some specific distance measures for computing differences between two probability distributions, such as Bhattacharyya distance [Bhattacharyya 1946, Mcfadyen 2016, Guo 2017], Kullback-Leibler divergence, and Jensen-Shannon divergence [Lin 1991, Guo 2017]. Calderara *et al.* [Calderara 2011] derived a closed-form of Bhattacharyya distance for single von Mises probability density functions. This kind of measure requires to estimate the probability density function of each trajectory firstly. The limitation is that if a trajectory has only a few points, the estimation is not accurate.

In summary, although many approaches have been proposed, selecting a good distance measure is still a challenge concerning various types of data, attribute under consideration and computational cost. The simplest Euclidean distance is fast but does work for trajectories with different number of points. However, even if the trajectory data have an equal number of points, Euclidean distance may not work well if many attributes are considered. Hausdorff distance, Frèchet distance, DTW and LCSS solve the problem of the unequal number of points by performing matching on points. Hausdorff distance is the fastest one but ignores the order of points. Based on these measures, some variants have been proposed to fit specific scenes. The measures to computing the distance between probability distributions are less sensitive to noise but are not accurate when the trajectories have a few points to estimate precise distributions. For most

2.2.3 Clustering Algorithm

The clustering algorithms aim at partitioning data into non-overlapping clusters where each cluster includes the points or trajectories (hereafter in this section, we use “object” to indicate point or trajectory for clustering) sharing similar attributes and indicates a pattern of behavior. Typically, the clustering algorithms can be divided into five categories: partition, hierarchy, density, grid and model-based [Han 2011, Yuan 2017a].

The partition-based clustering algorithms focus on separating a dataset into k clusters where k is a preset parameter. The dataset is initially divided into k groups, via several iterations in which each object is updated to change the group under a preset similarity criterion, the iterative procedure terminates with some conditions (e.g. optimizing a chosen objective function or reaching a given iteration number). The typical partition-based algorithms are k-means [MacQueen 1967, Gariel 2011] and k-medoids [Kaufman 1987, Prati 2008]. k-medoids is less sensitive to outliers than k-means because k-medoids uses the most central object in the cluster to represent the cluster rather than using the mean of all the objects in the cluster by k-means. Despite the wide popularity of the partition-based algorithms, some drawbacks call for improved variants. First, the initial centers influence the convergence speed. If the initial centers are distributed in corresponding different clusters, the clustering procedure terminates faster than that the initial centers are from the same cluster. Second, without any prior knowledge of the patterns in data, the number k is unknown and heavily depends on the user experience or extensive experiments to be defined. Third, selecting an appropriate similarity criterion is still a big challenge, especially, when dealing with trajectory data that have an unequal number of points or multi-dimensional features.

The hierarchy-based clustering methods are either agglomerative or divisive [Fu 2005, Morris 2009a, Jiang 2009, Wang 2011]. To start with, the agglomerative hierarchical algorithm takes each object as a unique cluster, then within each iteration, two clusters that have the minimum distance are merged. The process stops when the minimum distance at one iteration is bigger than a predefined distance threshold, or a given number of clusters are achieved, or there is only one cluster left. In contrast, the divisive hierarchical algorithm initially considers that all the objects belong to a single cluster, and in each iteration, a cluster is split into two. Similarly, many hierarchy-based clustering methods face the problem of selecting an appropriate distance measure between objects. Furthermore, considering the previous merged or split clusters are not changed in the following iterations, this type of algorithms may encounter a local optimum. The clustering performance heavily relies on the distance calculation. To measure the distance between two clusters where one cluster has more than one object, there are three strategies: single linkage, complete linkage, and average linkage [Yim 2015]. The single linkage selects the minimum value of all the distances between pairwise objects, while the complete linkage takes the maximum value. Both linkage measures do not take the structure of the cluster into account, which may degrade the performance especially if outliers exist. The average linkage uses the average of all the pairwise distances, which solves the problem of ignoring the structure of cluster although it requires more calculation than the other two linkage measures. The information bottleneck has been recently applied to perform clustering on trajectory data [Guo 2014, Guo 2015, Guo 2016, Guo 2017]. The agglomerative information bottleneck uses the loss of mutual information instead of a distance measure to quantify the similarity between trajectories or clusters. By comparing the difference of mutual information between two clusters before and after merging them together, the clusters are determined to be merged or not. In this way, the common issue of clustering algorithms of choosing a proper distance measure for different kinds of data is solved. Besides, the change of the loss of mutual information from merging clusters gives the clue to stop the process, which is more practical than pre-defining a distance threshold without any prior knowledge of the data.

The density-based clustering algorithms identify the contiguous areas of high density in a dataset as different clusters where they are separated by empty or sparse areas, and the objects in the sparse area are considered as outliers [Sander 2010]. A well-known algorithm is the density-based spatial clustering of applications with noise (DBSCAN) algorithm. DBSCAN groups the objects that are close to each other into a cluster where the number of objects is also above a predefined value. As a result, DBSCAN requires two parameters, epsilon and minimum points, to control the similarity between trajectories and the density of a cluster. There are two issues in setting the two parameters. First, when dealing with different types of data, the distance between trajectories changes a lot, therefore it is difficult to set a proper epsilon. Second, even within the same dataset, the densities of clusters can be very different, so setting a global number of minimum points does not fit the task. Hereafter, some variants have been proposed for different improvements, which are compared in a recent survey [Singh 2017].

The grid-based clustering algorithms generally map the spatial area into a grid with rectangular cells, then perform clustering of data based on each cell. One advantage is that the clustering is fast since the clustering only needs to visit the objects within the same cell rather than compare the similarity among all the objects. The model-based clustering methods assume that each cluster specifies a model, thus the goal is to find modes in data to obtain different clusters. Undoubtedly, finding a proper model for various types of data is still an open challenge. Considering that these two categories are more widely applied for point clustering instead of trajectory clustering, we only present a few review work, but more information can be found in the surveys [Han 2011, Yuan 2017a].

2.3 Anomaly Detection

In the last few years, a branch of research has made an effort to find efficient ways to detect outliers in trajectory data [Gupta 2014, Parmar 2017, Meng 2018]. Since in reality, the available trajectories are usually raw data without any prior knowledge, this section focuses on the unsupervised approaches of anomaly detection which can be grouped into two categories: clustering-based and non-clustering based [Meng 2018].

2.3.1 Clustering-Based Approaches

The clustering-based approaches obtain patterns by clustering all the trajectories. Some algorithms, like DBSCAN and its variants, detect the outliers at the same time. The others identify the outliers by comparing the difference with the learned patterns. DBSCAN is of particular interest for both clustering and outlier detection, considering that it is capable of discovering arbitrary shapes of clusters along with reporting outliers [Ester 1996, Gariel 2011, Ying 2009]. However, as mentioned in Section 2.2.3, DBSCAN faces to the problem with parameter estimation. Birant and Kut [Birant 2007] proposed the Spatial-temporal DBSCAN (ST-DBSCAN) to improves DBSCAN by additionally dealing with the time attribute, but it increases the computational cost to calculate the similarity on both spatial and temporal dimensions. It is well known that density-based algorithms face with the problem of varied densities in data, Zhu *et al.* [Zhu 2018] proposed to solve this issue by developing a multi-dimensional scaling (DScale) method to readjust the computed distance.

Kumar *et al.* [Kumar 2017] proposed iVAT+ and clusiVAT+ for trajectory analysis along with detecting outliers. These approaches group the trajectories into different clusters by partitioning the Minimum Spanning Tree (MST). To build MST, DTW distance between trajectories is regarded as the weight of the corresponding edge. The clusters which have very few trajectories are taken as irregular patterns, as a result, the included trajectories are outliers. This requires the user expectation for determining how "few" should be.

Jiang and An [Jiang 2008] presented a clustering-based outlier detection (CBOD) approach to pick out the outliers. Firstly, one-pass clustering algorithm [Jiang 2004] is applied to produce clusters from all data. Then the outlier factor is calculated for all the clusters, and later the sorted clusters contribute to get the anomaly threshold. Accordingly, the objects in the outlier cluster are treated as outliers.

Given the clusters (training set), the testing trajectory is marked anomalous if the difference between it and the closest cluster center (centroid or medoid) is over an anomaly threshold. The typical k-nearest neighbors (kNN) is widely applied due to its simplicity and low computational cost [Ramaswamy 2000]. A testing trajectory is determined as belonging to the same cluster as the most frequent trajectory in the k neighbors, or as abnormal if the distance with its k-th neighbor overs a threshold. Here, two important aspects that may influence the performance: the considered number of neighbors k and the distance computation. Fu *el al.* [Fu 2005] introduced an online anomaly detection method that detects the suspicious parts of a trajectory instead of the whole. Both the spatial and speed abnormal behaviors are taken into account through a two-step scheme. Given learned clusters, the testing trajectory is firstly labeled by the most similar cluster. In the first step, a sample point is regarded as an outlier if the part of trajectory up to this point closes to a cluster which is different from the one where the whole trajectory is assigned to, and the distance is calculated following Bayesian decision theory. In the second step, the speed value at each point is modeled using all the points in a cluster by Gaussian distribution. If the speed exceeds an anomaly threshold (sum of the mean and standard deviation of the model), then the corresponding point is determined as abnormal.

It is typical to use a distance measure to quantify the difference between a testing trajectory and learned patterns. The representative distance measures that have been developed and applied in different applications are Euclidean distance, Hausdorff distance, DTW, LCSS, etc. [Ding 2008]. As the threshold is indirect to determine for varied practical situations, some approaches based on the probabilistic models have been proposed, and the distance is usually taken as the trajectory likelihood. In [Annoni 2012], KDE is used to detect the incoming sample point of the aircraft trajectory in progress.

The sampling point is determined as abnormal or belonging to a certain cluster depending on the probability is small or not. Guo et al. [Guo 2017] proposed to apply the Shannon entropy to adaptively identify if the testing trajectory is normal or not. The normalized distances between the testing trajectory and the cluster centers obtained by the Information Bottleneck method build the probability distribution to compute the Shannon entropy, which measures the information used to detect the abnormality. Similarly, Das and Mishra [Das 2018] obtained outliers via the Shannon entropy after performing mean shift clustering algorithm to train patterns.

In [Prati 2008], only the angle attribute of trajectory data is considered to analyze the shape pattern. Each trajectory is represented by a statistical model using a mixture of von Mises (MoVM) distribution, and the similarity between distributions is measured by Bhattacharyya distance. After producing clusters based on iterative k-medoids, the testing trajectory can be determined as normal or not according to the distance with medoids is large or not. Similarly, Jiang *et al.* [Jiang 2009] introduced a 2-depth search strategy to the dynamic hierarchical clustering algorithm to obtain relatively reliable clusters where the trajectories are characterized by a hidden Markow model (HMM). In the presence of learned clusters, the abnormal trajectories can be determined if the probability of being generated by learned clusters is lower than a data-adaptive threshold. However, properly setting parameters that fit different datasets is difficult. Similarly, Qiao *et al.* [Qiao 2002] proposed to utilize the HMM to detect outliers. The HMM algorithm has the advantage of requiring a very small part of normal data to train a model, which leads to faster detection. However, as interpreted by authors, the number of states is sensitive but difficult to determine.

Piciarelli *et al.* [Piciarelli 2008c] proposed to fulfill the task of event detection in video surveillance via trajectory analysis. The trajectories are clustered using support vector machines (SVM) which is a powerful algorithm for both clustering and classification problems [Cortes 1995]. In terms of the abnormal trajectories, the authors proposed an automatic approach based on geometric considerations in the SVM feature space.

2.3.2 Non-Clustering-Based Approaches

For the clustering-based approaches, it is important to obtain reliable training set or pattern models, which has a high requirement for the accuracy of the algorithms. Some approaches have attempted to detect outliers directly without the clustering procedure as in some applications the users are only interested in abnormal behavior, for instance, event analysis in video sequences [Piciarelli 2011].

In 2005, Keogh *et al.* [Keogh 2005] introduced the definition of time series discord and proposed the heuristically ordered time series using symbolic aggregate approximation (SAX) algorithm to find the subsequence (defined as discord) in a time series that is most different to all the rest subsequences. The authors proposed to search the discord by comparing the distance of each possible subsequence to the nearest non-self match using the brute force algorithm. Although the brute force algorithm is intuitive and simple, the time complexity is very high, which drives them to improve the process in a heuristic way. This definition was then improved and applied in trajectory data by Yankov *et al.* [Yankov 2008] by treating each trajectory as a candidate subsequence.

In the specific application of picking anomalous driving patterns from taxi trajectories, Zhang *et al.* [Zhang 2011] proposed an Isolation-Based Anomalous Trajectory (iBAT) detection approach. Firstly, the city map is split into grid-cells with equal size. It assumes that the trajectories taking the same route are the same ignoring the speed feature, thus each trajectory can be augmented with pseudo-points between discontinuous cells. Secondly, the isolation Forest (iForest) method is introduced and adapted to trajectory data. iForest finds outliers by building the iTrees which separate data to different nodes according to the cell that trajectories pass through, and the trajectories that have short path in iTree are possible to be outliers. Furthermore, iBAT equips with lazy learning to deal with the case that anomaly trajectory detour in the cells that are contained by normal trajectories.

Based on the theory of conformal prediction, Laxhammar *et al.* successively proposed the Similarity based Nearest Neighbour Conformal Anomaly Detector (SNN-CAD) [Laxhammar 2011] and the Sequential Hausdorff Nearest-Neighbor Conformal Anomaly Detector (SHNN-CAD) [Laxhammar 2014b] for online learning and automatic anomaly detection. The work in [Laxhammar 2014b] improves [Laxhammar 2011] with a more comprehensive discussion of previous works and explanations of the algorithm. These two algorithms take advantage of the direct Hausdorff distance to calculate the similarity between each trajectory with its k nearest neighbors. Here, the direct Hausdorff distance can deal with trajectories with different numbers of sampling points. However, the problems of Hausdorff distance are the high computational cost as it visits every pairwise sample points in two trajectories, and that it cannot distinguish the direction while computing. Despite this, these two algorithms obtain quite promising results.

Zhu *et al.* [Zhu 2015] obtained the outliers through a so-called time-dependent popular routes based trajectory outlier detection (TPRO) algorithm. TPRO first separates trajectories with the same source and destination to the same group, and to avoid each group that has too few trajectories, the road network is split into grid-cells with equal size to assume the vertices of source or destination in the same cell have no difference. Then the top-k most popular routes of each group are picked by calculating the route popularity. At last, the testing trajectory is classified as abnormal if the time-dependent edit distance based trajectory route distance with popular routes is too big.

Considering that taking each trajectory as a whole to detect outliers may cause miss-detection due to different homogenization in the local part of trajectory data, some approaches identify outliers through the local change of trajectory data, which requires to first partition the trajectories into line segments. The theoretical basis of partition-based approaches is that if most segments of a trajectory are abnormal, then this trajectory is identified as an outlier, but the big issue is their expensive computational cost because of the segmentation process and the distance calculation between segments. Via a partition and detection framework, Lee *et al.* [Lee 2008] presented an efficient trajectory outlier detection algorithm (TRAOD) which has been a widely cited and compared approach in the field of sub-trajectory-based anomaly detection. In concrete, each trajectory is partitioned to a set of un-overlapping line segments based on the minimum description length (MDL) principle. Then the outlying line segments of a trajectory are picked based on the distance from neighboring trajectories, that is, if there are few trajectories in the dataset that are close to the considered segment, this line segment is identified as outlying. Finally, a trajectory is defined as outlier once a big portion of line segments are recognized as outliers. Due to the line distance measure applied, TRAOD can detect both the positional and angular outliers. The novelty is that this algorithm can detect the outlying line segments other than the whole trajectory. However, the determination of required parameters is not straightforward and relies on experience or repeated experiments, which brings difficulty to work in different applications.

Bao *et al.* [Bao 2017] adopted TRAOD to outlier detection work and proposed a trajectory outlier detection based on local outlier fraction algorithm (TODLOF). After the partition procedure, a local outlier fraction of each line segment is calculated to represent the average relative density difference between segments. Observing that in the partition part of TRAOD, only the spatial information is taken into consideration, which may miss useful knowledge from the temporal side, Hu *et al.* [Hu 2018] proposed an isolation-based trajectory outlier detection algorithm (IBTOD) to handle this problem. Firstly, after the same spatial partitioning, a finer partition of the sub-trajectory set is obtained based on the temporal information. Second, the same distance measure is applied to extract spatial features, namely, perpendicular, parallel and angular distances between sub-trajectories. Besides, the minimum, average and maximum velocities of sub-trajectories are extracted as temporal features. Finally, an isolation mechanism is implemented to identify outlying sub-trajectories.

Banerjee *et al.* [Banerjee 2016] designed Maximal ANomalous sub-TRAjectories (MANTRA) to solve the problem of mining maximal temporally anomalous sub-trajectory in the field of road network management. The type of trajectory data studied is specific and is called network-constrained trajectory which is a connected path in the road network, namely MANTRA is closely combined with practical applications. Thus, it is not easy to be accessed by the other anomaly detection approaches.

Yuan *et al.* [Yuan 2017b] tackled specific abnormal events for reminding drivers of danger. Both the location and direction of the moving object are taken into account and contribute to the sparse reconstruction framework and the motion descriptor by a Bayesian model, respectively. Instead of dealing with raw trajectory data, this work is based on the video data where the object motion (trajectory) is represented by the pixel change between frames.

Many existing partition-based outlier detection methods face the challenge of detecting outliers with continuous outlying line segments. To address this issue, Yu *et al.* [Yu 2018] developed a trajectory outlier detection algorithm based on common slices subsequence (TODCSS). Each trajectory is encoded with direction value at each sample point, which is used to find inflection points to further divide the trajectory into several slices. Then, the common slices (with identical direction-code) of two trajectories are used to measure the distance between trajectories. Finally, the slice outlier and trajectory outlier are respectively detected if they have a few number of neighbors less than a designated threshold which is generally not straightforward to define for different kinds of trajectory data.

2.3.3 Evaluation Measures

Given the groundtruth, there are many approaches to evaluate the performance of anomaly detection. The groundtruth means that each trajectory has a certain label of being an outlier or not. Let *TP*, *FP*, *TN* and *FN* be the true positives (number of objects being correctly detected as outliers), false positives (number of objects being wrongly identified as outliers), true negatives (number of objects being correctly detected as normal) and false negatives (number of objects being wrongly detected as normal), respectively. The calculations of some popular performance measures are described below [Nandeshwar 2011].

Precision specifies how many outliers are found in the objects identified as anomalies by the algorithm. Recall (named as detection rate in [Zhang 2011], sensitivity in [Laxhammar 2011]) measures the ability of the anomaly detection algorithm to find all the outliers in the dataset. $F1$-score is the harmonic mean of precision and recall and obtains an overall evaluation [Piciarelli 2008c, Guo 2017, Laxhammar 2014b]:

$$\text{precision} = \frac{TP}{TP + FP} \tag{2.14}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{2.15}$$

$$\text{F1} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.16}$$

Accuracy (called as partition accuracy (PA) of anomaly vs non-anomaly in [Kumar 2017], normal/abnormal accuracy in [Prati 2008]) computes the proportion of objects that are correctly identified objects (normal or abnormal) [Laxhammar 2011]:

$$\text{accuracy} = \frac{TP + FP}{TP + FP + TN + FN} \tag{2.17}$$

Two types of error rate are also widely used [Jiang 2009]: the false alarm rate measures the rate of objects that are normal in groundtruth but are wrongly identified as outliers by the anomaly detection algorithm [Kumar 2017, Zhang 2011, Laxhammar 2011], and the false rejection rate computes the rate of objects that are outliers in groundtruth but are rejected by the algorithm:

$$\text{false alarm rate} = \frac{FP}{FP + TN} \tag{2.18}$$

$$\text{false rejection rate} = \frac{FN}{TP + FN} \tag{2.19}$$

In [Zhang 2011, Yuan 2017b], the authors used AUC, the Area Under ROC (Receiver Operating Characteristic) Curve [Bradley 1997], as an evaluation criteria. Considering that a high detection rate (recall) with a low false alarm rate indicates a good anomaly detection performance, the ROC curve is obtained by mapping the detection rate to the y-axis and the false alarm rate to the x-axis. The AUC value is defined as the area under the ROC curve. As interpreted in [Zhang 2011, Yuan 2017b], a high AUC value indicates good performance.

2.4 Map Construction

In an earlier survey by Biagioni *et al.* [Biagioni 2012a], different map construction algorithms are divided into three categories: the k-means algorithm, trace merging and KDE-based. The k-means-based approaches perform k-means clustering on cluster points and use the small set of central points of clusters as vertices of the route graph. Then the vertices are linked together to obtain edges to complete the graph. The trace merging approaches produce the edges incrementally by adding new trajectories to update an existing map. The KDE-based methods firstly apply KDE on trajectory data to estimate a density distribution, then use some image processing techniques to make a route graph. Later, Ahmed *et al.* [Ahmed 2015d] made a new and more comprehensive survey of existing map construction algorithms. In their book, they proposed a new classification with the more recent map construction algorithms: incremental track insertion, intersection linking, and point clustering. By comparison, the point clustering category contains both the k-means-based and the KDE-based approaches, the incremental track insertion category includes the trace merging methods, and intersection linking is a new category. In this section, we follow the latter classification to analyze the characteristics and limitations of each category, and we also review some approaches that are not included in any category. Also, quantifying the quality of the generated map besides visual comparison is very important for evaluating the performance of map construction algorithms, thus, we also summarize some map validation approaches.

2.4.1 Incremental Track Insertion

The main idea of the algorithms in this category is to choose a trajectory as the initial map, then to gradually integrate a new trajectory to update the existing map.

Bruntrup [Bruntrup 2005] proposed to insert the new trajectories via three main modules to produce a directed graph where the nodes include the travel time information and each edge has the same length. In each iteration, several nodes from the existing map are selected if they are in the searching area of the new track and have a similar direction. If no nodes are selected, then the track is directly added to the map. In the second module, the depth-first search algorithm is used to obtain directional segments which include ordered nodes. In the final module, these segments with edges between nodes are merged to the map. Cao and Krumm [Cao 2009] also proposed an approach to generate a directed graph, but they paid more attention to firstly remove noise from trajectories. For each point, two types of attraction forces are simulated from its corresponding trajectory and the others in the dataset to pull the similar points to be closer. For the directed graph construction, the opposite directions are also differentiated.

The key idea of the approach proposed in [Niehöfer 2009] is to decompose a map into reasonable segments. In each iteration, the perpendicular distance between the segments from a new trajectory and constructed map from the last iteration is calculated to find the matching parts to merge and unmatching parts to create new nodes and edges. Tang *et al.* [Tang 2017] also used the trajectory segments to match with the existing map, and the map is refined iteratively with the segments by constrained Delaunay triangulation and a modified skeleton of the triangular network.

Ahmed and Wenk [Ahmed 2012] presented a two phases framework. For each iteration, given the constructed map from the last iteration (or an empty map for the first iteration), the first phase focuses on computing a reconstructed graph. The new trajectory is compared with the constructed map based on the curve-graph partial matching [Buchin 2009] to separate the trajectory into matched and unmatched portions. The unmatched portions are added to create or split edges and introduce new vertices. In the second phase, the minimum-link algorithm is applied to the matched portions with their matching edges to obtain minimum-link representative edges, which contributes to reducing the graph complexity. Zheng and Zhu proposed a very similar approach which also uses the partial curve matching method based on the Fréchet distance [Zheng 2018] to match a new track with the existing map. Then, in the second step, the map is updated with creating or splitting edges.

Unlike most related algorithms, at the beginning, Zhang *et al.* [Zhang 2010] utilized the reference road map from OpenStreetMap instead of starting from an empty map. The map is sampled at certain distances to obtain edges as the centerlines for further computing. The perpendicular lines of these centerlines are built to find the trajectories as candidates that have intersections. Finally, the candidate trajectories are used to refine the centerlines.

Although merging segments of a new trajectory to the existing map is the most popular way for many algorithms, processing each point of the trajectory one by one can be a better choice for data with very low sampling frequency where the distance between adjacent points is quite large. Li *et al.* [Li 2012] classified the point to be matched or unmatched by the spatial and semantic relationships with the possible matching edges, then the point is used to update the matched edge or to create a new edge. Hence, the computational cost could be very high if the volume of points is big.

In summary, the algorithms in this category have the advantage that the computation between the constructed map and a new trajectory in each iteration is light and the dataset can grow with time. However, the process is irreversible where the previous insertions are immutable for future trajectories, which may cause the local optimum issue and generate wrong roads. Besides, when the data size is very big, updating the existing map will be computational heavy.

2.4.2 Intersection Linking

The algorithms in this category require to find the intersections firstly and then to link them together. Here, an intersection means a node that connects more than two edges in a route graph, and a link between two adjacent intersections is regarded as an edge.

Fathi and Krumm [Fathi 2010] took the intersection as one of the most important components for map construction and proposed to find the intersections in three steps with also producing the edges. In the first step, a local shape descriptor is introduced to make the intersection detector for finding several points from trajectories as intersections. The next step deals with connecting these intersections according to the trajectories pass through and further pruning the number of intersections. In the final step, the iterative closest point algorithm is applied to refine the intersections associated with close trajectories to locate at more accurate and reasonable positions.

Karagiorgou and Pfoser [Karagiorgou 2012] also proposed a three-step framework (called TRACEBUNDLE algorithm in [Karagiorgou 2013]) for automatic map construction specifically for vehicle trajectory data. The intersections are found in the first step through performing two times hierarchical clustering respectively on turns and the corresponding clusters. Turns are the points where the speed reduces and the direction changes largely. In the second step, links between intersections are connected via using a sweep-line algorithm on all the trajectories. In the final step, the number of links is reduced by using the sweep-line algorithm again, which includes the position change of intersections. Based on this algorithm, the authors proposed to create the maps for different types of movement (distinguished by speed) and then the maps are combined into a single network. In a similar approach with TRACEBUNDLE proposed by Wu *et al.* [Wu 2013], the intersections are found by performing the x-means clustering on turns and converging points. Wang *et al.* [Wang 2017] used the same concept of turn to detect intersections by analyzing the density of turns and clustering with the mean-shift algorithm.

Padu and Pasi [Mariescu-Istodor 2018] developed a two-step method called CellNet to automatically generate the route map. The data is mapped into a grid and in each cell, the information of cell's location, indexes of passing trajectories are recorded. Firstly, the intersections are found through two processes. The mean-shift algorithm performs on the cells to detect splits and a split is a point at which the routes move to more than two principal directions. The concept of split is similar to turn. Then the splits that have a larger number of trajectories pass through than their neighbors are determined as intersections. Second, the intersections are connected to produce a route map. CellNet also includes a filtering process to remove possible wrong edges. All the trajectories are checked to make the connections between intersections, which is a time-consuming work that is demonstrated by their experiments on two real GPS datasets.

Additionally, more studies have attempted to detect intersections. Based on that the trajectories at road intersections have bigger turning angles (direction change) than at non-intersections, Deng *et al.* [Deng 2018] proposed to differentiate the intersection and non-intersection points as the hot spot and cold spot areas, respectively, via a local G^* statistic [Ord 1995]. Then the intersection points are clustered to obtain the centers (final intersections) by the adaptive spatial clustering algorithm based on Delaunay triangulation (ASCDT) [Deng 2011]. In order to generate intersections with detailed structures, Tang *et al.* [Tang 2019] further improved the method by Deng *et al.* [Deng 2018]. Once the clusters are computed via ASCDT, the circle boundaries of the clusters are estimated. Finally, the entrances and exits on the boundaries are detected to produce the intersections with detailed geometric structures. Xie *et al.* [Xie 2017] also made the same assumption that the intersections have high density. The longest common subsequence algorithm is applied on trajectories to segment them and to take the starting end ending points from the common sub-tracks as connecting points. The local maximums on the density distribution obtained estimated by kernel density estimation are regarded as intersections. Li *et al.* [Li 2017] developed the dominant orientation to extract road intersections.

As we can see, finding the intersections from numerous points in a trajectory dataset always needs the distance measurement between points, which is a major work. Besides, it is time-consuming to connect intersections using the information of all the trajectories. In general, algorithms in this category suffer from a heavy computation. The clustering technique is widely applied on track points to produce cluster centers as the rough intersections. As mentioned in Section 2.2.3, selecting a proper distance measure and determining the number of clusters are still open issues.

2.4.3 Point Clustering

Indeed, the point-clustering-based map construction algorithms can also be classified into two types: clustering-based and KDE-based [Ahmed 2015d]. Considering that KDE is not the only method to estimate the density of data, we prefer to use density-based instead of KDE-based. The concept of "density-based" is also used in [Tang 2017]. The clustering-based approaches start with a fixed number of cluster seeds (centers) selected from the points of trajectory data [Biagioni 2012a, Worrall 2007]. Then the seeds are updated via the clustering technique for further linking to produce edges (road centerlines) of the route graph. The density-based ones interpret the set points as a density image of the road networks while some density estimator is used to obtain the density distribution for further obtaining the road structure. The clustering-based category is related to the intersection linking (Section 2.4.2) that both find several numbers of nodes and then link them together to make the map. A basic difference is that the clustering-based approaches search the nodes that connect one or more edges, while the intersection-linking-based algorithms find the specific intersection nodes that connect more than two edges.

Utilizing the k-means clustering to locate the cluster seeds with randomly choosing the initial seeds from trace points is the most popular way in literature [Edelkamp 2003, Schroedl 2004, Stanojevic 2018]. Besides, Agamennoni *et al.* [Agamennoni 2011] obtained the locations of cluster centers via the dominant set clustering. As the motivation is to make a small digital map for energy saving, the links are further compressed by non-linear least squares fitting to generate final map with connected lines and arcs. Li *et al.* [Li 2016] proposed a novel spatial-linear clustering algorithm to cluster the track points which pays more attention to the moving direction of points in the same cluster. In each cluster, the edges are produced by incorporating the trajectory information. An obvious issue is that the edges of the generated route graph are not always connected, leading to the low map connectivity.

Liu *et al.* [Liu 2012b] proposed a trace clustering algorithm (named TC1) which mainly uses the line segments of trajectories as the unit of clustering. Direction and speed are taken into account for detecting and then pruning useless proportions of trajectories, as a result, the trajectory data turn to be a set of line segments. Through the agglomerative hierarchy clustering, the line segments are grouped to different clusters for further computing the centerline of the road.

With an existing base map which can be a commercial map such as OpenStreetMap or can be obtained from a spatial clustering algorithm [Agamennoni 2011], some approaches focus on producing an enhanced and refined map with high accuracy. Guo *et al.* [Guo 2007] modeled the point of each trajectory with Gaussian distribution. All the trajectories are matched to the road segments of the existing map to obtain the feature points by least squares approximation methods. Then, the spline curve fitting on the points is applied to produce curves used for the extraction of central lines to update the map.

The density-based algorithms focus on estimating the density distribution of track points by splitting the geographical area occupied by the trajectories into a grid of cells, then extract route graph from the distribution. A common way is converting the density distribution to a binary or grayscale image through thresholding. From the image, the edges and nodes are computed for the route graph by morphological operations. Ahmed *et al.* [Ahmed 2015a] proposed to set the threshold using persistent homology combined with statistical analysis. Davies *et al.* [Davies 2006] estimated the density distribution by counting the number of trajectories passing through each grid cell and computed the contour of the resulting bitmap. Then, the Voronoi diagram is used to determine the centerline which is cleaned-up to obtain a directed graph representing the street map. This method has the problem of ignoring paths with low frequency.

Shi *et al.* [Shi 2009] started cleaning data to construct the bitmap and then obtained the skeleton. Next, they elaborated on the road network using dilatation to fill gaps among road pixels with a mask, and morphological opening to smooth the road contours and eliminate noise. Finally, the graph is extracted by combustion using the information of the eight neighbors of a cell. Similar methods are presented in [Chen 2008, Chen 2010].

Biagioni and Eriksson [Biagioni 2012b] inferred a map taking into account GPS data with noise and disparity. KDE is utilized to estimate the density image and a gray-scale thinning algorithm with various thresholds is applied to compute several versions of a skeleton map. The last one is represented as an undirected graph whose edges are finally replaced with directed edges. It is a time-consuming algorithm that, in practice, records occasionally transited paths but loses the infrequent ones.

Wang *et al.* [Wang 2015] also used KDE to build the density distribution, then the discrete Morse theory is applied to extract the route map. Based on this work, Dey *et al.* [Dey 2017] made improvements to enhance the performance.

It is clear that the common drawbacks of the algorithms in this category are that infrequent paths are usually ignored, and the extensive clustering and KDE make the computation time-consuming.

2.4.4 Other Map Construction Approaches

There are still some approaches that cannot be classified into any known categories. Different from the point-clustering-based map construction algorithms, the approach proposed by Buchin *et al.* [Buchin 2017] performs the clustering on sub-trajectories rather than points, and the advantage can deal with data of varying sampling rates. In the first step, the sub-trajectories are clustered into bundles, and the most relevant bundles are selected. In the second step, the route map is constructed from the bundles by incrementally growing an initial map with the representative sub-trajectories. The paths with only one or two sub-trajectories are removed, which decreases the data coverage.

Unlike most density-based algorithms that perform density estimation and morphological operation to obtain the skeleton, Huang *et al.* [Huang 2018] proposed to generate a road map through principal graph structure learning on points and tree linking strategy. The best-fitting graph structure of points can be regarded as a skeleton.

2.4.5 Map Validation

In general, there are two types of methods to evaluate the map quality. One is to do the visual inspection to check the difference between the generated map and the groundtruth in a standard road map, such as OpenStreetMap [Schroedl 2004, Zhang 2010, Biagioni 2012a, Ahmed 2014, Ahmed 2015b]. The other one is to quantitatively measure the difference. For simplicity, the generated map and groundtruth are denoted by G and G_{tru}, respectively, in this section.

Visual inspection is the most common and intuitive method in literature [Shi 2009, Zhang 2010, Fathi 2010, Agamennoni 2011, Li 2017, Zheng 2018]. Cao and Krumm [Cao 2009] visually checked that if G covers most of the information from data compared with the road network in Microsoft Bing Maps. Similarly, in [Chen 2008, Niehöfer 2009], the map from Google Earth is adopted to do the visual comparison.

There are different approaches proposed to give a quantitative result. In [Liu 2012a], Liu *et al.* made a comparison of algorithms respectively from [Liu 2012b] and [Davies 2006], showing that the TC1 algorithm (see [Liu 2012b] for more details) produces the best overall performance. The precision (called coverage in [Liu 2012b]), recall and $F1$-score are applied to do the quantitative evaluation based on the common road length concerning G and G_{tru} from the OpenStreetMap. These three metrics are also adopted in [Biagioni 2012b, Li 2016, Li 2017]. In Liu's previous work [Liu 2012b], the false positive rate and accuracy are also used for performance evaluation. In the work by Chen *et al.* [Chen 2010], the coverage is used as a performance measure. Instead of comparing the distance between the edges of maps, Fathi and Krumm [Fathi 2010] measured the distance between locations of the nodes and also computed the accuracy of finding the right nodes from G_{tru}.

In [Karagiorgou 2012], Karagiorgou and Pfoser proposed the shortest path-based measure. Considering that G always includes fewer edges than G_{tru} since the low coverage of trajectory data, this measure starts with randomly selecting a certain number of pairs of source-destination nodes from G, then finds their corresponding nodes on G_{tru}. Using the Dijkstra's algorithm, the shortest path between the source-destination nodes can be easily obtained. Finally, the similarity of the shortest paths from the G and G_{tru} are measured by both the discrete Frèchet distance and the average vertical distance. This measure ignores the missing routes in G. For instance, as shown in Figure 2.2, an algorithm produces a map (A) which is a sub-map of the map (B) resulted from another algorithm from a set of trajectories. Assuming that the nodes and edges on map A are the same with the routes in G_{tru} but map B includes some nodes and edges that have distance with their corresponding locations in G_{tru}. Map B has better coverage of data, however, under this measure, it is evaluated to be worse.

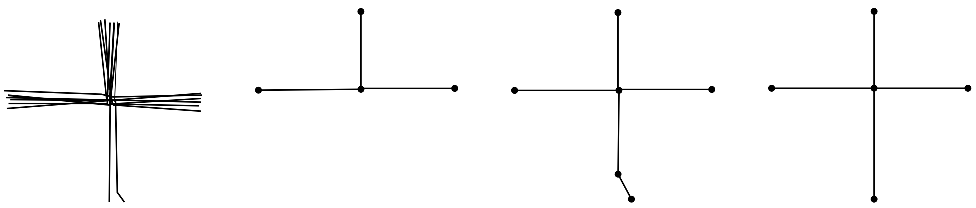

(a) Trajectory data (b) Map A (c) Map B (d) Groundtruth

Figure 2.2: Plots of four urban GPS trajectory datasets. (a) Trajectory data. (b) Map A. (c) Map B. (d) Groundtruth.

In [Karagiorgou 2012], Karagiorgou and Pfoser also applied the directed Hausdorff distance to measure the similarity between G and G_{tru}. Each map is regarded as a set of paths and a path is an edge with two vertices. The similarity is obtained by computing the directed Hausdorff distance between these two sets. However, this measure is not good enough. Imagine that G_{tru} only has one path which is represented as two paths in G. These two maps are the same regardless of the number of vertices and edges, but this distance measure does not output zero.

In the survey work [Biagioni 2012a], to compare the performance of different map construction algorithms, Biagioni and Eriksson proposed a method taking into account both the geometric and topological similarity. This method is called holes and marbles in [Stanojevic 2018] and graph-sampling based distance measure in [Ahmed 2015c], respectively. The main idea is to randomly select a start location in G_{tru}, then the area within a radius from the start location is considered. This area is divided into a grid with holes (or cells). Via mapping G_{tru} and G, spurious marbles, matched marbles and empty holes are defined under different rules. The matched marbles are the holes appearing in both maps. The spurious marbles are the holes existing in G, not G_{tru}. The empty holes are only included in G_{tru} not G. Then the quantitative indices are computed as below:

$$\text{spurious} = \frac{\text{spurious\_marbles}}{\text{spurious\_marbles} + \text{matched\_marbles}} \tag{2.20}$$

$$\text{missing} = \frac{\text{empty\_holes}}{\text{empty\_holes} + \text{matched\_holes}} \tag{2.21}$$

$$\mathrm{F1} = 2 \cdot \frac{(1-\text{spurious})(1-\text{missing})}{(1-\text{spurious})+(1-\text{missing})} \tag{2.22}$$

The unmatched marbles indicate the parts which do not appear in G_{tru}. An obvious problem is that, if the trajectory data pass through many unknown routes from G_{tru}, then even if G is very complete, the measure would give low score since there will be many unmatched marbles and the spurious index would be quite high.

Ahmed *et al.* [Ahmed 2015b] proposed a path-based distance measure to make the comparison. G and G_{tru} are represented as sets of paths, and the Frèchet distance is applied to compute the similarity between paths. Then the distance between these two maps is obtained by the directed Hausdorff distance which is the maximum of all the minimum Frèchet distances between a path in G and paths in G_{tru} as indicated in Equation 2.4. The smaller the distance is, the better the performance is.

In summary, the visual inspection is straightforward to check the correspondence between generated maps and trajectory data or groundtruth, but does not give any numeral result to compare different algorithms. A common issue of the quantitative methods is that the trajectory data may include roads that do not appear in G_{tru} due to the information delay of new roads or user development of unknown paths, then the result will not reflect the real performance. We utilize the visual inspection and path-based distance which has a public implementation [Ahmed 2015d] to do the evaluation.

CHAPTER 3

Adaptive Online Trajectory Anomaly Detection

Contents

3.1 Introduction

The main objective of anomaly detection is to pick out anomalous data which are significantly different from the patterns that frequently occur in data. A variety of approaches have been proposed for the task of trajectory anomaly detection [Meng 2018], however, most of them have limitations of computational cost or parameter selection [Keogh 2004], leading to the difficulty in reproducing experimental results. Usually, the trajectory of a moving object is collected and stored as a sequence of sample points which record the location and timestamp information, but the complementary information of data is lacking. Here, the complementary information refers to information about the number of patterns included, the trajectories labeled with certain patterns, the abnormal pattern, and so on, which can help the analysis of data. To automatically find this information, unsupervised approaches are commonly applied. In this case, it is not straightforward or easy to finely tune the parameters for these approaches to cope with different kinds of data. Although some approaches make effort to estimate the parameters simply by experience and a lot of experiments, or complicatedly, by introducing more assisted parameters and rules, the parameter setting is not trivial with respect to different distributions. Especially, for online handling of massive datasets, the low computational complexity is of great importance. Thus, it is better to avoid complex and time-consuming pre-processing on data.

SHNN-CAD is an algorithm based on the theory of conformal prediction [Vovk 2005]. It has the following main advantages. First, it deals with raw data, which prevents the problems of information loss from dimension reduction, and over-fitting from modeling. Second, it makes use of the direct Hausdorff distance to calculate the similarity between trajectories. As it is well known, selecting a proper distance measure is still a challenge when the trajectories in a set have an unequal number of points due to sampling rate, sampling duration, and moving speed. To address this issue, the direct Hausdorff distance is a good choice as it is parameter-free, and it can handle the case of the unequal number of points. Third, unlike most related approaches requiring several not intuitive parameters, SHNN-CAD is parameter-light and does not assume data structure, which enables the easy reproduction of experiments. The authors provided a method to adjust the anomaly threshold based on the desired alarm rate or the expected frequency of anomalies. Fourth, SHNN-CAD can perform online anomaly detection, which enables the increase of data size. However, it also has some limitations. In this chapter, we propose SHNN-CAD$^+$ to enhance the performance of SHNN-CAD from different aspects. Compared with the previous approach, SHNN-CAD$^+$ has the following improvements:

1. The problems of applying Hausdorff distance directly to trajectory data are the high computational cost as it visits every pairwise sample points in two trajectories, and that it cannot distinguish the direction while computing because the trajectories are considered as a set of points. In [Laxhammar 2014b], the Voronoi diagram is used to accelerate the calculation of the Hausdorff distance, but it is complicated to implement. On the other hand, the direction attribute can be added when computing distance, but this extension will increase the computational cost. To solve this, a modified distance measure based on the directed Hausdorff distance is proposed to calculate the difference between trajectories. Besides, the modified measure has the advantage of a faster computation, which meets the requirement of performing online learning in a fast manner.

2. According to the description in [Laxhammar 2014b], when the data size is quite small, the new coming trajectory can be regarded to be abnormal, however, when more trajectories are considered, this trajectory may have enough similar neighbors to be identified as normal. Our solution is introducing a re-do step into the detection procedure to identify anomalous data more accurately.

3. The anomaly threshold is a critical parameter since it controls the sensitivity to true anomalies and the error rate. In [Laxhammar 2014b], Laxhammar *et al.* manually select this threshold. Instead of pre-defining the anomaly threshold,

an adaptive and data-based method is proposed to make the algorithm more parameter-light, which is more easily applicable for practical use.

Besides, compared with the work in [Laxhammar 2014b], we expand the experiments in two aspects:

1. To evaluate the performance of anomaly detection, F1-score is used in [Laxhammar 2014b] to compare SHNN-CAD with different approaches. We propose to apply more performance measures, such as precision, recall, accuracy, and false alarm rate, to analyze the behavior of anomaly detection algorithms comprehensively.

2. One important advantage of the Hausdorff distance is that it can deal with trajectory data with different numbers of sample points. However, in the experiments of evaluating SHNN-CAD [Laxhammar 2014b], all the testing data have the same number of sample points. Also, the experiments are enriched by introducing more datasets with an unequal number of points.

The rest of this chapter is organized as follows. Following this introduction, Section 3.2 reviews the basic theory of conformal prediction and its application to trajectory anomaly detection which refers to the conformal anomaly detector (CAD), the preliminary SNN-CAD and also the SHNN-CAD. Section 3.3 interprets where SHNN-CAD can be improved for practical use from our observations, and then explains the SHNN-CAD$^+$ that improves some limitations of SHNN-CAD. Section 3.4 presents extensive experiments on both real and synthetic datasets and discusses the performance of the proposed improvement strategies. Finally, concluding remarks and future work are given in Section 3.5.

3.2 Previous work

The related work about trajectory anomaly detection has been reviewed in Section 2.3. In this chapter, we briefly describe the development of SHNN-CAD, which includes the basic conformal prediction theory and its application in trajectory anomaly detection, CAD, SNN-CAD and also SHNN-CAD. A more comprehensive interpretation is given in [Laxhammar 2014a].

3.2.1 Conformal Anomaly Detection

Firstly, Laxhammar *et al.* proposed CAD which is a novel application of the theory of conformal prediction [Vovk 2005] to trajectory anomaly detection. In brief, given a specified nonconformity measure (NCM) and a training set $\mathbb{T} = \{x_1, x_2, \ldots, x_l\}$ with objects in different certain labels, the conformal prediction framework provides the p-values (probability values) of possible labels for a new test x_{l+1} with respect to the training set with a significance level (error probability) ε.

The NCM is the core of conformal prediction and calculates how different a new test is from the training set. Given the training set $\mathbb{T} = \{x_1, x_2, \ldots, x_l\}$ and a new test

x_{l+1}, for each available label, NCM returns the nonconformity score α_{l+1} of x_{l+1}. Considering that the conformal prediction assumes that the objects are independent and identically distributed, the p-value can be determined as the ratio of the number of observed objects that have greater or equal nonconformity scores to x_{l+1} to the total number of objects.

$$p_{l+1} = \frac{|\{\alpha_i | \alpha_i \geq \alpha_{l+1}, 1 \leq i \leq l+1\}|}{l+1}, \tag{3.1}$$

where $|\{.\}|$ gives the size of the resulting set. If p_{l+1} is bigger than the significance level ε, then the corresponding label is included in the prediction set.

Based on the theory of conformal prediction to anomaly detection, CAD works to determine a new test is abnormal or not by taking the significance level ε as an anomaly threshold [Laxhammar 2010]. If $p_{l+1} < \varepsilon$, then x_{l+1} is identified as conformal anomaly, otherwise, x_{l+1} is grouped to the normal set. kNN equipped with Euclidean distance is applied as the NCM and the nonconformity score of x_i is computed by

$$\alpha_i = \sum_{x_j \in NNeighbor} e\left(x_i, x_j\right), \tag{3.2}$$

where $e\left(x_i, x_j\right)$ measures the Euclidean distance between x_i and its jth nearest neighbor x_j. As discussed in Section 2.2.2, this NCM has a strict requirement is that all the trajectories should be represented with the same number of points due to the characteristic of Euclidean distance. As a result, it may face the problem that trajectories are represented with unequal number of points.

3.2.2 SNN-CAD Based Anomaly Detection

To solve the problem of CAD aforementioned, Laxhammar *et al.* [Laxhammar 2011] proposed a novel NCM by using a more general similarity measure (for example, the directed Hausdorff distance) instead of the Euclidean distance, and this novel NCM is called as Directed Hausdorff k-Nearest Neighbour Non-Conformity Measure (DH-kNN NCM). To be able to do sequential anomaly detection where the testing trajectory is incomplete, the directed Hausdorff distance between an incomplete trajectory and the trajectories in the training set is computed recursively. Let $(s_1, s_2, \ldots, s_m)$ and x be an incomplete trajectory with m points and a complete trajectory from the training set, respectively, the distance is defined by

$$\overrightarrow{dh}((s_1, s_2, \ldots, s_m), x) = \max\left(\overrightarrow{dh}((s_{m-1}, s_m), x), \overrightarrow{dh}((s_1, s_2, \ldots, s_{m-1}), x)\right) \tag{3.3}$$

Unlike most sequential anomaly detection algorithms that determine each point in a testing trajectory is abnormal or not, the directed Hausdorff distance enables the identification of an incomplete trajectory.

3.2.3 SHNN-CAD Based Anomaly Detection

As indicated in [Laxhammar 2011], Laxhammar *et al.* expanded SNN-CAD by presenting more discussion of anomaly detection, formalizing CAD, DH-kNN NCM and more experiments, SNN-CAD is reintroduced as SHNN-CAD with detailed description.

Usually, unsupervised algorithms do not use any training set, and the outliers are defined to be observations that are far more infrequent than normal patterns [Chandola 2009]. In this chapter, we also adopt the "training set" concept as used in [Laxhammar 2014b], assuming that the training set has only normal instances. In practical applications, due to the advantage of SHNN-CAD that only a small volume of data is needed as training set, these data can be chosen by users to make the algorithm work more effectively.

3.3 SHNN-CAD$^+$: An Improvement of SHNN-CAD

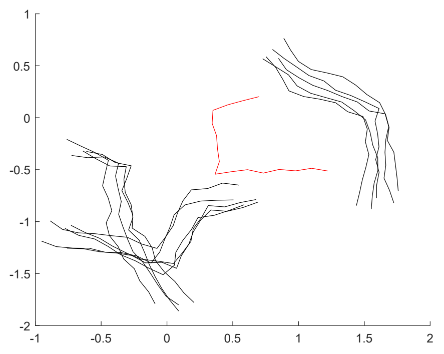
First, we discuss several factors that influence the performance of SHNN-CAD [Laxhammar 2014b]. Afterward, we introduce the improvement strategies to improve the performance.

3.3.1 Discussion of SHNN-CAD

SHNN-CAD is not capable enough to adaptively detect outliers efficiently. The reason is threefold which is interpreted below.

First, using directed Hausdorff distance (DHD) to quantify the distance between trajectories cannot distinguish the difference in direction. DHD has the advantage of dealing with trajectory data with different numbers of sample points, showing the ability of computing distance for a single sample point or a line segment, which is important for the sequential anomaly detection of SHNN-CAD. Nonetheless, DHD was originally designed for point sets with no order between points as shown from the definition. Given two trajectories, A and B, with m and n points, respectively, the DHD from A to B is defined by (see Equation 2.4). Computing the DHD matrix of data is the most time-consuming part of SHNN-CAD, while the time complexity of DHD, $\mathcal{O}(mn)$ (without loss of granularity, assuming that $m \geq n$ henceforth), is high in the traditional computation way that visits every two sample points from corresponding trajectories, which is almost impractical for large size datasets in real world. Alternatively, in [Laxhammar 2014b], the authors adopted the algorithm proposed by Alt [Alt 2009] which benefits from Voronoi diagram to reduce the time complexity to $\mathcal{O}((m+n)\log(m+n))$, but this algorithm requires to pre-process each trajectory by representing the included sample point based on its former neighbor. Moreover, although the trajectory is recorded as a collection of sample points, the order between points must be considered since it refers to the moving direction. For example, a car running in the lane with inverse direction should be detected as abnormal. DHD is not sensitive to the direction attribute. As suggested in some literatures, the direction attribute at each sample point can be extracted and added to the feature matrix to obtain the distance, but the extra feature will increase the computational cost of the distance measure. On the other hand, the direction attribute is generally computed as the intersection angle between the line segment and the horizontal axis, which may introduce noise to the feature matrix.

Second, SHNN-CAD is designed for online learning and anomaly detection which is highly desirable in practical applications, such as video surveillance. When a new observation is added into the database, SHNN-CAD decides it to be abnormal or not based on its p-value. As it can be seen in Equation (3.1), the p-value of an observation counts the number of trajectories from the training set that have greater or equal nonconformity score to it. The greater the p-value, the closer the observation to its k nearest neighbors, thus, it has a higher probability of being normal. According to the mechanism of SHNN-CAD, once the trajectory comes to the dataset, it is identified as normal or not, and then the training set is updated for the next testing trajectory. As shown in Figure 3.1(a), the red trajectory has no similar items, thus it is detected as abnormal assuming that the p-value is bigger than the predefined anomaly threshold. However, unlike the case of a fixed dataset, the neighbors of an observation in the online analysis are dynamic. In Figure 3.1(b), the red trajectory has several similar items (in blue color), which means that its p-value may change to be greater than the anomaly threshold, and should be considered as normal. In conclusion, ignoring the influence of time evolution may bring errors in online anomaly detection.


(a) Dataset with 15 trajectories

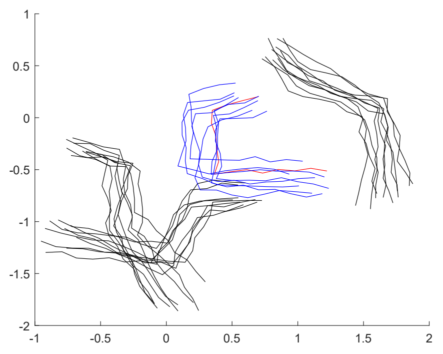

(b) Dataset with 28 trajectories

Figure 3.1: Plots of the trajectory change in dataset.

Third, SHNN-CAD has two settings of the anomaly threshold. The first one, $\frac{1}{l+1}$, is to deal with the problem of zero sensitivity when the dataset size l is small. The second one is a predefined ε ($\varepsilon > \frac{1}{l+1}$). In the first case, zero sensitivity means that, as long as $l < \varepsilon$, a normal observation that has the smallest p-value, $\frac{1}{l+1}$, is identified as abnormal if there is only one threshold ε. Essentially, the first threshold defines the new coming observation as abnormal in default if it has the smallest p-value, $\frac{1}{l+1}$. However, this strategy causes another problem of zero precision of anomaly detection. For example, an actual abnormal observation that has the smallest p-value, is classified as normal. In the second case, in theory, when the anomaly threshold is equal to the prior unknown probability of abnormal class λ, SHNN-CAD achieves perfect performance. However, for unsupervised CAD, no background information can help to tune ε.

3.3.2 SHNN-CAD$^+$

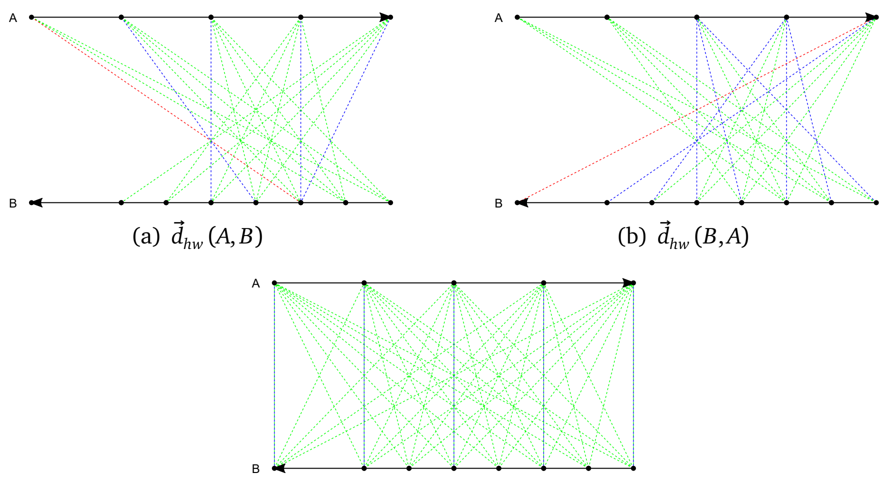
According to the previous discussion, three improvement strategies to enhance the performance of anomaly detection are proposed and explained clearly and thoroughly. Finally, the pseudocode of the SHNN-CAD$^+$ method to detect if a new coming trajectory is abnormal is given and described.

First, as aforementioned, directly utilizing DHD for trajectory distance measure happens the problems of direction neglect and high computational complexity. The first problem is due to that DHD considers the two trajectories as point sets, which ignores the order between sample points, leading to the decline of accuracy. The second one is because that all the pairwise distances between the points of both sets have to be computed. To address the above issues, we propose to modify DHD by introducing a constraint window. The definition of DHD with constraint window, DHD(ω), from trajectory A to trajectory B is

$$\vec{d}_{hw}(A,B) = \begin{cases} \max\left\{\max_{i=1}^{n+\omega}\left(\min_{|j-i|\le w} d_p\left(a_i, b_j\right)\right)\right\}, & m-n \le w \\ \max\left\{\max_{i=1}^{n+\omega}\left(\min_{|j-i|\le w} d_p\left(a_i, b_j\right)\right), \max_{i=n+\omega+1}^{m} d_p\left(a_i, b_n\right)\right\}, & m-n > w \end{cases} \tag{3.4}$$

where ω denotes the size of constraint window. Considering that trajectories with different number of points have a large difference in speed, ω is set as $\lceil \frac{m}{n} \rceil$ to limit that similar trajectories are homogeneous in speed. Each sample point in trajectory A visits at most $2\omega + 1$ sample points in trajectory B, resulting in a linear time complexity $\mathcal{O}((m+n)\,\omega)$, where $\omega \ll m, n$. On the other hand, the search space is limited to the sample points that follow temporal order, which not only enables the consideration of direction, but also improves the accuracy for measuring the distance as an extended visit may introduce wrong matching between two trajectories. Note that the direction of a sample point is an important attribute to give higher performance of detecting abnormal events for trajectory data in practical applications, such as vehicle reverse driving.

Figure 3.2 gives an example to illustrate the distance computation. Trajectories A and B have 5 and 8 sample points, respectively, and they have opposite directions which are indicated by the arrows at the endpoints. The dashed lines visualized in red, green, and blue colors represent the distances between sample points that are required to compute the distance two trajectories. The shortest distance between one sample point and its corresponding points from another trajectory is shown in blue or red. The red line indicates the maximum from all the shortest distances, namely the distance between A and B. Suppose that the window size is 2, the distances from A to B and from B to A by DHD(ω) are computed as illustrated in Figure 3.2(a) and Figure 3.2(b), respectively. Figure 3.2(c) shows the distance computation by DHD. The distances from A to B and from B to A are the same (all the blue lines are equal). In contrast, DHD(ω) saves the computational cost. Additionally, due to that, the order of sample points is taken into account, DHD(ω) captures the difference between trajectories more accurately than DHD concerning different features in trajectory data.



(a) $\vec{d}_{hw}(A,B)$ (b) $\vec{d}_{hw}(B,A)$

(c) $\vec{d}(A,B)$, $\vec{d}(B,A)$

Figure 3.2: Plots of the distance between two trajectories A and B by DHD(ω) and DHD.

Second, considering that in online learning, the outlier may turn into normal once it has enough similar trajectories, we propose to apply a re-do step. To save time, not all outliers are rechecked with every new coming. According to the theory of conformal anomaly detector, the anomaly threshold ε indicates how much probability of outliers occur in the dataset. Thus, if the size of outliers arrives larger than expected, several previously detected outliers may be detected as normal. For example, when the size of training data is l, if the new coming x_{l+1} is identified as an outlier but the number of outliers is greater than expected $(l+1)\cdot\varepsilon$, the previous outliers will be rechecked if they can be moved to the normal class or not. In particular, this strategy helps to solve the problem of zero precision by SHNN-CAD when the data size is small. Regarding the use of a predefined anomaly threshold, it is more reasonable to treat the new coming as abnormal when its p-value is too small, which means it has few similar items from the training set. Via the re-do strategy, the outliers can be picked out gradually with new comings. We don't recheck the normal ones, because their similar trajectories may increase or remain the same. In fact, for large volume data, the normal trajectories are usually pruned to discard redundant information or are trained into mixtures of models to avoid the high computational cost, which will be considered in the future work.

Third, automated identification of outliers requires a data-adaptive anomaly threshold instead of explicitly adjusting for different kind of trajectory datasets. Unlike SHNN-CAD, we define the threshold for the new coming when the size of the training set, l, is the minimum probability of a normal trajectory from training set $\mathcal{N}_l$.

$$\varepsilon_l = \min_{i\in\mathcal{N}_l} p_i \tag{3.5}$$

where p_i is the p-value of ith trajectory. This setting is intuitive and straightforward since the p-values of the other trajectories in the training set are greater or equal to the defined anomaly threshold, which enables them to be normal. The determination of ε depends on the condition of the considering training set, which makes the approach more applicable for different datasets.

Algorithm 1 shows how SHNN-CAD^{+} works for a new coming. Compared with SHNN-CAD, the previously detected outliers are also inputted to perform the re-do strategy. Given the input, two zero distance arrays are built for the new coming x_{l+m+1} (Lines 1 and 2). Lines 3–10 compute the nonconformity scores of all trajectories by summing the distances with the k nearest neighbors. The element $D_{i,j}$ denotes the distance from ith trajectory to its jth neighbor, which is obtained through the modified Hausdorff distance. Then the p-values are calculated in Lines 11–12 according to Equation (3.1). Differing from SHNN-CAD, the anomaly threshold ε is dynamically updated depending on the training set (Line 13). From Line 14 to 25, the new coming is identified as an outlier or not with the defined ε, and the outlier and training sets are updated correspondingly. If the size of the outlier set is over the expected value, the re-do strategy is performed on each previous outlier to check if it can be turned to the normal set (Lines 17–22).

3.4 Results and Discussion

In this section, we present the conducted experiments. The MATLAB implementation is available in [Guo 2018a]. Firstly, a comparison of the proposed DHD(ω) to the typical DHD is given. Secondly, the performance of applying DHD(ω) to the anomaly detection measure is analyzed. Finally, the improvement over SHNN-CAD is evaluated. All the experimental results are obtained by MATLAB 2018a software running on a Windows machine with Intel Core i7 2.40 GHZ CPU and 8 GB RAM.

3.4.1 Comparison of Distance Measure

To evaluate the performance of measuring distance of DHD(ω), we adopt the 10 cross-validation test using 1-Nearest Neighbor (1NN) classifier which has been demonstrated to work well to achieve this goal [Tan 2007, Ding 2008]. 1NN is parameter-free and the classification error ratio of 1NN only depends on the performance of the distance measure. Initially, the dataset is randomly divided into 10 groups. Then each group is successively taken as a testing set, and the rest work as the training set for the 1NN classifier. Finally, each testing trajectory is classified into the same cluster with its nearest neighbor in the training set. The 10 cross-validation test runs 100 times to obtain average error ratio. The classification error ratio of each run is calculated as follows:

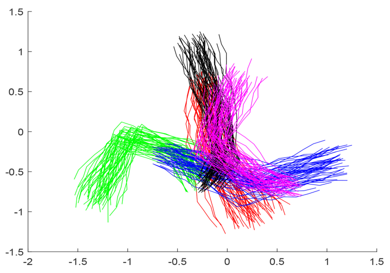
$$\text{classification error ratio} = \frac{1}{10}\sum_{i=1}^{10}\frac{\text{number of trajectories wrongly classified}}{\text{number of trajectories in the } i\text{th testing set}} \tag{3.6}$$

1000 synthetic, 1 simulated, and 1 real trajectory datasets are utilized in this experiment. The Synthetic Trajectories I dataset (numbered “I” to distinguish from the

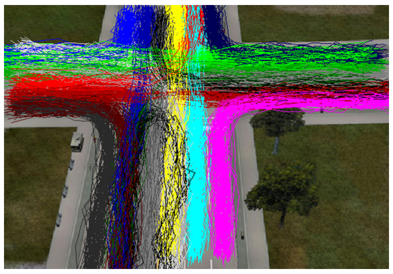
Algorithm 1: Adaptive online trajectory anomaly detection with SHNN-CAD$^+$

```
Input: Training set $\mathbb{T} = (x_1, x_2, \ldots, x_l)$, detected outliers
       $\mathbb{O} = (x_{l+1}, x_{l+2}, \ldots, x_{l+m})$, new coming $x_{l+m+1}$, number of nearest
       neighbors $k$, distance matrix $D$
Output: Updated training set $\mathbb{T}'$; updated outlier set $\mathbb{O}'$, abnormal index of
        new coming $Anom_{l+m+1}$, updated distance matrix $D'$
1  Initialize Zero distance array from $x_{l+m+1}$ to $\mathbb{T} \bigcup \mathbb{O}$, $(d_1, d_2, \ldots, d_{l+m})$;
2  Zero distance array from $\mathbb{T} \bigcup \mathbb{O}$ to $x_{l+m+1}$, $\left(d'_1, d'_2, \ldots, d'_{l+m}\right)$;
3  for $i \leftarrow 1$ to $l + m$ do
4      $v_i \leftarrow \text{sum}\left\{D_{i,1}, D_{i,2}, \ldots, D_{i,k-1}\right\}$;
5      $d_i \leftarrow \max\left\{\vec{d}_{hw}(x_{l+m+1}, x_i), h_i\right\}$;
6      $d'_i \leftarrow \vec{d}_{hw}(x_i, x_{l+m+1})$;
7      $D' \leftarrow$ Replace $D_{i,k}$ with $d'_i$ if $d'_i < D_{i,k}$; //Update the distance matrix
8      $\alpha_i \leftarrow v_i + D_{i,k}$; //Compute the nonconformity score
9  $D' \leftarrow$ Add $k$ smallest distances from $\left(d'_1, d'_2, \ldots, d'_{l+m}\right)$ to $D$;
10 $\alpha_{l+m+1} \leftarrow \text{sum}\left\{D'_{l+m+1,1}, D'_{l+m+1,2}, \ldots, D'_{l+m+1,k}\right\}$;
11 for $i \leftarrow 1$ to $l + m + 1$ do
12     $p_i \leftarrow \frac{|\{\alpha_i | \alpha_i \geq \alpha_{l+1}, 1 \leq i \leq l+1\}|}{l+1}$; //Compute the $p$-value
13 $\varepsilon_l \leftarrow \min\{p_1, p_2, \ldots, p_l\}$; //Obtain the dynamic threshold according to
   Equation (3.5)
14 if $p_{l+m+1} < \varepsilon_l$ then
15     $Anom_{l+m+1} \leftarrow true$;
16     $\mathbb{O}' \leftarrow \mathbb{O} \bigcup x_{l+m+1}$; //Update the outlier set
17     if $|\mathbb{O}'| > \varepsilon \cdot (tSize + 1)$ then
18         //Perform the re-do strategy
19         for $i \leftarrow l + 1$ to $l + m$ do
20             if $p_i \geq \varepsilon_l$ then
21                 $Anom_i \leftarrow false$;
22                 $\mathbb{T}' \leftarrow \mathbb{T} \bigcup x_i$; //Update the training set
23 else
24     $Anom_{l+m+1} \leftarrow false$;
25     $\mathbb{T}' \leftarrow \mathbb{T} \bigcup x_{l+m+1}$; //Update the training set
```

datasets in Section 3.4.3) is generated by Piciarelli *et al.* [Piciarelli 2008a], which includes 1000 datasets. In each dataset, 250 trajectories are equally divided into 5 clusters and the remaining 10 trajectories are abnormal (abnormal trajectories are not considered in this experiment). See an example in Figure 3.3(a). Each trajectory is recorded by the locations of 16 sample points. The simulated dataset CROSS and real dataset LABOMNI are contributed by Morris and Trivedi [Morris 2009b, Morris 2009a]. The trajectories in CROSS are designed to be in a four-way traffic intersection as shown in Figure 3.3(b). CROSS includes 1999 trajectories which evenly belong to 19 clusters, and the number of sample points varies from 5 to 23. The trajectories in LABOMNI are obtained from humans walking through a laboratory as shown in Figure 3.3(c). LABOMNI has 209 trajectories from 15 clusters and the number of sample points varies from 30 to 624.



(a) Synthetic Dataset "TS1"


(b) CROSS Dataset

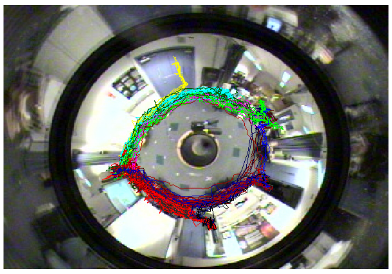

(c) LABMONI Dataset

Figure 3.3: Plots of trajectory datasets used for the evaluation of distance measures. Trajectories in the same cluster have the same color.

Table 3.1 gives the comparison results of the two different distance measures in terms of the average and standard deviation of the classification error ratio. Only the average result of the 1000 datasets in Synthetic Trajectories I is given. From the results on all datasets, we can see that DHD(ω) works better than DHD to measure the difference between trajectories with only having the location information. In particular, in the case of real dataset LABOMNI, the classification performance improves a lot with the use of DHD(ω). Compared with the datasets of Synthetic Trajectories I and CROSS, the trajectories in LABOMNI are more complex in twofold: first, the number of sample points varies more dramatically; second, the trajectories with opposite directions are closer in location, while the trajectories following the same traffic rules in CROSS are distributed in different lanes. In the final column of Table 3.1, the p-value for the null hypothesis that the results from the two distance measures are similar by Kruskal-Wallis test [Kruskal 1952] is presented. The results of Synthetic Trajectories I and LABOMNI are largely different at a 1% significance level, while the difference in CROSS is not so great. Note that Synthetic Trajectories I includes 1000 datasets, thus in conclusion, the results between DHD(ω) and DHD are significantly different. The reason is that using a constraint window to limit the search space helps to reduce the wrong matching and take into account the order of point. Besides, if the data has a small difference in direction, for example, Synthetic Trajectories I and CROSS, DHD(ω) and DHD give a similar distance between trajectories.

Table 3.1: Classification error ratio (%) on different trajectory datasets and the corresponding p-value.

Distance Measures / Datasets	DHD		DHD(ω)		p-Value
	Average	Std	Average	Std	
Synthetic Trajectories I (average)	0.1634	0.0032	0.1566	0.0031	$\ll 0.001$
CROSS	0.6100	0.0792	0.5937	0.0694	0.2182
LABOMNI	31.23	1.37	10.20	0.87	$\ll 0.001$

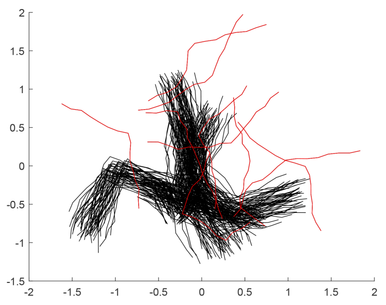
Besides, we compare the distance measures on time series data. In Table A.1, Appendix A, the result also demonstrates that DHD(ω) performs better than DHD.

3.4.2 Comparison of Anomaly Detection Methods

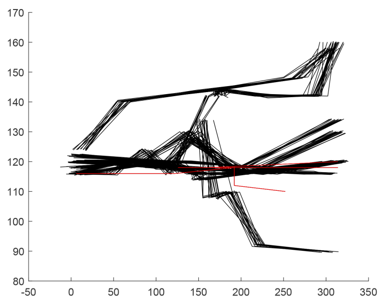
The DH-kNN NCM measure is the core part of SHNN-CAD which computes the nonconformity score and further contributes to calculating the p-value of testing trajectory for classification. In this section, we evaluate the performance of DH-kNN NCM with the utilization of DHD(ω). The same datasets and criteria used in [Laxhammar 2014b] are adopted for comparative analysis, besides, a real dataset from [Guo 2017] is tested.

The 1000 synthetic trajectory datasets mentioned in Section 3.4.1 are the first group of testing data. Note that the outliers in each dataset are also included in the experiment, see Figure 3.4(a). The second dataset consisting of 238 recorded video trajectories is provided by Lazarević *et al.* [Lazarević 2007]. Each trajectory includes 5 sample points and is labeled as normal or not. In this dataset, only 2 trajectories are abnormal, as shown in Figure 3.4(b). The Aircraft Dataset used by Guo *et al.* [Guo 2017] includes 325 aircraft trajectories with the number of sample points varying between 102 and 1023, and 5 trajectories are labeled as abnormal, as shown in Figure 3.4(c). For each dataset, the nonconformity scores of all the trajectories are calculated and sorted. The accuracy of anomaly detection is calculated as the proportion of outliers in the top n nonconformity scores. Here, n is the number of outliers in the dataset according to the groundtruth.

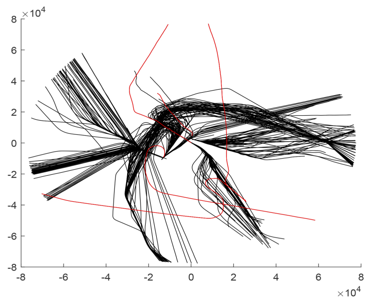
Table 3.2 shows the accuracy performances of DH-kNN NCM and the version with DHD(ω) on 1002 datasets. Only the average result of the 1000 datasets in Synthetic Trajectories I is given. For Synthetic Trajectories I, using DHD(ω) improves the detection quality of DH-kNN NCM regardless of the number of nearest neighbors k. Addi-



(a) Synthetic Dataset "TS1"



(b) Recorded Video Dataset



(c) Aircraft Dataset

Figure 3.4: Plots of trajectory datasets used for the evaluation of anomaly detection measures. The trajectories in red are abnormal.

tionally, with DHD and DHD(ω), the anomaly detection measure works the best when $k = 2$. For Recorded Video Trajectories and Aircraft Trajectories, the replacement of DHD(ω) achieves the same detection result. In the Synthetic Trajectories I, it is clear that the accuracy increases to a maximum then decreases with k increases. This helps to determine the optimal k.

Table 3.2: Accuracy (%) of anomaly detection on different trajectory datasets.

Datasets	**Nonconformity Measures**	**# of Most Similar Neighbors Considered**				
		$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
Synthetic Trajectories I (average)	DH-kNN NCM	96.42	97.09	97.05	96.95	96.77
	using DHD(ω)	96.45	97.85	97.81	97.74	97.65
Recorded Video Trajectories	DH-kNN NCM	100.00	100.00	100.00	100.00	100.00
	using DHD(ω)	100.00	100.00	100.00	100.00	100.00
Aircraft Trajectories	DH-kNN NCM	80.00	80.00	80.00	80.00	80.00
	using DHD(ω)	80.00	80.00	80.00	80.00	80.00

3.4.3 Comparison of Online Anomaly Detection

To demonstrate the efficiency and reliability of the proposed improvement, we compare the performance of SHNN-CAD$^+$ with SHNN-CAD on the same datasets applied in [Laxhammar 2014b] and further introduce more datasets.

The synthetic trajectories [Laxhammar 2013] presented in [Laxhammar 2014b] for online anomaly detection are created by Laxhammar using the trajectory generator software written by Piciarelli *et al.* [Piciarelli 2008b]. Synthetic Trajectories II includes 100 datasets, and each dataset has 2000 trajectories. Each trajectory has 16 sample points recorded with location attribute and has the probability of 1% of being abnormal. To expand the dataset for our experiments, we reuse Synthetic Trajectories I and rename it by Synthetic Trajectories III [Guo 2018b]. The trajectories in each dataset of Synthetic Trajectories I are randomly reordered since they are organized regularly by cluster, which is not common in practical applications. Besides, considering that the Hausdorff distance has the advantage of dealing with trajectory data with different number of sample points, however, in the experiments of [Laxhammar 2014b], all the testing data have same number of points for online learning and anomaly detection, we produce a collection of datasets where the trajectories have various number of points, called Synthetic Trajectories IV [Guo 2018b]. The trajectory generator software [Piciarelli 2008b] is enhanced to produce trajectories with the number of sample points ranging from 20 to 100. For each dataset, firstly, 2000 normal trajectories from 10 equal-size clusters are generated with the randomness parameter 0.7 and are reordered to simulate the real scene. Then 1000 abnormal trajectories are generated. Finally, each normal trajectory is independently replaced with the probability of λ by an abnormal one. The collection has 3 groups of datasets with λ equal to 0.005, 0.01, and 0.02, respectively, and each group contains 100 trajectory datasets.

In [Laxhammar 2014b], $F1$-score is utilized to compare the overall performance of online learning and anomaly detection. $F1$-score (see Equation 2.16) is the harmonic mean of precision and recall (also called detection rate in the field of anomaly detection). In addition to this, we also analyze the false alarm rate and accuracy values for comprehensive evaluation from different aspects [Nandeshwar 2011]. The related calculations have been reviewed in Section 2.3.3.

Firstly, the performance of SHNN-CAD and SHNN-CAD$^+$ is compared using the aforementioned 1400 trajectory datasets. k is set to 2 as suggested in [Laxhammar 2014b]. The average results for each collection of trajectory data are given in Table 3.3 where the best score of each performance measure is highlighted. Note that it is necessary to preset the anomaly threshold ε for SHNN-CAD, thus, we define ε based on the real probability of anomaly λ. For Synthetic Trajectories III, the λ is computed as 10/260 as each dataset has 10 outliers and 250 normal trajectories. Additionally, note that we found a mistake in Table 3 of [Laxhammar 2014b] where the given result of SHNN-CAD is different from the description of **Algorithm 2** in [Laxhammar 2014b]. The F1 result, 53.52, 74.61, and 61.68, of SHNN-CAD with $\varepsilon = 0.005, 0.01,$ and 0.02, respectively, is given under the condition that if p-value $\leq \varepsilon$ not p-value $< \varepsilon$, the testing trajectory is classified as abnormal. In the table below, we follow the rule in **Algorithm 2** [Laxhammar 2014b] for SHNN-CAD.

It is clear from the table that SHNN-CAD works the best in most datasets concerning the F1 score when ε is close to λ, which is consistent with expected and with the description in [Laxhammar 2014b]. When using λ as the anomaly threshold, the accuracy of SHNN-CAD is not always better than the other values of ε. By comparison, SHNN-CAD$^+$ obtains better result with adaptive and dynamic threshold than using λ.

It is important to point out that for unsupervised anomaly detection, λ is not available and no information can be used to help to determine ε. For example, in a different collection of datasets, ε is set with a different value. Besides, SHNN-CAD$^+$ achieves better results in all the datasets with the accuracy index. Thus, SHNN-CAD is less applicable in real-world applications than SHNN-CAD$^+$, which utilizes the adaptive anomaly threshold. Furthermore, the average runtime of dealing with 2000 trajectories in Synthetic Trajectories II is 39.85 s by SHNN-CAD$^+$. For comparison, the typical implementation of DHD with computing distance between every pairwise sample points is equipped to the anomaly detection procedure of SHNN-CAD to get the runtime, which is 128.08 s in this case. For 2000 trajectories in Synthetic Trajectories IV, the runtime becomes longer as 109.34 s by SHNN-CAD$^+$ and 556.28 s by SHNN-CAD with typical DHD implementation. Note that optimal implementations (as the one suggested in [Laxhammar 2014b], which is based on the Voronoi diagram) will improve the result.

Table 3.3: Five performance measures (%) of online anomaly detection. The best performance of each collection of dataset is in bold.

Trajectory Datasets	Approaches		Precision	Recall	F1	Accuracy	False Alarm Rate
Synthetic Trajectories II ($\lambda = 0.01$)	SHNN-CAD	$\varepsilon = 0.005$	**98.70**	40.39	54.75	99.39	**0.01**
		$\varepsilon = 0.01$	87.15	77.48	79.80	99.63	0.13
		$\varepsilon = 0.02$	50.24	**94.59**	64.35	98.98	0.98
	SHNN-CAD$^+$		88.41	89.64	**86.38**	**99.77**	0.13
Synthetic Trajectories III ($\lambda = 0.038$)	SHNN-CAD	$\varepsilon = 0.03$	**97.34**	55.51	67.92	98.19	**0.10**
		$\varepsilon = 0.04$	91.52	73.95	79.36	98.63	0.38
		$\varepsilon = 0.05$	80.01	**83.40**	**79.83**	98.39	1.01
	SHNN-CAD$^+$		84.75	82.70	79.39	**98.68**	0.70
Synthetic Trajectories IV ($\lambda = 0.005$)	SHNN-CAD	$\varepsilon = 0.004$	**90.97**	54.53	63.78	99.73	**0.03**
		$\varepsilon = 0.005$	83.82	65.04	69.40	99.75	0.07
		$\varepsilon = 0.01$	52.63	88.21	64.01	99.53	0.41
	SHNN-CAD$^+$		78.43	**91.76**	**81.47**	**99.82**	0.15
Synthetic Trajectories IV ($\lambda = 0.01$)	SHNN-CAD	$\varepsilon = 0.005$	**99.17**	37.31	52.39	99.33	**0.00**
		$\varepsilon = 0.01$	89.31	74.31	79.31	99.61	0.11
		$\varepsilon = 0.02$	52.27	**92.09**	65.60	99.01	0.92
	SHNN-CAD$^+$		88.64	89.52	**85.66**	**99.75**	0.14
Synthetic Trajectories IV ($\lambda = 0.02$)	SHNN-CAD	$\varepsilon = 0.01$	**98.99**	45.79	61.64	98.91	**0.01**
		$\varepsilon = 0.02$	87.47	83.88	**84.62**	99.42	0.26
		$\varepsilon = 0.03$	63.18	**93.02**	74.53	98.78	1.10
	SHNN-CAD$^+$		95.36	78.75	81.45	**99.48**	0.10

Next, to test the relative performance of each proposed improvement strategy, we conduct experiments based on three objectives. First (Objective 1), to verify that DHD(ω) helps to improve the performance of SHNN-CAD, SHNN-CAD is implemented with DHD(ω) computing the distance between trajectories. Second (Objective 2), to demonstrate the rationality of the re-do step, SHNN-CAD is equipped with a re-do step in the procedure of anomaly detection. Third (Objective 3), to prove the effectiveness of the adaptive anomaly threshold, the pre-definition of ε is replaced in SHNN-CAD. The results are listed in Table 3.4. For the task of objective 1 and objective 2, the results are compared with those by SHNN-CAD in Table 3.3. In the case of objective 1, the utilization of DHD(ω) improves the behavior of anomaly detection regardless of most performance measures for all the datasets. In the case of objective 2, only the recall index for some datasets is not as well as SHNN-CAD, which means the missing recognition of outliers. However, the comprehensive F1 score indicates that the performance is promising. In the case of objective 3, the results are compared with SHNN-CAD in Table 3.3 when ε is closest to the corresponding λ. Clearly, for most datasets, the adaptive anomaly threshold can make up the shortcoming of pre-definition and strengthen the capability of anomaly detection. Compared with the SHNN-CAD$^+$ in Table 3.3, all the improvement strategies work together to accomplish the enhancement of SHNN-CAD.

Table 3.4: Five performance measures (%) of proposed improvement strategies on different trajectory datasets.

Trajectory Datasets	Approaches		Precision	Recall	F1	Accuracy	False Alarm Rate
Synthetic Trajectories II ($\lambda = 0.01$)	Objective 1	$\varepsilon = 0.005$	98.86	40.23	54.89	99.38	0.01
		$\varepsilon = 0.01$	87.93	77.91	80.38	99.64	0.12
		$\varepsilon = 0.02$	50.77	95.16	64.89	98.99	0.97
	Objective 2	$\varepsilon = 0.005$	98.70	40.39	54.75	99.39	0.01
		$\varepsilon = 0.01$	87.15	77.48	79.80	99.63	0.13
		$\varepsilon = 0.02$	50.24	94.59	64.35	98.98	0.98
	Objective 3		87.08	87.21	84.97	99.75	0.13
Synthetic Trajectories III ($\lambda = 0.038$)	Objective 1	$\varepsilon = 0.03$	97.01	55.78	67.99	98.21	0.10
		$\varepsilon = 0.04$	91.86	74.31	79.81	98.66	0.37
		$\varepsilon = 0.05$	80.41	83.94	80.30	98.43	1.00
	Objective 2	$\varepsilon = 0.03$	97.34	55.51	67.92	98.19	0.10
		$\varepsilon = 0.04$	91.52	73.95	79.36	98.63	0.38
		$\varepsilon = 0.05$	80.01	83.40	79.83	98.39	1.01
	Objective 3		85.10	82.25	79.02	98.64	0.72
Synthetic Trajectories IV ($\lambda = 0.005$)	Objective 1	$\varepsilon = 0.004$	93.88	58.22	67.45	99.75	0.02
		$\varepsilon = 0.005$	87.23	69.39	73.28	99.78	0.06
		$\varepsilon = 0.01$	52.75	91.08	65.01	99.54	0.41
	Objective 2	$\varepsilon = 0.004$	90.97	54.53	63.78	99.73	0.03
		$\varepsilon = 0.005$	83.82	65.04	69.40	99.75	0.07
		$\varepsilon = 0.01$	52.63	88.21	64.01	99.53	0.41
	Objective 3		77.90	86.10	78.30	99.79	0.14
Synthetic Trajectories IV ($\lambda = 0.01$)	Objective 1	$\varepsilon = 0.005$	99.53	37.74	53.20	99.34	0.00
		$\varepsilon = 0.01$	92.35	77.94	82.84	99.68	0.08
		$\varepsilon = 0.02$	53.80	94.76	67.59	99.07	0.89
	Objective 2	$\varepsilon = 0.005$	99.17	37.31	52.39	99.33	0.00
		$\varepsilon = 0.01$	89.31	74.31	79.31	99.61	0.11
		$\varepsilon = 0.02$	52.27	92.09	65.60	99.01	0.92
	Objective 3		87.55	84.08	82.63	99.69	0.14
Synthetic Trajectories IV ($\lambda = 0.02$)	Objective 1	$\varepsilon = 0.01$	99.60	45.80	61.70	98.92	0.01
		$\varepsilon = 0.02$	90.18	86.59	87.38	99.52	0.20
		$\varepsilon = 0.03$	65.68	95.28	77.02	98.91	1.02
	Objective 2	$\varepsilon = 0.01$	98.99	45.79	61.64	98.91	0.01
		$\varepsilon = 0.02$	87.47	83.88	84.62	99.42	0.26
		$\varepsilon = 0.03$	63.18	93.02	74.53	98.78	1.10
	Objective 3		95.05	72.66	77.94	99.37	0.10

3.5 Conclusions

Based on SHNN-CAD, which focuses on online detecting outliers from trajectory data, we have presented an enhanced version, called SHNN-CAD$^+$, to improve the anomaly detection performance. The proposal includes three improvement strategies: first, modifying typical point-based Hausdorff distance to be suitable for trajectory data and to be faster in distance calculation; second, adding a re-do step to avoid false positives in the initial stages of the algorithm; third, defining data-adaptive and dynamic anomaly threshold rather than a preset and fixed one. Experimental results on both real-world and synthetic data have shown that the performance of the presented approach has been improved over SHNN-CAD concerning accuracy and runtime. Especially, with the adaptive and dynamic anomaly threshold, the accuracy of SHNN-CAD$^+$ is higher than that of SHNN-CAD by using the optimal but fixed one. Besides, on average, the runtime is around 4 times faster. Considering that the training set will increase a lot with time, further research will focus on incremental learning which prunes the historical data for future process.

CHAPTER 4

Map Construction From GPS Trajectory Data by a Three-Step Framework

Contents

4.1 Introduction

Nowadays, more and more devices (smartphones, smartwatches, bracelets, etc.) are equipped with GPS to track moving objects. The increasing popularity of GPS tracking devices leads to a massive amount of trajectory data generated every day. Analyzing these data helps to understand the mobility of users and provides the basis of navigation and recommendation systems. From different communities, many research works are devoted to mining the information from trajectory data on various topics, such as anomaly detection, trajectory clustering, and automatic map construction. In this chapter, we focus on the map construction problem.

The traditional way to produce a road map requires a large number of images from satellite, photos from providers and platforms, massive field surveys, and intensive post-processing. This work is labor-expensive and time-consuming, also faces technical challenges [Chen 2008, Li 2016, Deng 2018, Huang 2018]. Developing algorithms to generate road networks from massive GPS data is an alternative. In addition, the generated map can act as a complement of the existing maps considering different cases. First, due to lacking up-to-date information on the buildings, streets and open areas, the existing maps may include incorrect and incomplete road networks [Zheng 2018]. Second, because of inevitable human and technical errors, the existing maps may be inaccurate [Guo 2007]. Third, the roads formed by people can also be interesting and useful. For example, for outdoor activities, the unknown route made by hikers may provide insights into a more efficient and attractive path.

Various approaches have been proposed to generate the road networks. As reviewed in Section 2.4, each category of map construction algorithm has its characteristics. The process of incremental-track-insertion-based algorithms is irreversible, so the order of adding a new trajectory to an existed map influences the result. The intersection-linking-based approaches require heavy work in linking intersections via visiting each trajectory. By comparison, the density-based approaches in the category of point clustering are simpler. In general, there are two common limitations: first, the mechanism is time-consuming due to largely using the clustering technique or kernel density estimation on numerous trajectory points; second, the low-frequency areas are ignored since the trajectories are regarded as unimportant [Cao 2009, Wang 2015]. In this chapter, we aim at developing a three-step framework that is fast, simple and skeleton-based to extract a route graph from the input data that records almost all the paths appearing in the original GPS trajectories. The three steps are: First, GPS data are driven into a grid that covers the geographical area to build a density surface. Second, trajectories are adjusted by the Slide tool according to the density surface, and a new, more compact density surface is computed with the adjusted data. Third, the route graph is extracted from the density surface using a thinning algorithm followed by a polygonal line simplification method. The main novelty of our three-step framework is a smart combination of several well-known techniques to extract a schematic route graph that records almost all the paths in the original GPS data. In addition, all the steps of our framework are easily parallelizable due to the processing objects (either trajectories or links) are independent and the process does not require cross computation, so they are run in parallel to achieve high efficiency.

The remainder of this chapter is organized as follows. Section 4.2 introduces the algorithms for comparison, the Slide tool and the thinning algorithm which are utilized in our algorithm. Section 4.3 presents the three-step framework in detail. Section 4.4 describes the experiments and comparison analysis on different GPS datasets. Finally, Section 4.5 gives the conclusion and future work.

4.2 Preliminaries

In this section, we first introduce seven algorithms for comparison in the experiments, then we describe two important techniques for our framework, the Slide tool and the thinning algorithm.

4.2.1 Algorithms for Comparison

In the case of urban data, following the survey [Ahmed 2015c], we compare our framework with the algorithms by Ahmed and Wenk [Ahmed 2012], Biagioni and Eriksson [Biagioni 2012b], Cao and Krumm [Cao 2009], Davies *et al.* [Davies 2006], Edelkamp and Schrödl [Edelkamp 2003] and Karagiorgou and Pfoser [Karagiorgou 2012]. In the case of hiking data, following the survey [Duran 2020], we compare with algorithms by Ahmed and Wenk [Ahmed 2012], Cao and Krumm [Cao 2009], Davies *et al.* [Davies 2006], Edelkamp and Schrödl [Edelkamp 2003] and Karagiorgou and Pfoser [Karagiorgou 2012]. In both cases, a density-based algorithm by Wang *et al.* [Wang 2015] is compared. In this section, we briefly introduce these algorithms which are also mentioned in the Section 2.4.

4.2.1.1 Algorithm by Ahmed and Wenk [Ahmed 2012]

This algorithm is a two-phase incremental framework. For each iteration, given the constructed map from the last iteration (or an empty map in the first iteration), the first phase focuses on computing a reconstructed graph. The new trajectory is compared with the constructed map based on the curve-graph partial matching [Buchin 2009] to separate the trajectory into matched and unmatched portions. The unmatched portions are added to create or split edges and introduce new vertices. In the second phase, the minimum-link algorithm is applied to the matched portions with their matching edges to obtain minimum-link representative edges, which contributes to reducing the graph complexity.

4.2.1.2 Algorithm by Biagioni and Eriksson [Biagioni 2012b]

Biagioni and Eriksson [Biagioni 2012b] inferred a map taking into account GPS data with noise and disparity. KDE is utilized to estimate the density image and a gray-scale thinning algorithm with various thresholds is applied to compute several versions of a skeleton map. The last one is represented as an undirected graph whose edges are finally replaced with directed edges. Considering that KDE estimates the density at each point by using all the points in the dataset, this method is a time-consuming algorithm. In addition, due to the fact that the trajectories distribute non-uniformly, the infrequent paths may be eliminated with the thresholding process.

4.2.1.3 Algorithm by Cao and Krumm [Cao 2009]

This method also belongs to the category of incremental track insertion. It aims to generate a directed graph but pays more attention to firstly remove noise from trajectories. For each point, two types of attraction forces are simulated from its corresponding trajectory and the others in the dataset to pull the similar points to be closer. For the directed graph construction, the opposite directions are also differentiated. Then a simple graph generation algorithm is performed. To insert a new trajectory, each of its points is checked to be merged or used for creating a new graph node. After the map generation, the edges with weight less than a threshold are removed since they are considered as unreliable. Apparently, the infrequent paths are not included, leading to low coverage of the map.

4.2.1.4 Algorithm by Davies *et al.* [Davies 2006]

Davies *et al.* [Davies 2006] estimated the density distribution by counting the number of trajectories passing through each grid cell and computed the contour of the resulting bitmap. Then the Voronoi diagram is used to determine the centerline which is cleaned-up to obtain a directed graph representing the street map. This method has the problem of ignoring paths with low frequency.

4.2.1.5 Algorithm by Edelkamp and Schrödl [Edelkamp 2003]

This algorithm is point clustering-based. Firstly, the trajectories are decomposed into a sequence of line fragments, then k-means clustering is utilized on these fragments. Next, a number of equal-distance points following each trace are taken by a greedy strategy as the cluster seeds. These seeds are regarded as the vertices in the route graph and the rest of trajectories are considered is to link them together.

4.2.1.6 Algorithm by Karagiorgou and Pfoser [Karagiorgou 2012]

Karagiorgou and Pfoser [Karagiorgou 2012] also proposed a three-step framework (called TRACEBUNDLE algorithm in [Karagiorgou 2013]) for automatic map construction specifically for vehicle trajectory data. The intersections are found in the first step through performing two times hierarchical clustering respectively on turns and the corresponding clusters. Turns are the points where the speed reduces and the direction changes largely. In the second step, links between intersections are connected via using a sweep-line algorithm on all the trajectories. In the final step, the number of links is reduced by using the sweep-line algorithm again, which includes the position change of intersections. Based on this algorithm, the authors proposed to create the maps for different types of movement (distinguished by speed) and then the maps are combined into a single network. With the clustering technique, the points in low-density areas where only a few trajectories cross through are removed as outliers. As a result, the roads with infrequent trajectories are ignored. Furthermore, this method is time-consuming since there is a large computation cost for clustering and linking.

4.2.1.7 Algorithm by Wang *et al.* [Wang 2015]

Wang *et al.* [Wang 2015] proposed a density-based map construction algorithm. In the first step, KDE is applied to obtain the density distribution. Then, the route graph is extracted as the mountain ridges of the terrain (density distribution) through discrete Morse theory. The second step is quite fast while the KDE computation in the first step is really heavy if the size of each cell is small but the data size is big. In addition, it removes the unimportant roads with a persistence threshold, which would eliminate infrequent paths.

4.2.2 Slide

In general, moving objects do not strictly follow the centerline of roads, leading to the difference in trajectory data. To build a compact density surface, the trajectories that follow similar but non-identical paths should be brought together. In this chapter, we use the Slide tool to achieve this goal.

Slide is a heuristic method proposed by Paul Mach from the Strava labs [Mach 2014b, Mach 2014a]. Slide was first developed as a tool to slide OpenStreetMap map geometry to the Strava global heatmap dataset. GPS data are continuously collected to create the heatmap which essentially represents the density distribution of data. The key idea of Slide is to match and merge the input polyline to the heatmap data, which is based on the principles of mathematical optimization. Figure 4.1 gives the flowchart of Slide (taken from [Mach 2014a]).

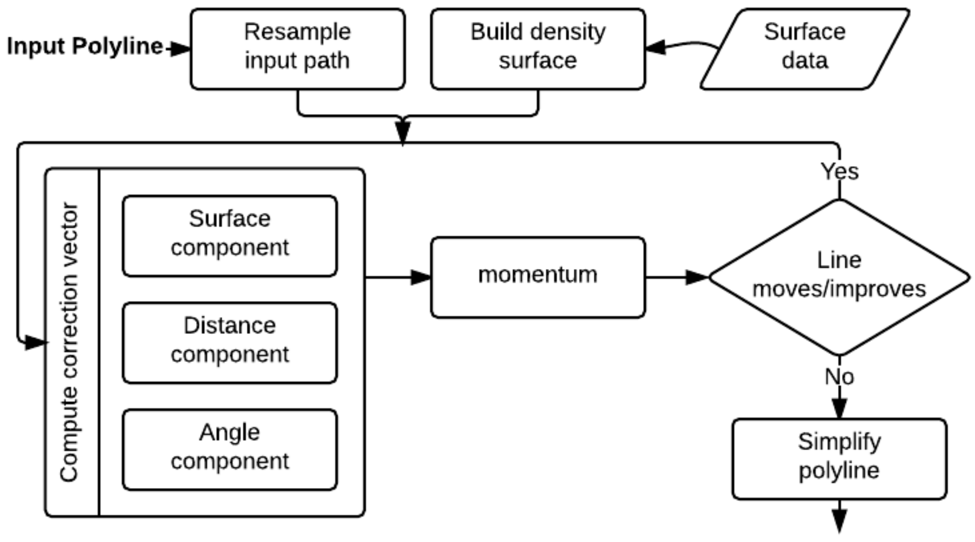


Figure 4.1: Flowchart of Slide.

In the flowchart, the surface data is the Strava global heatmap which is used to build density surface where high density regions are lower (resulting in valleys). Slide makes the input polyline (we use “trajectory” to specify this polyline) fall (or slide) into the surface valleys by making force on each point. The input trajectory is resampled to have equal distance between adjacent points, which would be further used in the cost function of Slide. More precisely, given a trajectory $T = (p_1, p_2, \ldots, p_n)$ with n points, each interior point $p_i\,(1 < i < n)$ of T is iteratively perturbed by adding a correction vector to its current position, $\mathbf{cr}(p_i)$, which is defined as a weighted sum of the surface ($\mathbf{s_V}$), distance ($\mathbf{d_V}$), angle ($\mathbf{a_V}$) and momentum ($\mathbf{m_V}$) vector components with different weights:

$$\mathbf{cr}(p_i) = \omega_1 \mathbf{s_V}(p_i) + \omega_2 \mathbf{d_V}(p_i) + \omega_3 \mathbf{a_V}(p_i) + \omega_4 \mathbf{m_V}(p_i). \tag{4.1}$$

where ω_1, ω_2, ω_3 and ω_4 are the weights of corresponding components. Given three consecutive points, p_{i-1}, p_i and p_{i+1}, of a trajectory, let $\mathbf{u} = p_{i+1} - p_{i-1}$, $\mathbf{v} = p_i - p_{i-1}$ be the difference between neighbor points. According to the public implementation of Slide [Mach 2014a], we summarize the movements in each iteration by these components for p_i by the following formulas.

- The surface component calculates the movement of p_i with respect to the depth of the density surface. Intuitively, in order to make similar trajectories more compact, the point should move to the dense part. In Slide, the definition is $\mathbf{s_V}(p_i) = \mathit{gradientAt}(p_i)$ which means the surface gradient at p_i. This correction drags p_i to its closest valley. As the surface is represented in a two-dimensional grid, the bilinear interpolation [Smith 1981] is applied to approximately estimate the gradient of a certain point. Figure 4.2 illustrates the effect of this component. p_i will be pushed to p_i' matching the ridges of the surface.

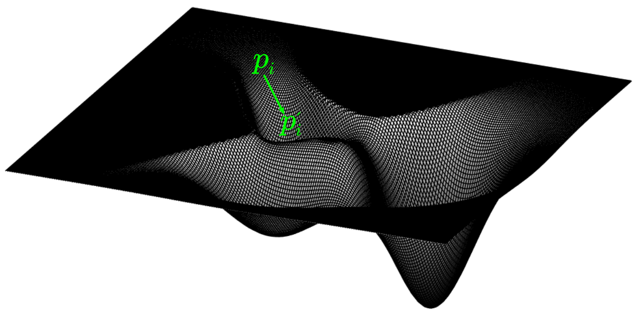


Figure 4.2: The effect of the surface component on a point.

- The distance component tries to ensure that p_i does not change too much from its previous position by maintaining the distance to its neighbors, considering that initially the trajectory is resampled to have equal distance between adjacent points. To achieve this, the vector is computed by

$$\mathbf{d_V}(p_i) = \begin{cases} \mathbf{0}, & p_{i-1} = p_{i+1} \\ \mathbf{m_1} + \mathbf{m_2}, & \mathit{otherwise} \end{cases} \tag{4.2}$$

where $\mathbf{m_1} = p_{i-1} - \mathbf{center}$, $\mathbf{m_2} = p_{i+1} - \mathbf{center}$, $\mathbf{center} = \mathrm{p}_{i-1} + \mathbf{u}\left(\frac{\mathbf{u}\cdot\mathbf{v}}{\mathbf{u}\cdot\mathbf{u}}\right)$. Here, the operation $\cdot$ denotes the scalar product.

- The angle component maximizes the vertex angle and minimizes curvature. The calculation is

$$\mathbf{a}_\mathbf{V}(p_i) = \begin{cases} 0, |u - v| = 0 \ or \ \delta = 0 \\ \delta \frac{\mathbf{u}-\mathbf{v}}{|\mathbf{u}-\mathbf{v}|} \min\{|\mathbf{v}|, |\mathbf{u}|\}, \quad otherwise \end{cases} \tag{4.3}$$

  where $\delta = 1 - \sqrt[3]{\mathbf{u} \cdot \mathbf{v}}$ and $|\ |$ is Euclidean norm.

- The momentum component, which is the correction vector used in the previous iteration, is additionally added to speed the convergence of the process.

In each iteration, every interior point of a trajectory is moved according to its correction vector. It prompts the interior part of the trajectory to a denser part of the surface, meanwhile, it maintains the endpoints in their original position. Slide terminates when the solution converges, i.e. the improvement between two consecutive iterations is smaller than a prefixed threshold (usually $5 \cdot 10^{-4}$). The improvement is measured as the difference in the score of the trajectories. The score of a trajectory is defined as the sum of the density value of all the points at the density surface.

The fact that Slide does not move the endpoints of a trajectory is, in many cases, a problem. We need the endpoints to move to a denser part of the surface in a similar way the interior points do. On the other hand, according to our experimentation and interpretation of the distance component has too much influence on the correction vector. These two important issues of the standard implementation of Slide and their impact in the results are explained in Section 4.3.3. There we also present our modified version of Slide.

4.2.3 Thinning Algorithm

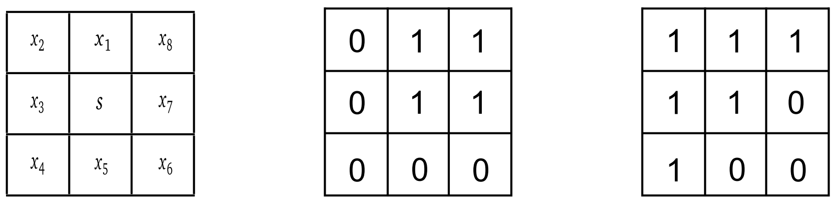
In the field of image processing, the thinning algorithm is a popular morphologic operation to extract a one-pixel-wide skeleton from the binary image by iteratively removing several foreground pixels. A comprehensive survey of thinning methodologies can be found in [Lam 1992]. In this chapter, we briefly describe the thinning algorithm from Guo and Hall [Guo 1989] which is the one used in this thesis.

In each iteration, the algorithm visits all the pixels of the image and each pixel is determined to be deleted or not based on a two-subiteration process. Let s be the pixel under consideration and $\{x_1, x_2, \ldots, x_8\}$ be the values of the 8-connected neighbors of s (as shown in Figure 4.3(a)), following in counter-clockwise direction. There are four conditions in the thinning algorithm to determine if a pixel with value 1 and have eight neighbors should be removed or not. The condition 1 is:

$$X_H(s) = 1 \tag{4.4}$$

where

$$X_H(s) = \sum_{i=1}^{4} b_i \tag{4.5}$$



(a) Nine pixels (b) First subiteration. (c) Second subiteration.

Figure 4.3: Illustration of thinning algorithm.

and

$$b_i = \begin{cases} 1, & x_{2i-1} = 0 \ \& \ (x_{2i} = 1 \ || \ x_{2i+1} = 1) \\ 0, & otherwise. \end{cases} \tag{4.6}$$

Here $X_H(s)$ is the number of transitions from 0 to 1 in the sequences (x_1, x_2, x_3), (x_3, x_4, x_5), (x_5, x_6, x_7), (x_7, x_8, x_1). The condition 2 is:

$$2 \leq \min\{n_1(s), n_2(s)\} \leq 3 \tag{4.7}$$

where

$$n_1(s) = \sum_{k=1}^{4} x_{2k-1} \vee x_{2k} \tag{4.8}$$

and

$$n_2(s) = \sum_{k=1}^{4} x_{2k} \vee x_{2k+1} \tag{4.9}$$

This condition ensures that at least one pixel in the 4-neighbors of s has value 0, and so do the diagonal neighbors. If all the pixels of the 4-neighbors or diagonal neighbors have value 1, then removing s breaks the connectivity of skeleton. The condition 3 is:

$$(x_2 \vee x_3 \vee \overline{x_8}) \wedge x_1 = 0. \tag{4.10}$$

The condition 4 is:

$$(x_6 \vee x_7 \vee \overline{x_4}) \wedge x_5 = 0. \tag{4.11}$$

In the first subiteration, pixel s is removed if and only if it satisfies all of the conditions (1, 2 and 3) (see Figure 4.3(b) as an example). In the second subiteration, pixel s is deleted if and only if it meets all of the conditions (1, 2, and 4) (see Figure 4.3(c) as an example).

4.3 The Proposed Method

This section starts with a pre-processing step where obvious noise in the input data is removed, then describes our proposed three-step approach. The three steps of this algorithm are i) *building density surface:* GPS data are driven into a grid to build a density surface. ii) *compacting density surface via Slide:* trajectories are adjusted according to the density surface and the Slide tool, then a new and compact density surface is computed with the adjusted trajectories. iii) *constructing route map:* The route graph is extracted from the density surface using a thinning algorithm followed by a polygonal line simplification method. Besides, in Section 4.3.5, we discuss the parameters related to our method.

4.3.1 Data Pre-processing

In practice, GPS data always includes noise. In this paper, the noise is caused by signal missing due to severe environment or rapid change of position, GPS errors, or signal distortion. For example, vehicle trajectories are noisier than hiking data, because the high speed may induce missing or wrong positions in the record. To reduce the impact of noise, for each trajectory, we check the distance between adjacent points. If the distance is too large, this is regarded as an error of signal absence, then the trajectory is divided into two parts by removing the corresponding line segment.

Next, trajectories are resampled by using linear interpolation so that they become defined by equidistant points, which is required in the Slide algorithm for the computation of the correction vector. Moreover, we use the term *fragment* of a trajectory t, to refer to the trajectory t' defined by a sub-sequence of the points defining t. Hence a fragment will be defined by two (or more) trajectory points.

4.3.2 Density Surface Computing

To compute the density surface, first, the area occupied by the given trajectories is covered by a regular grid. Then we map each trajectory into the grid. As shown in Figure 4.4, a trajectory with 9 points from p_1 to p_9 is mapped into several consecutive cells. Green cells correspond to the points defining the trajectory and the gray ones to the line joining the points. We include both the green and gray cells to keep the complete shape of the trajectory. This is important when the distance interval is larger than the size of the cell.

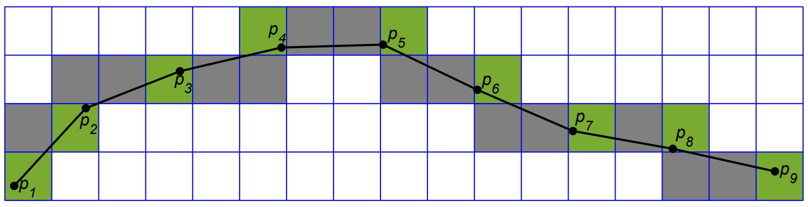


Figure 4.4: Mapping a trajectory into the grid.

With all the trajectories mapped into the grid, we build the density surface by counting the frequency of each cell being crossed by GPS data. Furthermore, to eliminate noise and artifacts, we apply the Gaussian blur [Gonzalez 2006] method on this initial density surface. Then a smoother density surface is obtained. Since the treatment of each trajectory is independent of the others, we map each trajectory into the grid in parallel to speed up the process.

4.3.3 Density Surface Compaction via Slide

To be able to aggregate similar parts of different trajectories into a single edge of a graph, we apply an improved version of the basic Slide method, the one described in Section 4.2.2, to each trajectory in the dataset. After that with the adjusted trajectories, the density surface is recomputed to be more compact.

As we mentioned before, according to our experimentation we found two limitations on Slide. In this section, we explain the defects and our solutions to overcome these shortcomings.

On the one hand, the distance component of Slide aims to maintain the equal distance between adjacent points, which helps to prevent the point from significantly deviating from its original position. Consistent with the symbols in Section 4.2.2, let p_{i-1}, p_i and p_{i+1} be three consecutive points of a trajectory, and $\mathbf{u} = p_{i+1} - p_{i-1}$, $\mathbf{v} = p_i - p_{i-1}$ be the difference between neighbor points. Considering the case that $p_{i-1} \neq p_{i+1}$, the correction vector by this component in Equation 4.2 can be interpreted with the geometric definition by

$$\begin{aligned} \mathbf{d}_{\mathrm{V}}(p_i) &= \mathbf{u} - 2\mathbf{u}\left(\tfrac{\mathbf{u}\cdot\mathbf{v}}{\mathbf{u}\cdot\mathbf{u}}\right) \\ &= 2\mathbf{u}\left(\tfrac{1}{2} - \tfrac{|u_0|}{|u|}\right) \end{aligned} \tag{4.12}$$

where $\mathbf{u}\cdot\mathbf{v} = |u||v|\cos\theta$ as illustrated in Figure 4.5.

It can be geometrically seen that $\mathbf{u}(\frac{1}{2} - \frac{|u_0|}{|u|})$ is vector $\mathbf{d}$ of Figure 4.5. To maintain the distance between points while adjusting them, p_i should be moved by $\mathbf{d}$. But according to Equation 4.12 of Slide, p_i is moved twice this amount. To preserve consistency between the intention and the used formula, we propose to define the correction vector of the distance component that p_i is moved only by $\mathbf{d}$. Hence, we divide by a factor of 2 of the original formula. We have been experimentally tested this change, and this new version gives better results. The same final effect could also be obtained by using half of the standard weight, i.e. $\omega_2/2$. However, for the mentioned consistency we propose to consider $\mathbf{d}_{\mathrm{V}}(p_i)$ to be defined as:

$$\mathbf{d}_{\mathrm{V}}(p_i) = \begin{cases} \mathbf{0}, & p_{i-1} = p_{i+1} \\ \frac{1}{2}(\mathbf{m_1} + \mathbf{m_2}), & otherwise \end{cases} \tag{4.13}$$

The second modification of the Slide tool involves endpoints. The original version of Slide, presented in Section 4.2.2, does not move the endpoints of the trajectories because the correction vector only acts on the interior points. Since the position of a point has a large influence on the distance and angle components of their neighbor points, the neighbors of the trajectory endpoints will not move properly and the movement of an important part of the trajectory will also be effected gradually. In practice, the parts

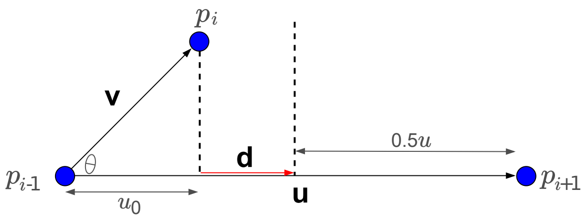
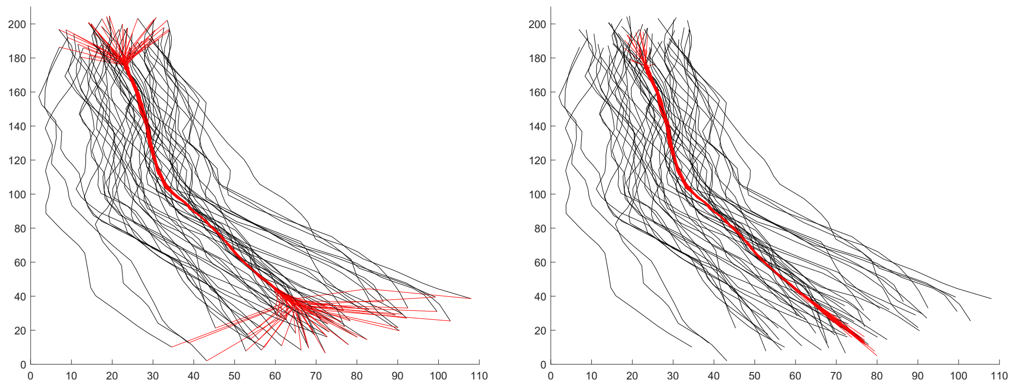


Figure 4.5: Geometric definition of the distance component of Slide.

of the trajectories near the endpoints present undesirable sharp changes. This aspect was already presented as a problem by the author of Slide [Mach 2014b]. Figure 4.6(a) shows an example using a synthetic dataset provided by [Piciarelli 2008a] showing these undesired shapes of the adjusted trajectories nearby their endpoints. The input dataset includes 50 trajectories that are rather similar to each other, consequently, Slide should adjust them until they are merged in a single trajectory. However, most parts of the adjusted trajectories move to the dense area, but the initial and final parts of the trajectories do not because they are anchored to the original endpoints.



(a) Result by the original Slide. (b) Result by the improved Slide.

Figure 4.6: The effect of (a) the original and (b) the improved Slide on a set of trajectories (in black). The adjusted trajectories are in red.

To solve this problem, we propose to move the endpoints, in each iteration, according to the movement of their neighbors. Let us consider p_1, p_2 and p_3, the first three points of a trajectory. The correction vector proposed by Slide is applied to the interior points of the trajectory, p_2 and p_3. Let p_2' and p_3' be the resulting points. Then, the endpoint p_1 is projected onto the line defined by $p_2'p_3'$ (see Figure 4.7), this orthogonal projection defines p_1'. Similarly, the other endpoint p_n is projected onto the line segment defined by p_{n-2}' and p_{n-1}'. Proceeding in this way the trajectories correctly meet at the denser part of the density surface (see Figure 4.6(b)). By experiments, we first tried to guarantee the points to be equidistant by placing p_1' on the line defined by p_2' and p_3' so that p_2' was the midpoint of p_1' and p_3'. However, the obtained results trajectories were not as good as expected.

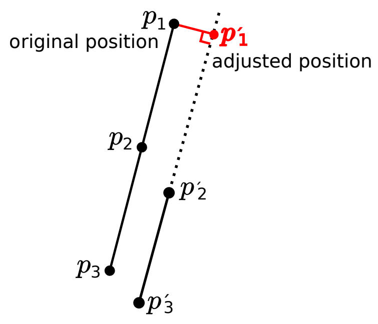
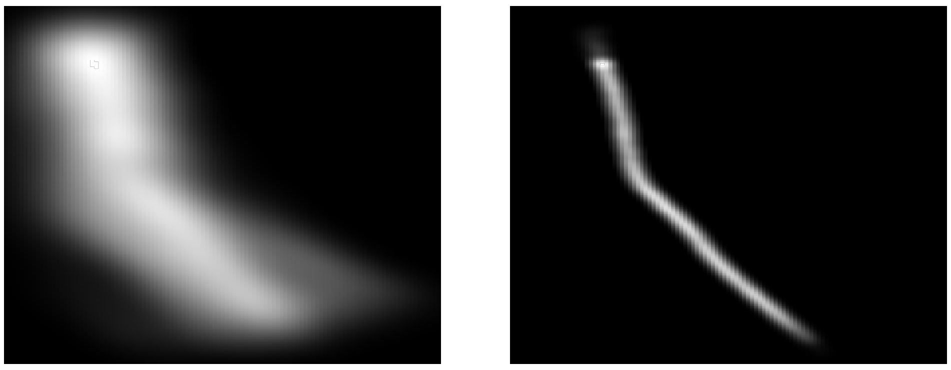


Figure 4.7: Illustration of determining the position of the endpoint.

Finally, after applying Slide to each trajectory, the density surface is recomputed with the adjusted trajectories proceeding as did in the previous step. As shown in Figure 4.8, the density distribution of the synthetic data [Piciarelli 2008a] becomes more compact, tending to get similar trajectories closer to each other. This recomputed density function is the one that will be used in the remaining phases.



(a) Original density surface. (b) Re-computed density surface.

Figure 4.8: Density surface obtained (a) before and (b) after using Slide. The pixel with higher density is more bright.

We remark that since Slide works independently on each trajectory, our method adjusts all the trajectories in parallel to reduce computation time.

4.3.4 Route Map Construction

The route map is modeled as an undirected graph comprising the vertices and edges set. Each vertex has a geographical coordinate. Each edge represents a line segment of a route in the map and connects two vertices.

The density surface obtained based on Slide is first transformed into a binary image through global thresholding [Gonzalez 2006]. Next, a thinning algorithm is applied to extract a one-pixel-wide skeleton of the binary image. Similar to [Biagioni 2012b, Shi 2009], we obtain the initial graph from the skeleton and then apply the line simplification algorithm by [Douglas 1973] to remove redundant vertices from each edge and obtain the final simplified route graph. To speed up the computation, the simplification algorithm is also run in parallel on each edge.

4.3.5 Discussion about Parameters

In the presented framework, several parameters have to be set. Table 4.1 gives an overview of these parameters and separates them into fixed setting and variable setting groups. Fixed setting means the parameters are the same in all the datasets. Variable setting indicates the parameters are set by experiments, and along this section we give hints on how they can be set.

Table 4.1: Overview of parameters

	parameter	symbol	setting
Fixed setting	weights in Slide	ω_1,ω_2,ω_3,ω_4	0.5,0.2,0.1,0.7
	threshold of path score in Slide	u	$5 * 10^{-4}$
	standard deviation of Gaussian blur for initial density distribution	σ_1	5
	distance for Douglas-Peucker	d	2 meters
Variable setting	resample distance	κ	set via experiments
	cell size	τ	
	standard deviation of Gaussian blur for re-computed density distribution	σ_2	
	threshold to convert binary image	ε	

All the parameters of Slide are from the original implementation. Concerning the distance threshold in Douglas-Peucker, it is the same meter resolution in all the cases because we use the hiker and vehicle data, and we are interested in obtaining a route map with the same precision.

Concerning the other parameters, if the input data is similar to one of the datasets used in this paper in noise and size, the same settings can be adopted. If not, since our framework is fast, the parameters are quite easy to tune experimentally taking into account the following considerations. Parameter κ specifies the distance between two consecutive points. As expected, a larger κ value results in a faster computation, but faces the problem of over-smoothing, losing a lot of information. We set κ to be similar to the average distance between consecutive data points. Parameter τ determines the resolution of the map, and the smaller it is, the more details the map will record. For noisy data like urban data, it has to be bigger and for the less-noisy data like hiking data smaller. τ is limited to be 1, 2 or 3 meters. Regarding σ_2, it specifies how much the density surface should be smoothed. It is set between 2 to 5. To obtain good results, the noisier the data, the bigger σ. Finally, ε is the threshold used to convert the density surface to a binary image. Paths with density higher than ε are preserved, the others are ignored. High values of ε may cause breaks in several paths of the map.

4.4 Results and Discussion

This section shows extensive experiments on different GPS datasets. We begin by introducing the two types of data applied in the experiments in Section 4.4.1, then do visual inspection on the generated maps in Section 4.4.2. Next, we compare the performance of different algorithms by a quantitative evaluation in Section 4.4.3. Since our framework is easy to parallelizable, Section 4.4.4 lists the runtime of performing our framework in parallel and with a single core. Finally, we give a summary of the performance in Section 4.4.5.

All the experiments are conducted by MATLAB 2018a software running on a Debian GNU/Linux machine with AMD Ryzen Threadripper 1950X 16-Core Processor and 32 GB RAM.

4.4.1 GPS Datasets

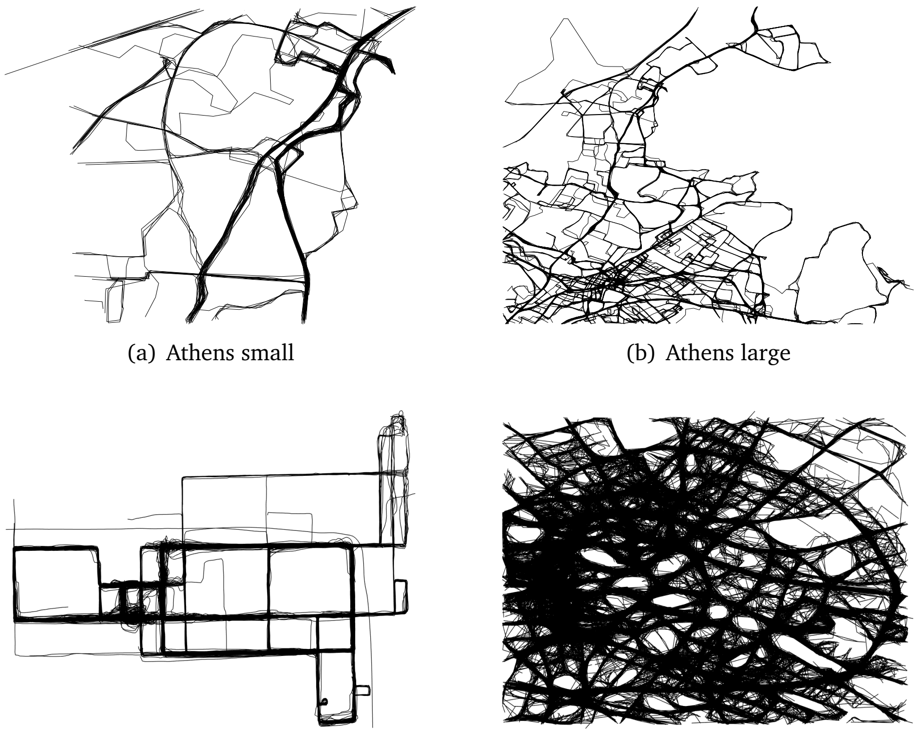
In this chapter, we use 8 real GPS datasets of two types, 4 urban datasets and 4 hiking datasets, to carry out the experiments. These datasets have been widely used in the literature, for example in [Ahmed 2015c] and [Duran 2020]. [Ahmed 2015c] gives a detailed comparison of several algorithms based on different distance measures using the urban datasets. Meanwhile, in [Duran 2020], Duran *et al.* give a local analysis of artifacts in maps generated when using both urban and hiking data. We borrow their results of Ahmed, Biagioni, Cao, Davies, Edelkmap and Karagiorgou methods. Besides, we compare our framework with [Wang 2015], a recent density-based method. In this case, we have used their public implementation [Wang 2016] to run their algorithm. In this case, we have tuned the parameters to obtain the best results. The setting of parameters are summarized in Table 4.2.

Table 4.2: Overview of parameters of our framework and Wang *et al.*

Datasets	**Our Framework**				**Wang *et al.***		
	κ	τ	σ_2	ε	r	t	δ
Athens small	10	3	3	0.05	5	400	0.006
Athens large	10	3	3	0.05	5	400	0.006
Chicago	10	3	3	0.05	10	400	0.0007
Berlin	100	3	5	5	10	400	0.002
Delta	10	1	2	0.05	3	25	0.02
Aiguamolls	5	2	3	0.05	5	400	0.005
Garraf	5	2	2	0.05	10	400	0.0006
Montseny	5	2	3	0.05	5	225	0.0015

4.4.1.1 Urban Data

The Athens small, Athens large, Chicago and Berlin are four widely used datasets for map construction [Ahmed 2015c]. The Athens small and Athens large datasets are collected by school buses from areas of Athens, Greece. The Chicago dataset covers an area in downtown Chicago, US, and includes trajectories from university buses. The Berlin, Germany, dataset are taxi trajectories. The statistics of the datasets are summarized in Table 4.3. Figure 4.9 visualizes each dataset. Note that, the dataset of Athens large given in [Pfoser 2016] has 120 trajectories as shown in Figure 4.10, which is not the one used in the literature. We use a bounding box (red frame) to cut the dataset and get the commonly used one that is shown in Figure 4.9(b).


(a) Athens small (b) Athens large

(c) Chicago (d) Berlin

Figure 4.9: Plots of urban GPS datasets.

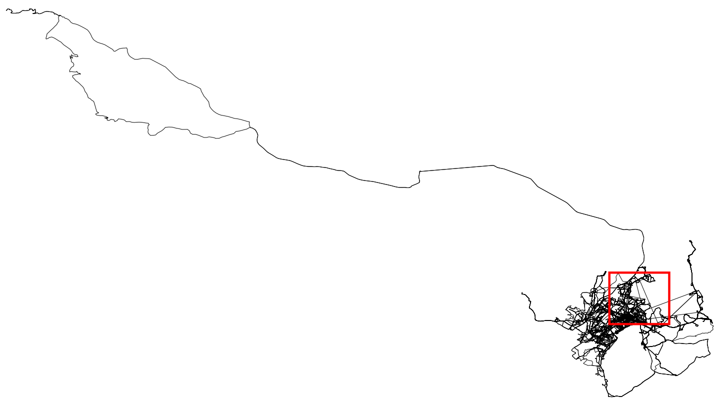


Figure 4.10: Plot of Athens large dataset using the data available online.

Table 4.3: Statistics of urban GPS datasets.

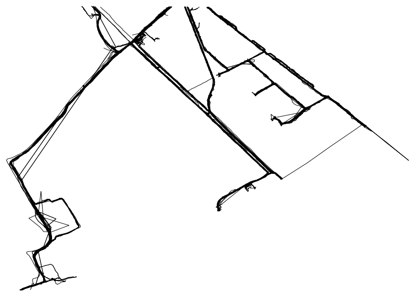
Dataset	Area	Trajectory Amount	Point Amount
Athens small	$2.6km \times 6km$	129	2839
Athens large	$12km \times 14km$	482	32745
Chicago	$3.8km \times 2.4km$	889	118360
Berlin	$6km \times 6km$	27189	192223

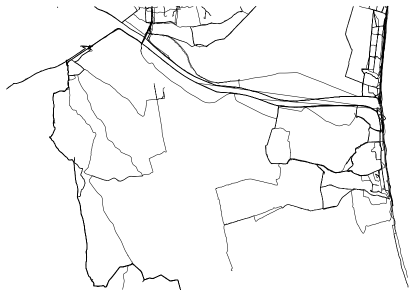
4.4.1.2 Hiking Data

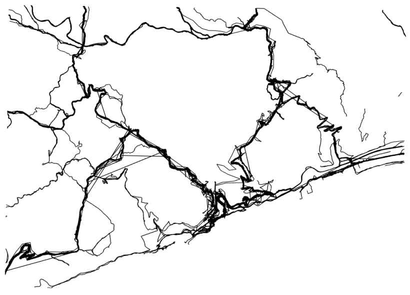
Four hiking datasets are denoted by Duran *et al.* [Duran 2020] and named as Delta, Aiguamolls, Garraf and Montseny, respectively. The statistics of the datasets are summarized in Table 4.4. Figure 4.11 visualizes each dataset.

Table 4.4: Statistics of hiking GPS trajectory datasets.

Dataset	Area	Trajectory Amount	Point Amount
Delta	$2.9km \times 2.8km$	161	38029
Aiguamolls	$9.6km \times 5.9km$	101	46116
Garraf	$6.7km \times 4.6km$	630	288472
Montseny	$7km \times 4.7km$	101	128181


(a) Delta


(b) Aiguamolls


(c) Garraf

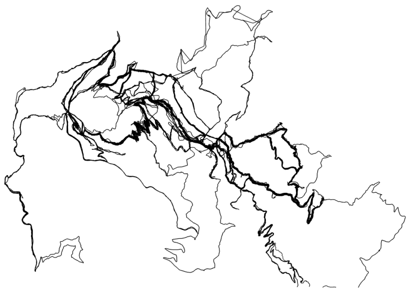

(d) Montseny

Figure 4.11: Plots of hiking GPS datasets.

4.4.2 Map Comparison by Visual Inspection

In this section, we visually compare the generated maps by different algorithms. We discuss two important factors, data coverage and artifacts existence. The data coverage checks if the generated map covers all the paths followed by the input dataset. We visually compare the data coverage of maps generated by different algorithms. The data coverage analyzes if the generated map covers all the paths followed by the input dataset. [Duran 2020] presents a detailed analysis of the artifacts in the generated maps. It shows that all of the algorithms for comparison produce artifacts in some specific areas. These artifacts especially appear when dealing with hiking data. This is because this kind of data is much more arbitrary and zigzagging than urban data. Hence, we check the artifacts in the generated maps.

4.4.2.1 Data Coverage

Figure 4.12 shows the maps generated from the Chicago dataset by different algorithms. We present the generated maps of the other datasets in Appendix B. Figure 4.12(d) and Figure 4.12(f) include obviously redundant edges around the same path, which results to large number of edges as listed in Table 4.5. Figure 4.12(b) has broken edges in the very right side. Compared with the other maps, Figures 4.12(c) - 4.12(e) have very low data coverage where only a few edges are produced. Figure 4.12(g) misses the left bottom and left up paths, which is also common in some other maps. Wang *et al.* (Figure 4.12(h)) generate a quite similar result to ours, but the map still does not cover all the paths, besides, the map complexity is very high. Our framework (Figure 4.12(i)) has the advantage of good coverage of all the paths and low map complexity.

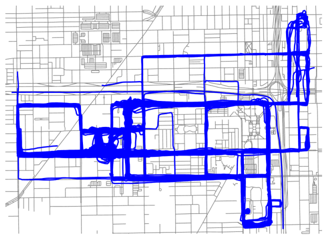
4.4.2.2 Artifacts Existence

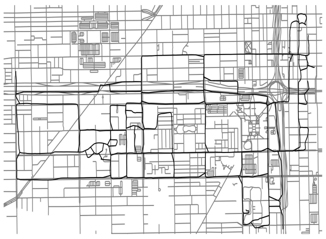
According to [Duran 2020], we analyze the 10 types of common artifacts (numbered by [$C1$] - [$C10$]) and 7 algorithmic-specific artifacts (numbered by [$S1$] - [$S7$]). In both cases, we present the maps generated by our framework and by Wang *et al.*. Besides, for each common artifact, we only take the best map in [Duran 2020]. For the algorithmic-specific artifacts, we show the examples in [Duran 2020]. Figure 4.13 to Figure 4.29 show artifacts on maps. In each figure, the first, second and third results are from [Duran 2020], by Wang *et al.*, and by our three-step framework, respectively.

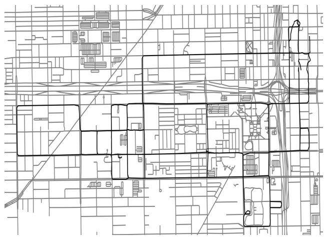
In Figure 4.13, artifact [$C1$] compares if the algorithm produces a y-shape map at narrow curves. Both Figure 4.13(b) and Figure 4.13(c) have this problem, while the map in Figure 4.13(c) is better concerning the similarity with the groundtruth.

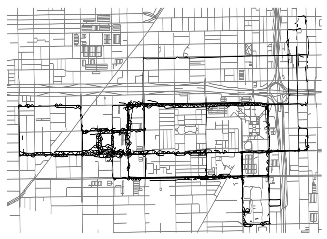
In Figure 4.14, artifact [$C2$] considers the shortcuts at intersections. Although our result (Figure 4.14(c)) does not approximate the bifurcation shape as well as Figure 4.14(a), the map has the bifurcation and is simpler and more schematic. Moreover, our map is better than that in Figure 4.14(b).

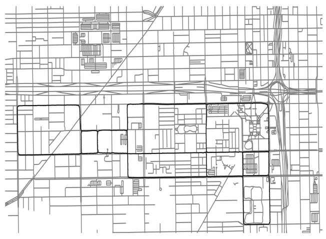
Artifact [$C3$], shown in Figure 4.15, checks the artificial bridges appearing between parallel close paths. Although the map in Figure 4.15(a) does not include any bridge, the algorithm wrongly merges the parallel paths. Contrarily, the map in Figure 4.15(b) shows two irregular paths joined with a nonexistent bridge. By comparison, our framework (Figure 4.15(c)) generates the two parallel paths accurately, while also includes two bridges.

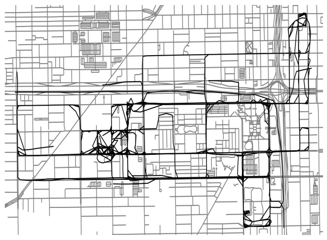

(a) Chicago data

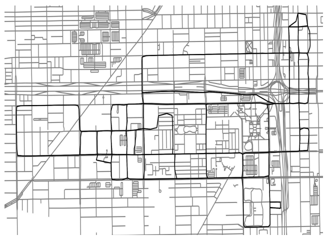

(b) By Ahmed and Wenk

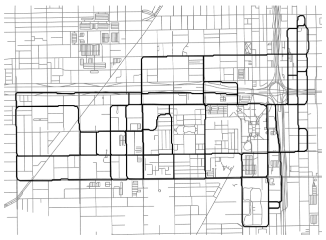

(c) By Biagioni and Eriksson


(d) By Cao and Krumm


(e) By Davies *et al.*


(f) By Edelkamp and Schrödl


(g) By Karagiorgou and Pfoser


(h) By Wang *et al.*

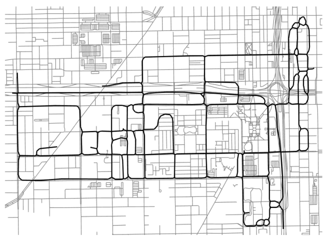
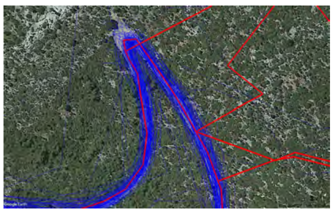
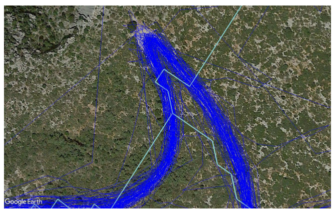

(i) By us

Figure 4.12: Plots of generated maps (black) on the groundtruth (gray) of Chicago datasets (blue) by different methods.


(a) By Ahmed and Wenk


(b) By Wang *et al.*

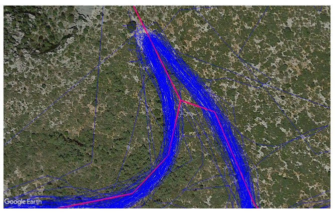
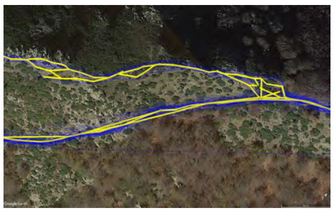
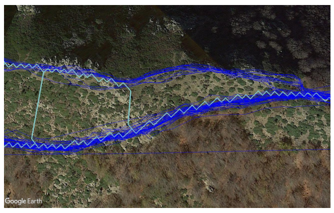

(c) By us

Figure 4.13: Plots of artifact $[C1]$ (merged narrow curves) in generated maps.


(a) By Cao and Krumm


(b) By wang *et al.*

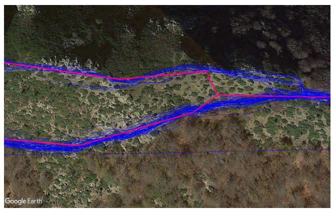
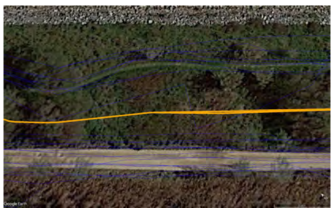
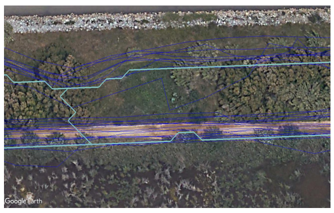

(c) By us

Figure 4.14: Plots of artifact [$C2$] (shortcuts at intersections) in generated maps.


(a) By Karagiorgou and Pfoser


(b) By Wang *et al.*

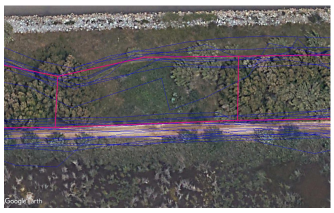
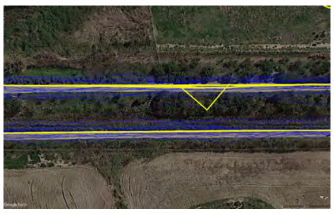
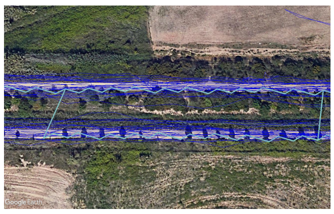

(c) By us

Figure 4.15: Plots of artifact [$C3$] (artificial bridges) in generated maps.

In Figure 4.16, artifact [$C4$] concerns the merging of parallel paths. By comparison, we generate the best map (Figure 4.16(c)). Figure 4.16(b) shows two zigzagging paths with several artificial bridges. Figure 4.16(a) identifies the two parallel paths but includes redundant edges in the upper path, and the map has a wrong triangle shape.


(a) By Cao and Krumm


(b) By Wang *et al.*

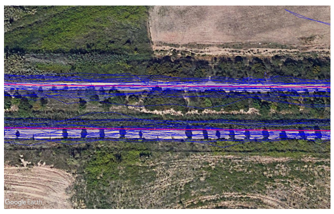

(c) By us

Figure 4.16: Plots of artifact [$C4$] (merged parallel paths) in generated maps.

Artifact [$C5$] evaluates if the algorithm generates duplications for one path. In Figure 4.17(c), all the algorithms do not have this problem. However, our framework is the only method that reports the perpendicular downwards path. This path has a low frequency as only one trajectory follows it.

In the area of Figure 4.18(c), the trajectories go back and forth on a single path. Artifact [$C6$] detects if the algorithm produces duplicated paths. None of the three methods has this problem. However, our framework generates the simplest map (Figure 4.18(c)), although the path is a little bit straightened. By contrast, the other two algorithms create redundant edges to represent the path.

In an area where multiple paths crossing each other, artifact [$C7$] (Figure 4.19)

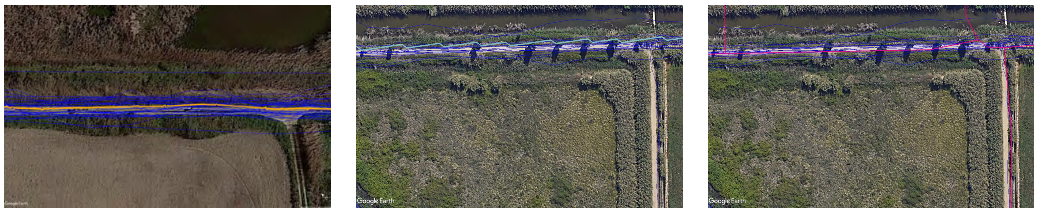

(a) By Karagiorgou and Pfoser (b) By Wang *et al.* (c) By us

Figure 4.17: Plots of artifact [$C5$] (duplicated paths) in generated maps.

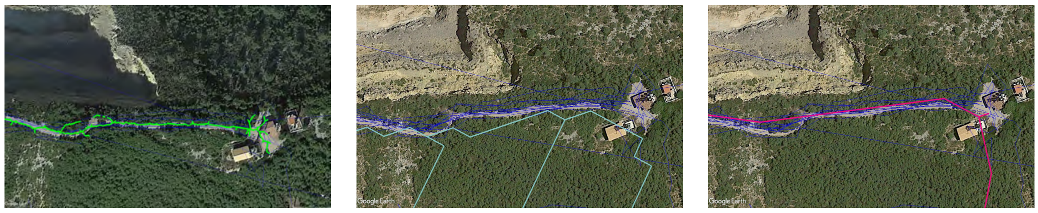

(a) By Davies *et al.* (b) By Wang *et al.* (c) By us

Figure 4.18: Plots of artifact [$C6$] (duplicated back-and-forth paths) in generated maps.

considers the excessive number of vertices and edges. By comparison, the map in Figure 4.19(a) has too many short paths that do not exist. Besides, it includes paths that are separated from the other paths and misses some infrequency paths that are followed by a few number of trajectories. Thus, this map is the worst. Compared with the ground truth, Wang *et al.* generates the best map. The map by our framework (Figure 4.19(c)) maintains the paths but includes some short nonexistent paths.

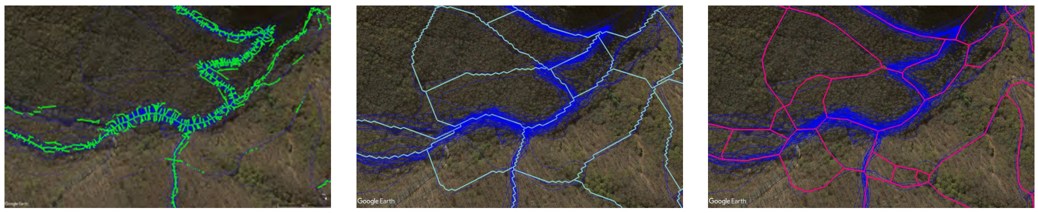

(a) By Davies *et al.* (b) By Wang *et al.* (c) By us

Figure 4.19: Plots of artifact [$C7$] (excessive number of connections in area) in generated maps.

In the case of artifact [$C8$], Figure 4.20 shows a simple path. Concerning generating a single path, all these three algorithms perform well. However, the map in Figure 4.20(a) has some zigzagging shape on the right side, and this issue happens along the whole path (Figure 4.20(b)) produced by Wang *et al.*. By comparison, our framework obtains the most schematic and accurate map 4.20(c).

In Figure 4.21, artifact [$C9$] focuses on the missing parts of the single path. All the maps cover the input data. However, the one in Figure 4.21(b) has artificial zigzagging

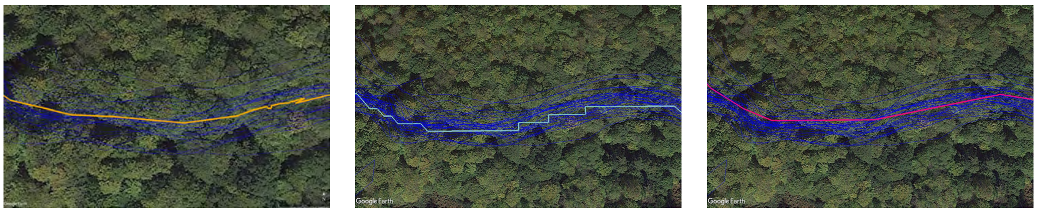

(a) By Karagiorgou and Pfoser (b) By Wang *et al.* (c) By us

Figure 4.20: Plots of artifact $[C8]$ (excessive number of connections along single path) in generated maps.

shapes, which greatly increases the map complexity.

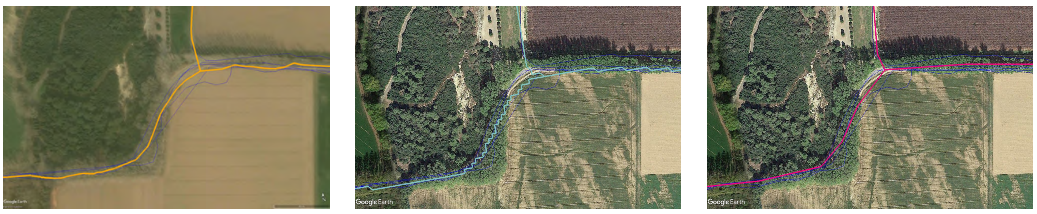

(a) By Karagiorgou and Pfoser (b) By Wang *et al.* (c) By us

Figure 4.21: Plots of artifact $[C9]$ (fragmented paths) in generated maps.

In Figure 4.22, artifact $[C10]$ detects the generated nonexistent paths. Due to the complicated trajectory data, none of the three algorithms works perfectly. The map in Figure 4.22(a) captures the main paths but includes too many wrong paths. Wang *et al.* merge the narrow curves and miss the path on the top left (Figure 4.22(b)). Figure 4.22(c) also merges the narrow curves but is slightly better. However, our map reports the main road and also some existing shortcuts present in the input data. Unfortunately, the central shortcut seems to be unreal.

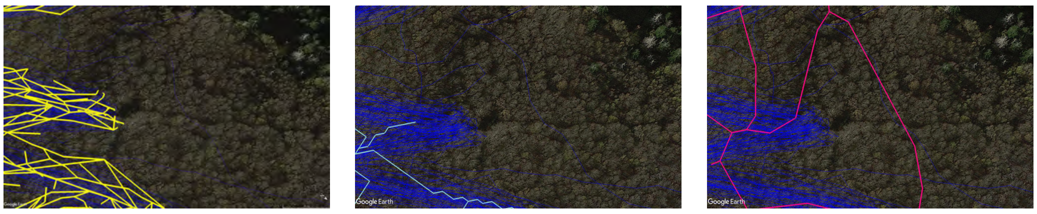
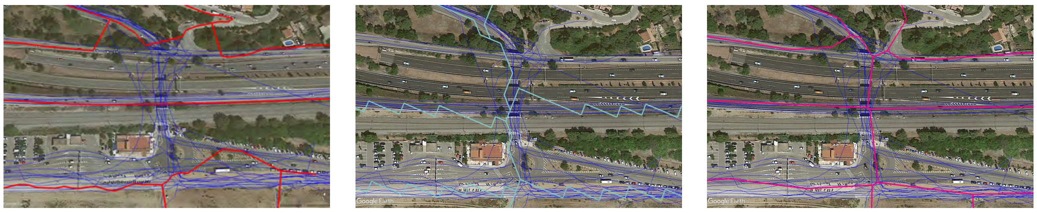

(a) By Cao and Krumm (b) By Wang *et al.* (c) By us

Figure 4.22: Plots of artifact $[C10]$ (nonexistent paths created) in generated maps.

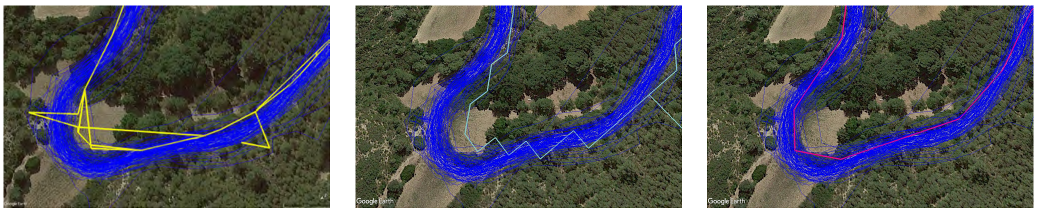
Artifact $[S1]$ (Figure 4.23) concerns if the algorithm misses a path. By comparison, our framework performs the best. Figure 4.23(a) ignores the vertical path in the center and a part of the top horizontal path. Figure 4.23(b) does include the whole top horizontal path and has zigzagging shapes along the paths.



(a) By Ahmed and Wenk (b) By Wang *et al.* (c) By us

Figure 4.23: Plots of artifact [$S1$] (missed set of trajectories) in generated maps.

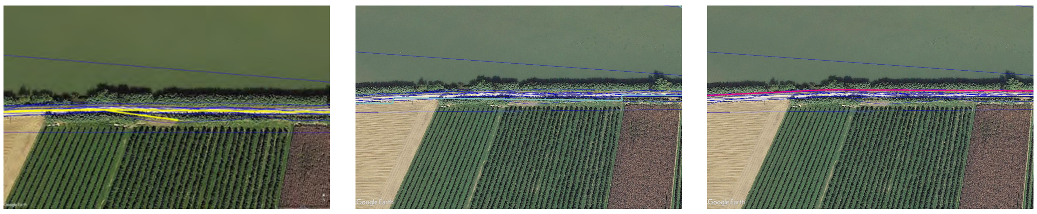
Artifact [$S2$] (Figure 4.24) considers a specific turning area. The map (Figure 4.24(a)) by Cao and Krum has redundant vertices and edges at the curvature, and the positions are inaccurate. In Figure 4.24(b), the position of the generated curvature is far from the groundtruth, and the map has the zigzagging shapes. By comparison, our framework gives the best result of a simple map and a well-kept curvature.



(a) By Cao and Krumm (b) By Wang *et al.* (c) By us

Figure 4.24: Plots of artifact [$S2$] (reduced curvature) in generated maps.

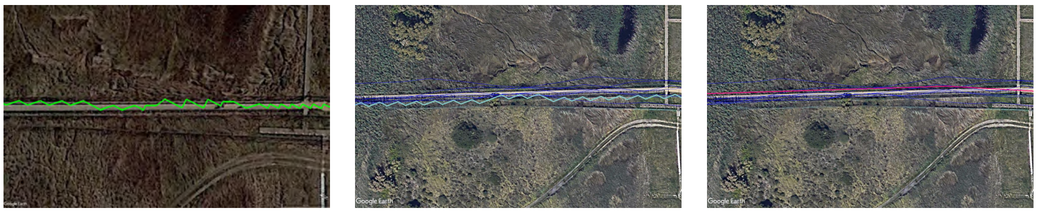
In artifact [$S3$] (Figure 4.25), a trajectory under the road is far from the other trajectories. In this case, Figure 4.25(a) includes a hair along the path. The algorithms by Wang *et al.* and our framework avoid this issue, however, the map in Figure 4.25(b) also has the zigzagging shapes.



(a) By Cao and Krumm (b) By Wang *et al.* (c) By us

Figure 4.25: Plots of artifact [$S3$] (hair in off-track trajectories) in generated maps.

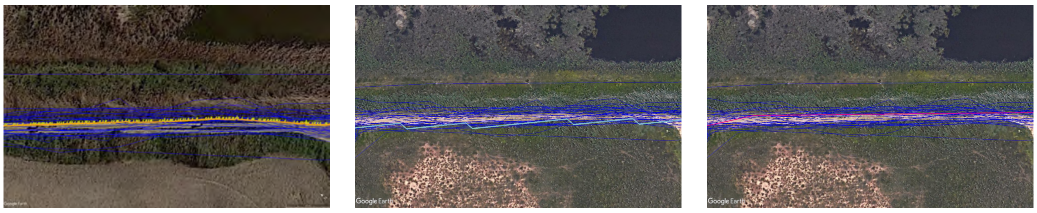
Artifact [$S4$] (Figure 4.26) checks the alias in generated maps. The alias also refers to a zigzagging shape. This is common in density-based algorithms because of the irregular density distribution. Both Figure 4.26(a) and Figure 4.26(b) have this problem. Our framework generates the most accurate and smooth map (Figure 4.26(c)).



(a) By Davies *et al.* (b) By Wang *et al.* (c) By us

Figure 4.26: Plots of artifact [*S*4] (aliased generated map) in generated maps.

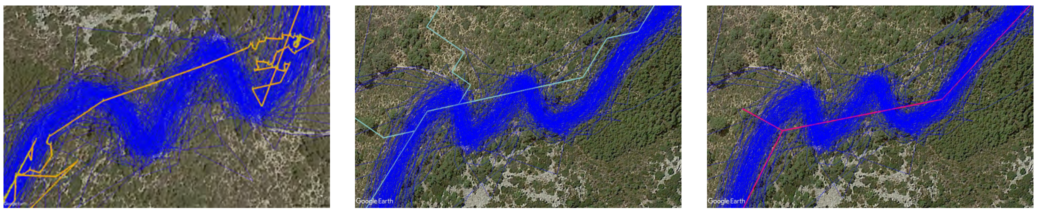
In artifact [*S*5] (Figure 4.27), the map by Cao and Krumm includes too many aligned vertices (Figure 4.27(a)). By comparison, Figure 4.27(b) and Figure 4.27(c) show better results, and our framework generates the best map concerning the similarity with the groundtruth.



(a) By Karagiorgou and Pfoser (b) By Wang *et al.* (c) By us

Figure 4.27: Plots of artifact [*S*5] (too many aligned vertices) in generated maps.

In artifact [*S*6] (Figure 4.28), due to the noise in data and complicated zigzagging paths, all the algorithms have the issue of over-simplifying. However, the maps in Figure 4.28(a) and Figure 4.28(b) are even worse considering either the redundant vertices and edges or the nonexistent paths.



(a) By Karagiorgou and Pfoser (b) By Wang *et al.* (c) By us

Figure 4.28: Plots of artifact [*S*6] (winding path simplified) in generated maps.

Artifact [*S*7] (Figure 4.29) checks if the algorithm ignores some paths. By contrast, Figure 4.29(a) shows the worst result as the map only maintains a small part. Wang *et al.* give a better map but still miss the paths on the right side. Our framework produces the best map that includes all the paths followed by trajectories.

The comparison of artifacts gives a good general idea of the behavior of our frame-

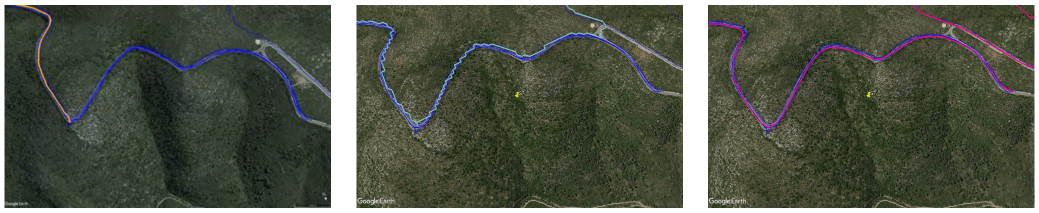

(a) By Karagiorgou and Pfoser (b) By Wang *et al.* (c) By us

Figure 4.29: Plots of artifact [$S7$] in generated maps (missing trajectories end).

work. Overall, we can say that our algorithm behaves really well in the simple and most of the complicated scenarios.

4.4.3 Map Comparison by Quantitative Evaluation

In this section, we first present the map complexity according to [Ahmed 2015c], then give the comparison by two distance measures.

4.4.3.1 Map Complexity

The map complexity refers to the number of vertices and edges in map, and the total length of edges. The length of an edge is the Euclidean distance between the two vertices defining the edge. Table 4.5 and Table 4.6 show the map complexities by different algorithms of urban and hiking datasets, respectively.

Concerning the number of vertices and edges, our framework generates less complex graphs than most algorithms, while Wang *et al.* and Davies produce the most complex maps for urban and hiking data, respectively. In comparison to map length, the result varies a lot by different algorithms, especially with hiking datasets. The reason of the differences is based on how the algorithms proceed.

In the intersection-linking-based [Karagiorgou 2012] and point-clustering-based [Edelkamp 2003] algorithms, the first step identifies intersections or vertices from all the points, then the second step connects them by associating with each trajectory. These methods always determine redundant vertices [Duran 2020]. The incremental methods [Ahmed 2012, Cao 2009] merge a trajectory to an original empty map incrementally. If a part of a trajectory is different from the existed map, the algorithms add this part as a new edge. Due to the noise in data, the trajectories following the same road are not the same, so the algorithms create many similar edges. The noise in data also affects the density-based algorithms [Davies 2006, Wang 2015], the algorithms produce zigzagging shapes that increase the map length. As described in Section 4.4.2.2, the examples of artifacts also show the problem of redundant vertices and edges.

4.4.3.2 Two Distance Measures

Two distance measures presented in [Ahmed 2015c], the path-based and directed Hausdorff, are used to compare maps. These measures are used the compute the distance

Table 4.5: Map complexity of urban data.

Methods	Vertex Amount	Edge Amount	Length (km)
Athens small			
Ahmed	344	378	35
Biagioni	391	398	22
Cao	20	14	3
Davies	209	227	2
Edelkamp	526	1037	197
Karagiorgou	660	637	35
Wang	7104	8260	44
Our	451	509	40
Athens large			
Ahmed	7067	7960	1358
Karagiorgou	6584	5280	252
Wang	56456	80377	430
Our	4571	5333	430
Chicago			
Ahmed	1195	1286	34
Biagioni	303	322	24
Cao	2092	2948	78
Davies	1277	1310	14
Edelkamp	828	1247	83
Karagiorgou	596	558	26
Wang	3234	3549	36
Our	293	338	35
Berlin			
Ahmed	1322	1567	164
Karagiorgou	2542	2262	161
Wang	20747	25711	277
Our	1650	1867	160

Table 4.6: Map complexity of hiking data.

Methods	Vertex Amount	Edge Amount	Length (km)
Delta			
Ahmed	2362	2459	395
Cao	2667	2436	1810
Davies	10229	10197	45
Edelkamp	1028	1756	11029
Karagiorgou	6787	4817	446
Wang	4994	5467	19
Our	457	508	20
Aiguamolls			
Ahmed	13454	13516	2179
Cao	10621	5308	2208
Davies	39786	39206	121
Edelkamp	4147	4918	40849
Karagiorgou	21690	21810	1990
Wang	22228	24258	131
Our	1589	1719	115
Garraf			
Ahmed	7827	7898	2005
Cao	13565	8172	4345
Davies	88009	87162	363
Edelkamp	5295	9320	30763
Karagiorgou	36487	36574	2229
Wang	7922	8720	95
Our	1415	1496	69
Montseny			
Ahmed	8893	8940	1721
Cao	19323	11625	2809
Davies	83025	81783	214
Edelkamp	4610	7774	9661
Karagiorgou	24329	24492	4478
Wang	17519	19143	103
Our	1754	1877	86

from the groundtruth and the generated map. Since the groundthruth is only available of urban data, we only consider the urban datasets in this comparison. The distance measures for our map are computed by using the available implementation of [Pfoser 2016]. Table 4.7 and Table 4.8 list the results. A small distance value indicates that the generated map is similar with the groundtruth, and hence good performance. The maps of Athens large, Chicago and Berlin generated by Wang *et al.* have too many edges, and the path-based distance cannot give a result after one week. According to the statement in [Ahmed 2015c], the algorithms by Karagiorgou and Pfoser [Karagiorgou 2012] and by Biagioni and Eriksson [Biagioni 2012b] perform better than the others. By comparison of the results in Table 4.7, the result of our framework does not seem to be as good as these two algorithms but is better than the other methods. On the other hand, according to Table 4.8 our algorithm behaves better with the directed Hausdorff distance measure.

For both the path-based and the discrete Hausdorff distance, the reason why our framework obtains these not very good evaluations is twofold. First, the number of vertices used to define the map influences these two distance measures (the Fréchet distance is used to compare paths in the path-based distance). As shown in Figure 4.30,

Table 4.7: Comparison of path-based distance measure of urban data

Methods	**Path-based Distance(m)**							
	min	**max**	**median**	**average**	**2%**	**5%**	**10%**	**15%**
Athens small								
Ahmed	9	224	45	52	101	101	81	72
Biagioni	5	73	35	36	67	66	61	57
Cao	The map is too small to perform this measure.							
Davies	4	38	11	11	38	18	14	14
Edelkamp	2	229	36	39	89	72	68	61
Karagiorgou	7	229	32	38	113	68	59	57
Wang	8	304	117	127	247	229	222	208
Our	5	229	36	42	95	75	73	65
Athens large								
Ahmed	7	849	70	85	250	164	132	114
Karagiorgou	2	175	25	32	109	80	63	53
Wang	The map is too large to perform this measure.							
Our	2	465	35	40	100	82	69	61
Chicago								
Ahmed	7	201	35	42	127	100	85	76
Biagioni	3	71	15	18	71	38	27	26
Cao	1	126	24	27	79	61	49	42
Davies	2	92	12	14	57	24	22	21
Edelkamp	1	205	29	37	99	84	72	66
Karagiorgou	3	89	15	23	72	72	65	51
Wang	The map is too large to perform this measure.							
Our	4	109	24	32	100	81	70	64
Berlin								
Ahmed	9	540	66	74	207	147	120	107
Karagiorgou	4	306	28	37	120	85	65	52
Wang	The map is too large to perform this measure.							
Our	5	395	32	39	104	81	67	60

it contains two simple trajectories, the black trajectory P and the blue trajectory Q. Q draws a straight line that is defined by 2 points in Figure 4.30(a) and 3 points in Figure 4.30(b). The Fréchet and Hausdorff distances between Q and P are given by the length of the dotted-green line segments. Hence, using three points to represent the straight line drawn by Q benefits these distance measures. Indeed, having a more schematic representation (Figure 4.30(a)) is better in complexity. However, using redundant points (Figure 4.30(b)) is better for any of these distance measures (it wouldn't happen if these distances were computed in their continuous version). Our method tries to obtain the simplest map representing the original data as schematically as possible, so we remove the expendable points. However, this is penalized by the distance measures. If we compute the path-based and Hausdorff distances before the Douglas-Peucker simplification, our map is better evaluated. Second, these two measures evaluate the similarity between the generated map and the groundtruth without taking into account whether they cover the same area or not. Smaller maps containing fewer paths will easily be better evaluated. Indeed identifying paths that exist in the input data but do not appear in the groundtruth would completely degrade the result. Since our algorithm tries to keep all the paths followed by trajectory data including the low-frequency trajectories, some of these paths are kept, but the other algorithms regard it as unimportant and eliminate it.

Table 4.8: Comparison of directed Hausdorff distance measure of urban data.

Methods	Directed Hausdorff Distance(m)							
	min	max	median	average	2%	5%	10%	15%
Athens small								
Ahmed	1	82	25	26	82	54	46	40
Biagioni	3	74	19	20	47	43	31	31
Cao	5	25	13	13	25	25	25	22
Davies	2	13	7	6	13	13	13	11
Edelkamp	1	86	18	21	63	50	42	37
Karagiorgou	2	84	14	17	54	40	33	30
Wang	1	106	9	12	56	38	26	21
Our	1	79	14	18	54	44	33	29
Athens large								
Ahmed	1	269	30	33	84	67	56	50
Karagiorgou	1	200	10	13	46	35	26	22
Wang	1	1143	7	12	47	28	20	17
Our	1	130	15	18	51	41	33	29
Chicago								
Ahmed	1	81	14	19	72	59	43	35
Biagioni	2	53	9	11	29	25	23	17
Cao	1	78	9	12	44	35	28	25
Davies	2	20	8	7	20	14	13	12
Edelkamp	1	93	8	13	57	48	35	25
Karagiorgou	1	48	7	8	41	23	15	13
Wang	1	111	6	10	63	38	17	13
Our	1	73	9	13	50	38	25	20
Berlin								
Ahmed	1	219	30	33	95	70	60	53
Karagiorgou	1	232	14	18	59	42	34	30
Wang	1	334	14	25	141	111	61	32
Our	1	162	14	18	51	42	34	29

4.4.4 Runtime of Our Framework

In this section, to see how much the parallelization speeds up computation, we run our framework both in parallel and with one core. Table 4.9 and Table 4.10 present the runtime of urban and hiking datasets, respectively. In the case of urban data, running in parallel improves the speed by 7 times on average, and 5 times in hiking data.

Table 4.9: Runtime (second) of urban datasets.

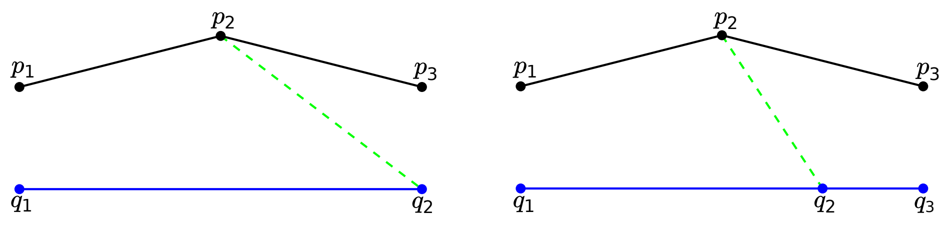
Datasets	Runtime (with one core)	Runtime (in parallel)
Athens small	26.81	4.22
Athens large	595.28	115.70
Chicago	119.13	10.32
Berlin	1253.13	110.01
Average	498.59	60.06

Table 4.10: Runtime (second) of hiking datasets.

Datasets	Runtime (with one core)	Runtime (in parallel)
Delta	19.67	6.29
Aiguamolls	38.34	18.17
Garraf	259.67	30.43
Montseny	168.69	29.52
Average	121.59	21.10

4.4.5 Summary

To sum up, concerning the data coverage and artifacts, our framework performs the best. According to the examples in artifacts, there are two weak points of our framework. First, in the case of narrow curves, if the trajectories are noisy, the algorithm tends



(a) The blue trajectory has 2 points. (b) The blue trajectory has 3 points.

Figure 4.30: Distances between two trajectories with different number of points. The blue trajectory has (a) 2 and (b) 3 points.

to merge the paths and produces y-shape. The examples are in Figure 4.13(c), Figure 4.14(c), Figure 4.19(c), Figure 4.22(c), Figure 4.23(c) and Figure 4.28(c). Second, our framework generates some artificial bridges. The examples are in Figure 4.15(c), Figure 4.17(c) Figure 4.18(c) and Figure 4.19(c). The reason for both issues is that the noise degrades the accuracy of the density surface. In terms of the map complexity, our framework generates the simplest maps for most datasets. The other algorithms produce redundant vertices and edges either by wrong identification of vertices or by zigzagging shapes. Considering the evaluation based on two distance measures, our result is competitive. Besides, as explained in Section 4.4.3.2, keeping infrequent paths followed by a few trajectories in the map degrades the evaluation but is meaningful. Also, as shown in the last section, as a parallelizable framework, the performing of map construction is really fast.

4.5 Conclusions

In this work, we have presented a three-step framework that is fast, robust and parallelizable to extract a route graph from GPS trajectories. The framework combines a density map smoothed by a Gaussian filter to reduce noise; the Slide tool to adjust trajectories to the higher density zones making the density surface more compact; and a thinning algorithm followed by a Douglas-Peucker simplification algorithm to construct the route graph that records the transited zones. Besides, we have proposed two solutions to solve some defects of Slide, it has been demonstrated to be effective. Overall, we obtain a good schematic representation of the initial data with almost no redundant edges, reporting both frequent and infrequent paths.

For the large-scale GPS data, the framework would face the limitation of computer performance and computing speed. Using the weight of edges to visualize the generated map and to remove possible wrong routes are also helpful to improve the quality of the generated maps. In the next chapter, new proposals will face these issues.

CHAPTER 5

Improvements on the Three-Step Map Construction Framework

Contents

5.1 Introduction

Based on our three-step framework for generating maps from GPS data (see Chapter 4 for more details), we propose to improve its practicality and performance in two aspects. The first is utilizing a split-and-merge strategy to deal with the challenges when processing a large amount of data. The second is taking advantage of the knowledge of how many trajectories are represented by an edge to visualize the map and improve the map quality.

Many map construction approaches have been proposed in the literature. Undoubtedly, most algorithms need extensive experiments to produce high-quality maps, while is time-consuming. For example, as claimed by Biagioni and Eriksson in a survey [Biagioni 2012a], for a small dataset with 899 traces, the Cao algorithm [Cao 2009] takes 2.5 days. In another survey [Ahmed 2015c] by Ahmed *et al.*, for a large Berlin dataset which occupies $6km \times 6km$ but has very dense trajectories, the runtimes of different algorithms range from 2 hours to 4 days. With the widespread GPS devices, a large amount of trajectory data from a big area are daily collected. This brings two challenges to perform a map construction algorithm. The first challenge is to handle a large number of trajectories with an acceptable computational cost. Second, performing an algorithm needs to set memories for different variables. For example, for the density-based algorithm, the density distribution is an important element to be computed and stored. If the geographical area is very large and not sparse which requires a big matrix to save the information, the memory capacity of computes will be a key factor to run the algorithm. To overcome the limitations of storage and computational cost, we propose a split-and-merge strategy to improve the map construction algorithm. Firstly, the geographical area is divided into small regions with less amount of trajectories, then the map construction algorithm is performed on each region individually, Finally, the graphs from different regions are merged to generate a whole map.

The map generated from a set of trajectories helps to understand the behavior of moving objects and update the existing maps, while it is always not the same with the groundtruth since the quantity of data is limited and the trajectories have noise. Besides, the generated map usually has the wrong edges because of the mechanism of algorithms or noise in data. Some algorithms simply remove the edges if the density distribution of data is low [Wang 2015]. In this chapter, we propose to compute the edge weight to visualize the generated map, which gives insights to the popular and rare routes, and to remove possible wrong edges. Generally, the generated wrong edges are usually short and have few numbers of trajectories passing through, for instance, the artificial bridges presented in Section 4.4.2.2. Thus, we take into account two factors, length and frequency. The length of an edge is the Euclidean distance between the two vertices of this edge. The frequency of an edge is the number of trajectories crossing through this edge. Intuitively, to obtain the frequencies of edges, each trajectory is required to be mapped into the road graph. Directly finding the closest edge for each line segment of a trajectory based on a distance measure is not efficient because of two reasons. First, computing the distance accurately between line segment and edge asks for a good distance measure, which is still an open challenge for similarity measurement as explained in Section 2.2.2. However, measuring the distance between two points is easier. Second, the trajectory data may be noisy due to local shape change from inaccurate GPS receiver or signal errors, which would cause errors in finding corresponding edges. For example, Figure 5.1 shows a simple graph with 2 parallel roads including 9 vertices from v_1 to v_9 ($v_1, v_2, \ldots, v_9 \in V$) and 7 edges from e_1 to e_7 ($e_1, e_2, \ldots, e_7 \in E$), and a trajectory visualized in blue. Clearly, the trajectory has the second part in gray area close to e_2, and the final mapping would be $\left(e_4, e_2, e_6, e_7\right)$ with respect to the spatial similarity. According to the internal homogeneity, this trajectory should follow the road below, thus adding weight to e_2 instead of e_3 is not reasonable. Benefiting from the Slide tool, we make use of the adjusted trajectories to find the mapping between trajectories and edges.

The outline of this chapter is organized as follows. Section 5.2 reviews the map construction algorithms and summarizes the discussion on runtime. Section 5.3 describes the proposed split-and-merge strategy in detail. Section 5.4 explains the proposal of computing the edge weight. Section 5.5 presents experiments and comparisons. Section 5.6 discusses the conclusions and future work.

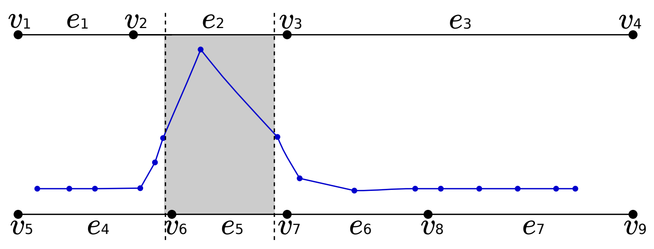


Figure 5.1: Example of local noise in trajectory influences the mapping result.

5.2 Related Work

Map construction focuses on generating a route map from a set of GPS data. Many approaches have been proposed and discussed [Biagioni 2012a, Ahmed 2015c], and several of them have been presented in Chapter 4. The algorithms are mainly classified into three categories: incremental track insertion, intersection linking, and point clustering. However, most of them do not consider the size of data which influences the computing time or visualize the generated map by giving insights into road popularity. Table 5.1 gives an overview of several approaches and their description of data, runtime, and the visualization. It is clear that only a few algorithms include the runtime discussion and all of them plot the generated map only to show the roads (we call it plain visualization).

5.3 The Split-and-Merge Strategy

In this section, we describe the proposed split-and-merge strategy in detail. At the beginning, proceeding similarly to Section 4.3.1 in Chapter 4, the GPS data go through a pre-processing to remove the obvious noise. First, the geographical area is divided into different regions with small overlapping zones (Section 5.3.1). Then, the three-step framework is applied to each region to generate the respective route graphs. Finally, the generated maps are merged to produce a map for the whole area (Section 5.3.2). The edges and vertices of the overlapping areas are fixed so that they summarize the information obtained in the overlapped area of the different generated maps.

5.3.1 Geographical Area Splitting

The first step is to evenly split the geographical area into small regions. To keep the consistency of the graph at the splitting boundaries, we enlarge the adjacent regions by an overlapping zone. Figure 5.2 shows a simple example where the geographical area is split into two regions (region 1 and region 2) by a vertical splitting line, the midline (black dotted). To have an overlapping zone, the splitting boundary (blue dotted line) of the region 1 is placed at a certain distance to the right of the splitting line, and the splitting boundary (red dotted line) of region 2 is located at the same distance to the left of the splitting line. The overlapping zone (gray shadow) is the area between two

Table 5.1: Overview of map construction algorithms that include description of data and runtime.

Reference	Dataset			Give	Result
	Name	**Area Size**	**Track Amount**	**Runtime**	**Display**
[Edelkamp 2003]	Palo Alto	66 roads	unknown	no	plain
[Schroedl 2004]	Palo Alto	66 roads	unknown	no	plain
[Bruntrup 2005]	no	$3075km$ of roads	107	no	plain
[Cao 2009]	no	unknown	unknown	no	plain
[Niehöfer 2009]	no	unknown	4	no	plain
[Chen 2010]	Moscow	unknown	5000	no	plain
[Fathi 2010]	Microsoft Shuttles King County Paratransit Vehicles	unknown	unknown	no	plain
[Agamennoni 2011]	Western Australia	$3.5km \times 10.5km$	400	yes	plain
[Karagiorgou 2012]	Athens large	$12km \times 14km$	511	no	plain
[Li 2012]	no	$39°12'14" - 42°8'52"N$ $112°38'55" - 117°3'30"E$	893493	no	plain
[Ahmed 2012]	Berlin Taxi	unknown	3237	no	plain
[Biagioni 2012a]	Chicago	$3.4km \times 2.6km$	889	yes	plain
[Biagioni 2012b]	Chicago	$3.4km \times 2.6km$	889	no	plain
[Karagiorgou 2013]	Berlin Vienna Athens	unknown	unknown	no	plain
[Wu 2013]	Shenyang	unknown	2827	no	plain
[Liu 2012b]	Shanghai Taxi	$14.5km \times 14km$	unknown	no	plain
[Ahmed 2015d]	Athens large Athens small Berlin Chicago	$12km \times 14km$ $2.6km \times 6km$ $6km \times 6km$ $3.8km \times 2.4km$	511 129 26831 889	little	plain
[Li 2016]	Chicago Porto Taxi	$3.8km \times 2.4km$ 8.612244-8.588826 longitude, 41.13711-41.158098 latitude	889 1000	no	plain
[Tang 2017]	Qingshan Taxi Zhuankou Taxi	unknown	9089 3765	no	plain
[Li 2017]	Chicago Chengdu Taxi	$3.8km \times 2.4km$ $3.4km \times 2.6km$	889 2371	no	plain
[Stanojevic 2018]	Doha UIC	$6km \times 8km$ $2km \times 3km$	unknown	yes	plain
[Zheng 2018]	Wuhan Taxi	unknown	252677	no	plain
[Huang 2018]	Chicago Athens small Berlin Wuhan Taxi	$3.8km \times 2.4km$ $2.6km \times 6km$ $6km \times 6km$ $3km \times 3km$	889 129 26831 4885	no	plain

splitting boundaries. The width of the overlapping zone depends on τ, the cell size of the grid used to compute the density surface. According to our experimentation, this zone should occupy 30 grid cells, i.e. 30 pixels. Consequently, the splitting boundaries are obtained by translating the splitting line 15τ meters in the respective direction. The example splits the original region by a single splitting line. If more than one line is used, we should proceed similarly to each of them.

The GPS trajectories are then split according to the splitting boundaries. Each trajectory crossing the splitting boundary is cut into sub-trajectories. The intersection points of the trajectory and the splitting boundary, together with the trajectory endpoints, are the sub-trajectories endpoints. These sub-trajectories maintain the shape of the original trajectory, and each of them is entirely contained in a region of the split original area.

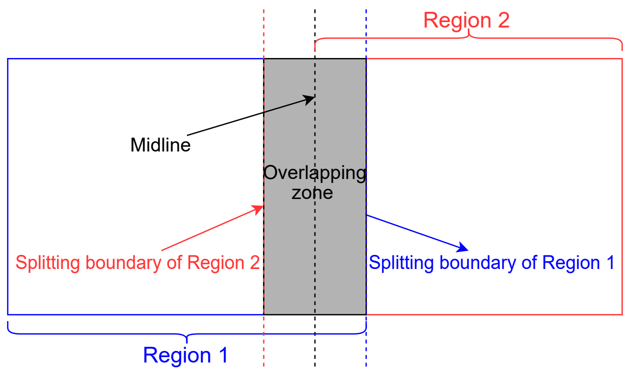

Region 2
Midline
Overlapping zone
Splitting boundary of Region 2
Splitting boundary of Region 1
Region 1

Figure 5.2: Illustration of splitting an geographical area into two regions.

Next, for each region, we apply the three-step framework of map construction defined in Chapter 4 to obtain a map. The defined sub-trajectories are considered as independent trajectories. After obtaining the map of each region, producing the whole final map requires merging the individual maps as is explained in the next section.

5.3.2 Boundaries Fixing and Maps Merging

In the merging step, we cut off the obtained maps along the splitting line (midline in Figure 5.2, red dashed line in Figure 5.3) and add new vertices on this splitting line to join the edges of the original map that intersect the splitting line.

Note that in the way we split the original data, we have doubly processed all the sub-trajectories contained in the overlapping zone. However, these sub-trajectories are not adjusted in the same way when dealing with each region. This is because when cutting the original trajectories, we have added endpoints and as explained in Section 4.3.3, Slide works differently on interior points and endpoints. As a result, trajectories are not adjusted equally, then the re-computed density distribution changes slightly. This directly influences the skeleton image. Thus, simply cutting the graphs along the splitting line, adding vertices at the end of the edges and combining the graphs according to the new vertices is insufficient. This does not ensure the connectivity and consistency of the global map in the splitting line. Next, we explain how to fix the map in this splitting line, and we call this *fix boundaries* for simplicity.

To fix the boundaries, we use the maps in the overlapping zones. Take Figure 5.2 as an example, we take the map of region 1 in the left 15τ-meter-width area and the map of region 2 in the right 15τ-meter-width area of the overlapping zone, then combine them as the initial map of the overlapping zone. Figure 5.3 gives an example of fixing boundaries using the Athens small dataset (please refer to [Ahmed 2015c] for more details). The map in Figure 5.3(a) is generated by our framework without the split-and-merge strategy. By comparison, the edges from different regions in Figure 5.3(b) are not correctly connected but are very close. Thus, we compute the intersections between the splitting line and the maps, then merge close-enough intersections to define new vertices. If the intersections are at an Euclidean distance smaller than a threshold, λ_{dis}, we use a new vertex located at the average position to represent these points. Finally, the edges are updated accordingly by replacing the intersection points with the corresponding new vertices. The fixed map in Figure 5.3(c) is almost the same with that in Figure 5.3(a).

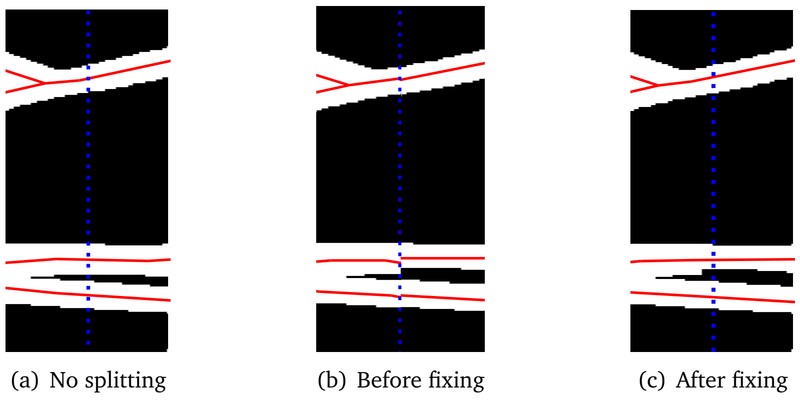

(a) No splitting (b) Before fixing (c) After fixing

Figure 5.3: The graph (in red) in the binary image of the overlapping zone. The midline is in blue.

Since we use the overlapping zone, the change of the density surface in the overlapping zone is very slight (see Figure 5.3). According to this, we set λ_{dis} to be related to σ. By extensive experiments, we found that $\lambda_{dis} = 3\tau\sigma$ achieves good results.

5.4 Edge Weight

The edge weight can be used in the visualization of a route graph, which gives insights into the popular and rare zones. Besides, utilizing the edge weight helps to remove possible wrong edges that are usually short and have few numbers of trajectories passing through. We define the weight of an edge as a pair of values based on two factors: frequency and length. The frequency of an edge is the number of trajectories passing through it. We compute it by counting the number of trajectories mapped to this edge. The length of an edge is the Euclidean distance between its two vertices. In the following, we explain how to do the mapping first.

The adjusted trajectories by Slide are mapped to edges for further computing the edge frequency. When recomputing the density surface, the adjusted trajectories are mapped to the grid where each cell corresponds to a pixel in the skeleton image. We take advantage of this information to find the closest pixels from the skeleton image for each trajectory point. Such a pixel corresponds to a map edge that is initially associated with the trajectory. Around the map vertices with degree bigger than two, wrong mappings usually happen. Figure 5.4 shows an example of mapping a trajectory (in blue) from the Athens small dataset. In Figure 5.4(b), the trajectory is initially mapped to the red edges. Note that this mapping is not correct, the top-right red edge should not appear. It happens because some points of the trajectory near the intersection are closer to this edge. To solve this problem, we remove the edges in which the trajectory occupies less than half of the pixels defining the edge. It is done by extracting information from the skeleton image. Hence, after removing these underused edges, the whole trajectory is mapped to a sequence of edges. Figure 5.4(c) shows that the final result of the mapping is accurate. The distance between pixels is measured by Euclidean distance. To speed the computation, the Euclidean distance transform [Maurer 2003] is applied to the skeleton image. For each pixel, the distance transform assigns the nearest nonzero pixel.

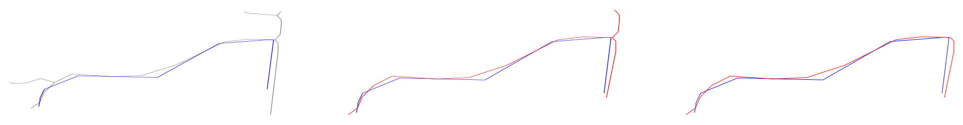

(a) Example trajectory (b) First mapping of edges (c) Final result with pruning

Figure 5.4: Plots of mapping a trajectory (in blue) to the route graph.

The advantage of this method is three-fold. First, the inputs (BW, $codeT$ and links) used for mapping have been produced along with the map construction, which saves time from data preparation. Second, the adjusted trajectory by Slide is used instead of the original one, which degrades the influence of noise and makes the mapping more accurate. Third, the similarity between the trajectory and the graph is computed by the distance between pixels which is more simple than measuring between line segments, and avoids the difficulty of selecting a distance measure.

5.5 Results and Discussion

This section presents the experimental results. We apply the same datasets used in Chapter 4 to carry out the experiments. First, we check the performance of the split-and-merge strategy. Then, we utilize the edge weight to visualize and filter the map. Finally, we give a summary of the experiments.

All the experiments are conducted by MATLAB 2018a software running on a Debian GNU/Linux machine with AMD Ryzen Threadripper 1950X 16-Core Processor and 32 GB RAM.

5.5.1 Performance of the Split-and-Merge Strategy

In this section, we compare the similarity between the map generated by the three-step framework and by adding the split-and-merge strategy. First, we do a visual inspection on the generated maps. Second, we give the quantitative evaluation, including map complexity, runtime, and the directed Hausdorff distance.

5.5.1.1 Visual Inspection on the Generated Maps

To perform the split-and-merge framework, we split all the datasets split into two regions like Figure 5.2 (see Appendix C.1 for more details). Although splitting an area into more regions would save time for the map construction of each region, it would increase the cost of the merging step and induce errors in the overlapping zone. Hereafter, we use "single" to indicate performing the three-step framework on the whole dataset directly, and "split-and-merge" to mean performing the map construction with split-and-merge strategy by splitting the dataset into two small regions. Figure 5.5 shows the generated maps of the Athens large and Garraf datasets (see Appendix C.2 for the other datasets). The maps by single and split-and-merge are red and blue, respectively. Besides, a zoom in view of the overlapping zone with the midline in red is shown on the right side.

By visual inspection, the generated maps by splitting and no splitting are very similar as they cover each other in most parts. Comparing the maps in the overlapping zone, we can see that the edges have difference in the Garraf dataset (Figure 5.5(b)). The reason is that in the overlapping zone, the trajectories are complicated, resulting in a complex map. In this case, when generating the maps for each region, the slight change of the density surface makes a greater difference on the edges.

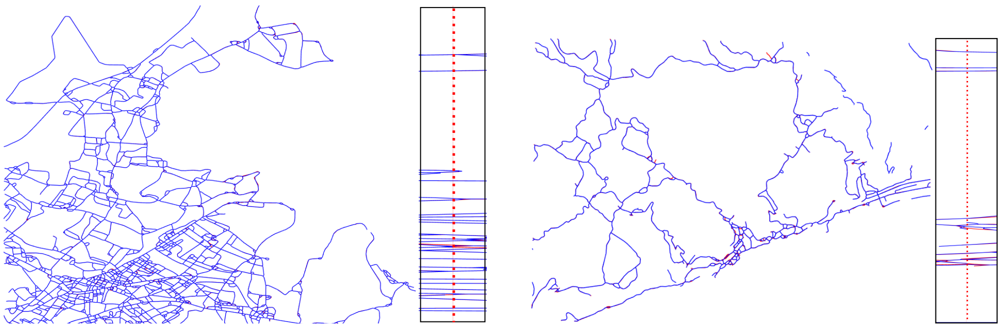

(a) Athens large (b) Montseny

Figure 5.5: Plots of generated maps of the Athens large and Garraf datasets.

5.5.1.2 Quantitative Evaluation

In this section, we evaluate the similarity of the maps generated by single and split-and-merge based on quantitative measures.

Table 5.2 gives the comparison on map complexity and runtime. Due to the splitting, the number of vertices and edges increases but the map length does not change a lot. In general, the split-and-merge strategy saves time. Note that the running times of Athens small, Chicago and Berlin dataset slightly increases. There are two reasons for this raise in time. First, for simple datasets like Athens small, generating the maps without splitting is fast enough, and splitting and merging will take additional time and make changes in the map. Second, fixing the boundaries takes more time if the graph in the overlapping zone is more complicated. Since the trajectories in Berlin dataset are densely distributed especially in the overlapping zone, both the splitting and fixing take a long time. This inspires us to split the area in a more clever way so that the number of trajectories intersecting the line is not that large.

Based on the directed Hausdorff distance, Table 5.3 compares the maps by single and by split-and-merge with the groundtruth on four urban datasets. Concerning that

Table 5.2: Comparison of map complexity and runtime.

Methods	Vertex Amount	Edge Amount	Length (km)	Runtime(second)
Athens small				
single	451	509	40	4.22
split-and-merge	460	519	40	4.38
Athens large				
single	4571	5333	430	115.70
split-and-merge	4779	5551	434	86.52
Chicago				
single	293	338	35	10.32
split-and-merge	314	360	35	10.95
Berlin				
single	1650	1867	160	110.01
split-and-merge	1763	1947	164	143.84
Delta				
single	457	508	20	6.29
split-and-merge	477	523	20	5.05
Aiguamolls				
single	1589	1719	115	18.17
split-and-merge	1620	1736	115	11.67
Garraf				
single	1415	1496	69	30.43
split-and-merge	1455	1518	73	28.43
Montseny				
single	1754	1877	86	29.52
split-and-merge	1786	1908	86	24.61

most values are the same, we can say the maps are almost the same.

Table 5.3: Comparison of directed Hausdorff distance measure of urban GPS datasets.

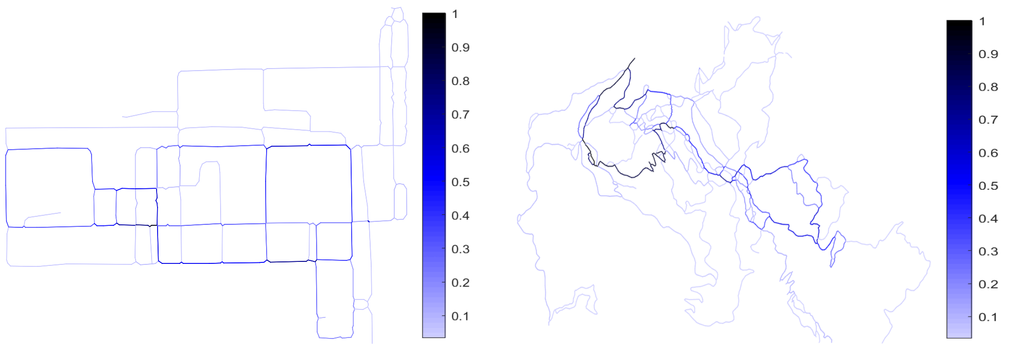
Methods	Directed Hausdorff Distance(m)							
	min	max	median	average	2%	5%	10%	15%
Athens small								
single	1	79	14	18	54	44	33	29
split-and-merge	1	79	14	18	54	44	34	29
Athens large								
single	1	130	15	18	51	41	33	29
split-and-merge	1	139	15	18	51	42	33	29
Chicago								
single	1	73	9	13	50	38	25	20
split-and-merge	1	73	9	13	51	40	31	20
Berlin								
single	1	162	14	18	51	42	34	29
split-and-merge	1	162	14	18	51	42	34	28

5.5.2 Applications of Edge Weight

In this section, we validate the performance of utilizing the edge weight. First, we visualize the maps with a colormap and analyze the advantage of removing the wrong edges. Second, we show the improvement of map quality based on the directed Hausdorff distance.

5.5.2.1 Generated Maps

We use the edge weight to filter out the wrong edges, and we call the maps as filtered maps. The threshold of edge weight includes the edge frequency f and edge length l, f is set as the smallest edge frequency and l as the median length of edges with frequency f. With higher values of f and l, the map will be simpler. We use the frequency of edges to assign colors to do the visualization. Figure D.1 shows the filtered maps of the Chicago and Montseny datasets (see Appendix D for the other datasets). The frequencies of edges are normalized to assign the colors from blue to black with the increasing frequency. Via the visualization with colors, the data distribution is more clear than the plain plot (see the definition in Section 5.2).



(a) Chicago (b) Montseny

Figure 5.6: Plots of filtered maps of the (a) Chicago and (b) Montseny datasets.

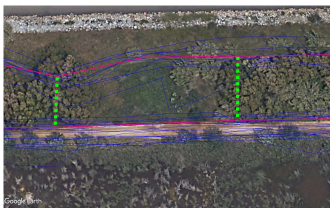
Next, we check the performance with the artifacts (see details in Section 4.4.2.2). Figure Fig. 5.7 shows the maps in artifacts that are different from that by performing the three-step framework. Utilizing the edge weight helps to filter out some edges, and we use a green dotted line to point out the removed edges. As we can see, all the wrong edges are removed. For example, in Figure 5.7(a), two artificial bridges are correctly filtered out.

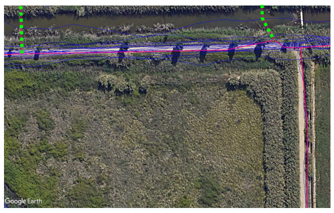
5.5.2.2 Map Filtering by the Edge Weight

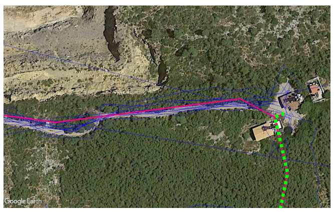
In this section, we evaluate quality of the filtered maps based on the map complexity (Table 5.4) and the directed Hausdorff distance (Table 5.5). Compared with the results of the “single” rows in Table 5.2 and Table 5.3, the map complexity is reduced due to the removal of vertices and edges. The generated maps are more similar to the groundtruth as some wrong edges are filtered out.

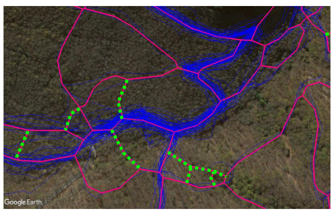
5.5.3 Summary

According to the visual inspection and quantitative evaluation, the maps generated by utilizing the split-and-merge strategy are almost the same with that by the three-step


(a) Artifact [$C3$]


(b) Artifact [$C5$]


(c) Artifact [$C6$]


(d) Artifact [$C7$]

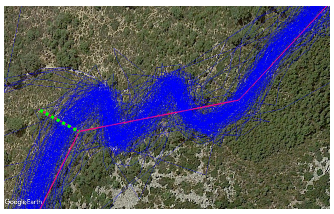

(e) Artifact [$S6$]

Figure 5.7: Plots of artifacts in filtered maps.

Table 5.4: Map complexity of filtered maps.

Dataset	Vertex Amount	Edge Amount	Length(km)
Athens small	433	473	39
Athens large	4364	4837	413
Chicago	272	307	33
Berlin	1650	1867	160
Delta	443	466	20
Aiguamolls	1537	1608	112
Garraf	1290	1338	66
Montseny	1678	1754	82

framework. The slight difference mainly happens in the overlapping zone. The strategy of using the overlapping zone is effective to reduce the influence of the change of the density surface because of splitting. Concerning the computational cost, the split-and-merge framework accelerates the generation especially when the trajectories in the overlapping zone are simple. However, if the data is dense around the splitting boundaries, the runtime increases by fixing the edges and vertices, which is demonstrated by the Berlin dataset. Taking advantage of the edge weight to visualize the map gives insight into the route popularity from the given data. Besides, filtering the map with edge weight helps to clear errors, such as the artificial bridges which are induced by

Table 5.5: Directed Hausdorff distance measure of the filtered maps of urban GPS datasets.

Datasets	Directed Hausdorff Distance(m)							
	min	max	median	average	2%	5%	10%	15%
Athens small	1	79	14	18	54	43	32	28
Athens large	1	130	14	17	49	40	32	28
Chicago	1	60	9	11	40	29	20	17
Berlin	1	162	14	18	51	42	34	29

noise. According to the quantitative evaluation, using the edge weight improves the map quality and accuracy.

5.6 Conclusions

In this chapter, we have presented two proposals to improve the three-step framework of map construction. First, considering the limitations of storage and computational cost, we have designed a split-and-merge strategy that allows handling large-scale data. The geographical area is split into small regions. As a novelty, we keep an overlapping zone between adjacent regions for the further map combination. For each region, the three-step framework is applied to generate a route graph. Finally, we merge the individual graphs into a global single graph taking advantage of the overlapping zone to fix the adjacent maps. Experiments on real datasets demonstrate that the split-and-merge strategy succeeds in producing an accurate global map and accelerating the process. Second, to remove the wrong edges (like the artificial bridges), we propose to compute the edge weight that combines length and frequency. Also, utilizing the edge eight to visualize the map gives insight into the importance or popularity of each route. The experiments show that utilizing the edge weight gives a better view of the map and improves the map quality, including map complexity and accuracy.

There is still room for improvement. First, a more clever way to split the geographical area, based on the density distribution instead of rigid segmentation, can be developed. This would reduce work and errors in fixing boundaries in the merging step. Second, the Slide tool always smooths the sharp turns considering the route is straight, which causes excessive simplification in turns and the wrong merging in narrow curves. A possible solution would be segmenting the trajectories into small sub-trajectories based on the direction change.

CHAPTER 6

Conclusions

Contents

6.1 Contributions

The main goal of this thesis is to develop advanced techniques to promote the task of trajectory analysis in two hot topics: anomaly detection and map construction. Based on this, we have introduced several algorithms, and the main contributions are summarized as follows:

1. We have presented an adaptive algorithm SHNN-CAD$^+$ for online trajectory anomaly detection based on a recent approach called SHNN-CAD. SHNN-CAD has the advantage of being parameter-light and dealing with online detection, while also has the defects that it requires a pre-defined anomaly threshold and does not take into account the direction attribute of trajectory data. Based on the observations, three improvements have been proposed. First, a data-adaptive anomaly threshold has been defined instead of determining it without any prior knowledge, which makes the algorithm more usable in practical applications. Second, a modified Hausdorff distance measure has been introduced to additionally distinguish the direction and to also reduce the computational complexity. Third, a re-do step has been added to make the detection process more flexible and accurate. Besides, more datasets have been designed and used. Extensive experiments on both real and synthetic datasets have shown that SHNN-CAD$^+$ enhances the performance of SHNN-CAD concerning accuracy and runtime.

2. We have developed a three-step framework that is fast, robust and simple skeleton-based to generate a route graph from GPS data. This framework is a smart combination of three well-known algorithms that are applied in three main steps, respectively. First, we estimate the density surface by mapping the trajectories into a grid, then smooth the density surface with Gaussian blur. Second, we recompute the density surface by adjusting the trajectory data by the Slide tool. This helps

to reduce the noise in data and make the density distribution more compact. Besides, we have proposed two strategies to solve the limitations of Slide. The first is to modify the distance component. The second is to find a solution to move the endpoints properly instead of staying fixed. Third, we use a thinning algorithm to obtain the skeleton, then convert it to a route map. Here, the Douglas-Peucker algorithm is used to simplify the vertices. The experiments have been carried on with both urban and hiking trajectory data. Via comparing with several map construction algorithms through the visual inspection and quantitative measures, the proposed framework has been demonstrated to obtain promising results concerning the data coverage, artificial, and similarity with the groundtruth.

3. We have made two proposals to improve the three-step framework. First, a split-and-merge strategy has been proposed to generate a route graph from large-scale GPS data. We split the geographical area into small regions where the map construction algorithm is performed. Then, we combine the graphs from different regions as the final output. Due to the changes in density surface, the graph in a region has slight offsets, especially around the splitting boundary. To solve this issue, an overlapping zone between two adjacent regions is kept for fixing the boundaries. Second, we compute the edge to help the result visualization and filter possible errors in the graph. We consider two factors, length and frequency, to give the weight of an edge. The experiments on real urban and hiking GPS data have shown that the split-and-merge strategy generates almost the same maps with running the three-step framework without splitting the geographical area, and has an advantage of less computational cost. Besides, applications of edge weights have shown satisfying results. Utilizing the edge weight to filter the generated maps is good to avoid the artifacts and improve the map quality.

6.2 Future work

In this thesis, we have addressed the problem of anomaly detection and map construction, while there are still some things left to try and improve. The future work will focus on the following directions:

1. In trajectory anomaly detection

   - In the field of trajectory analysis, measuring the similarity between trajectories is still an open challenge due to the specific properties of trajectory data. With the same two trajectories, different distance measures give different results especially when the trajectories have unequal number of points, thus there is no standard rule to determine whether two trajectories are similar or not. Both SHNN-CAD and SHNN-CAD$^+$ require distance computing as the basis to identify a test is far more different from the training set or not. In order to avoid the influence of inappropriate distance measure, an alternative would be utilizing some statistical models and information divergences, which avoids the direct use of a distance measure.

- In the current algorithm, all the trajectories in the training set are kept. However, as time goes on, the size of the training set will be very large and redundant trajectories will emerge. Thus, for the purpose of reducing the computational cost and storage, further research will consider incremental learning which prunes the historical data along with the update of the training set.

2. In map construction from GPS data

   - Defining parameters is always a big issue in different algorithms since the data are various. Via extensive experiments, the correlation might be found, which helps a lot to reduce the heavy work of determining several parameters when changing the dataset.

   - The Slide tool has the advantage of push similar trajectories to be closer with each other through the correction of three components: surface, distance, and angle. Due to that the angle component tries to maximize the vertex angle, which means to straight the adjacent edges, there is the risk of over-smoothing the turns and merging narrow curves. The possible solution could be segmenting the trajectories based on the direction change.

   - The proposed framework converts a skeleton to a route graph where the thinning algorithm is applied to obtain the skeleton from the density surface estimated from trajectories. Because the thinning algorithm generates the one-pixel-wide skeleton by iteratively removing the foreground pixels, while the density of each pixel is not taken into account, the final links in the skeleton always have some offsets compared with the groundtruth. Thus, it may be interesting to add the density at each pixel to perform the thinning algorithm.

   - Quantitatively comparing the map quality is an important but difficult work for developing good map construction algorithms. In literature, most evaluation measures focus on the similarity between generated map and groundtruth, ignoring the coverage of trajectory data. Due to the missing information of updated routes, the groundtruth is not suitable for comparison. In addition, there are many unknown routes discovered by outdoor activities, which are interesting and useful for route design and customization. Thus, a measure that compares the similarity between the generated map and the GPS data would be quite useful to indicate the coverage of data.

3. In split-and-merge strategy

   - The splitting way influences the computational cost and accuracy of map construction. Thus, the current rigid even segmentation could be improved by some data-adaptive strategies where the splitting boundary crosses through the low-density area.

6.3 Publications

Publications that support this thesis are:

- Yuejun Guo, Anton Bardera, Marta Fort and Rodrigo I. Silveira. Global schematic complete map construction from urban and hiking trajectory data. To submit to International Journal of Geographical Information Science, 2020.

- Yuejun Guo and Anton Bardera. *SHNN-CAD$^+$: An Improvement on SHNN-CAD for Adaptive Online Trajectory Anomaly Detection.* Sensors, vol. 19, no. 1, 2019.

- Anton Bardera, Marta Fort and Yuejun Guo. *Route Graph Construction from GPS Trajectory Data.* Accepted by XVIII Spanish Meeting on Computational Geometry, Girona, July 1-3, 2019.

Previous publications related with this thesis that have been achieved during phd study are:

- Yuejun Guo, Qing Xu, Peng Li, Mateu Sbert and Yu Yang. *Trajectory Shape Analysis and Anomaly Detection Utilizing Information Theory Tools.* Entropy, vol. 19, no. 7, page 323, 2017.

- Yuejun Guo, Qing Xu and Mateu Sbert. *IBVis: Interactive Visual Analytics for Information Bottleneck Based Trajectory Clustering.* Entropy, vol. 20, no. 3, page 159, 2018.

- Yuejun Guo, Qing Xu, Xiaoxiao Luo, Hao Wei, Hongjuan Bu and Mateu Sbert. *A Group-Based Signal Filtering Approach for Trajectory Abstraction and Restoration.* Neural Computing and Applications, vol. 29, no. 9, pages 371-387, May 2018.

Bibliography

[Agamennoni 2011] Gabriel Agamennoni, Juan I. Nieto and Eduardo M. Nebot. *Robust inference of principal road paths for intelligent transportation systems*. IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 1, pages 298–308, March 2011. (Cited on pages 23, 24, 25 and 78.)

[Ahmed 2012] Mahmuda Ahmed and Carola Wenk. *Constructing street networks from gps trajectories*. In Leah Epstein and Paolo Ferragina, editeurs, Algorithms – ESA 2012, pages 60–71, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. (Cited on pages 2, 21, 49, 70 and 78.)

[Ahmed 2014] Mahmuda Ahmed, Brittany Terese Fasy and Carola Wenk. *Local persistent homology based distance between maps*. In ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '14, pages 43–52, New York, NY, USA, 2014. ACM. (Cited on page 25.)

[Ahmed 2015a] Mahmuda Ahmed, Brittany Terese Fasy, Matt Gibson and Carola Wenk. *Choosing thresholds for density-based map construction algorithms*. In International Conference on Advances in Geographic Information Systems, SIGSPATIAL '15, pages 24:1–24:10, New York, NY, USA, 2015. ACM. (Cited on page 24.)

[Ahmed 2015b] Mahmuda Ahmed, Brittany Terese Fasy, Kyle S. Hickmann and Carola Wenk. *A path-based distance for street map comparison*. ACM Transactions on Spatial Algorithms and Systems, vol. 1, no. 1, pages 3:1–3:28, July 2015. (Cited on pages 25 and 27.)

[Ahmed 2015c] Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser and Carola Wenk. *A comparison and evaluation of map construction algorithms using vehicle tracking Data*. GeoInformatica, vol. 19, no. 3, pages 601–632, July 2015. (Cited on pages 8, 26, 49, 60, 70, 71, 75, 77 and 79.)

[Ahmed 2015d] Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser and Carola Wenk. Map construction algorithms. Springer International Publishing, Cham, 2015. (Cited on pages 2, 20, 23, 27 and 78.)

[Alt 1992] Helmut Alt and Michael Godau. *Measuring the resemblance of polygonal curves*. In Annual Symposium on Computational Geometry, pages 102–109. ACM, 1992. (Cited on page 9.)

[Alt 2009] Helmut Alt. *The computational geometry of comparing shapes*. In Susanne Albers, Helmut Alt and Stefan Näher, editeurs, Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday, pages 235–248, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. (Cited on page 33.)

[Annoni 2012] Ronald Annoni and Carlos Henrique Quartucci Forster. *Analysis of aircraft trajectories using Fourier descriptors and kernel density estimation*. In International IEEE Conference on Intelligent Transportation Systems, pages 1441–1446. IEEE, September 2012. (Cited on page 15.)

[Antunes 2001] Cláudia M. Antunes and Arlindo L. Oliveira. *Temporal data mining: an overview*. In KDD Workshop on Temporal Data Mining, volume 1, pages 1–13, 2001. (Cited on page 107.)

[Atev 2006] Stefan Atev, Osama Masoud and Nikos Papanikolopoulos. *Learning traffic patterns at intersections by spectral clustering of motion trajectories*. In International Conference on Intelligent Robots and Systems, pages 4851–4856, October 2006. (Cited on page 10.)

[Banerjee 2016] Prithu Banerjee, Pranali Yawalkar and Sayan Ranu. *MANTRA: a scalable approach to mining temporally anomalous sub-trajectories*. In ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pages 1415–1424, New York, NY, USA, 2016. ACM. (Cited on page 18.)

[Bao 2017] Liang Bao, Shanshan Wu, Weizhao Chen, Zisheng Zhu and Fan Yi. *Trajectory outlier detection based on partition-and-detection framework*. In International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery, pages 1978–1983, July 2017. (Cited on page 18.)

[Barnett 1974] Vic Barnett and Toby Lewis. Outliers in statistical data. Wiley, 1974. (Cited on page 1.)

[Berndt 1994] Donald J. Berndt and James Clifford. *Using dynamic time warping to find patterns in time series.* In KDD workshop, pages 359–370. Seattle, WA, 1994. (Cited on page 10.)

[Bhattacharyya 1946] Anil Bhattacharyya. *On a measure of divergence between two multinomial populations*. Sankhyā: The Indian Journal of Statistics (1933-1960), vol. 7, no. 4, pages 401–406, July 1946. (Cited on page 12.)

[Biagioni 2012a] James Biagioni and Jakob Eriksson. *Inferring road maps from global positioning system traces: survey and comparative evaluation*. Transportation Research Record, vol. 2291, no. 1, pages 61–71, 2012. (Cited on pages 2, 20, 23, 25, 26, 75, 77 and 78.)

[Biagioni 2012b] James Biagioni and Jakob Eriksson. *Map inference in the face of noise and disparity*. In International Conference on Advances in Geographic Information Systems, SIGSPATIAL '12, pages 79–88, New York, NY, USA, 2012. ACM. (Cited on pages 2, 3, 8, 24, 25, 49, 58, 71 and 78.)

[Birant 2007] Derya Birant and Alp Kut. *ST-DBSCAN: an algorithm for clustering spatial-temporal data*. Data & Knowledge Engineering, vol. 60, no. 1, pages 208–221, 2007. (Cited on page 15.)

[Bradley 1997] Andrew P. Bradley. *The use of the area under the roc curve in the evaluation of machine learning algorithms*. Pattern Recognition, vol. 30, no. 7, pages 1145–1159, 1997. (Cited on page 19.)

[Bruntrup 2005] René Bruntrup, Stefan Edelkamp, Shahid Jabbar and Björn Scholz. *Incremental map generation with gps traces*. In IEEE Intelligent Transportation Systems, pages 574–579, September 2005. (Cited on pages 20 and 78.)

[Buchin 2009] Kevin Buchin, Maike Buchin and Yusu Wang. *Exact algorithms for partial curve matching via the Fréchet distance*. In Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '09, pages 645–654, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics. (Cited on pages 21 and 49.)

[Buchin 2017] Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristan, Rodrigo I. Silveira, Frank Staals and Carola Wenk. *Clustering trajectories for map construction*. In ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '17, pages 14:1–14:10, New York, NY, USA, 2017. ACM. (Cited on page 25.)

[Calderara 2011] Simone Calderara, Andrea Prati and Rita Cucchiara. *Mixtures of von Mises distributions for people trajectory shape analysis*. IEEE Transactions on Circuits and Systems for Video Technology, vol. 21, no. 4, pages 457–471, April 2011. (Cited on page 12.)

[Cao 2009] Lili Cao and John Krumm. *From gps traces to a routable road map*. In ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '09, pages 3–12, New York, NY, USA, 2009. ACM. (Cited on pages 2, 8, 20, 25, 48, 49, 50, 70, 75 and 78.)

[Chandola 2009] Varun Chandola, Arindam Banerjee and Vipin Kumar. *Anomaly detection: a survey*. ACM Comput. Surv., vol. 41, no. 3, pages 15:1–15:58, July 2009. (Cited on page 33.)

[Chen 2008] Chen Chen and Yinhang Cheng. *Roads digital map generation with multi-track gps data*. In International Workshop on Education Technology and Training & International Workshop on Geoscience and Remote Sensing, pages 508–511, December 2008. (Cited on pages 24, 25 and 48.)

[Chen 2010] Daniel Chen, Leonidas J. Guibas, John Hershberger and Jian Sun. *Road network reconstruction for organizing paths*. In Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, pages 1309–1320, 2010. (Cited on pages 24, 25 and 78.)

[Chen 2015] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen and Gustavo Batista. *UCR time series classification archive*. www.cs.ucr.edu/~eamonn/time_series_data/, July 2015. Online; accessed on 09 December 2019. (Cited on page 107.)

[Choong 2017] Mei Yeen Choong, Lorita Angeline, Renee Ka Yin Chin, Kiam Beng Yeo and Kenneth Tze Kin Teo. *Modeling of vehicle trajectory clustering based on lcss for traffic pattern extraction*. In International Conference on Automatic Control and Intelligent Systems, pages 74–79, October 2017. (Cited on page 11.)

[Cortes 1995] Corinna Cortes and Vladimir Vapnik. *Support-vector networks*. Machine Learning, vol. 20, no. 3, pages 273–297, September 1995. (Cited on page 16.)

[Das 2018] Deepan Das and Deepak Mishra. *Unsupervised anomalous trajectory detection for crowded scenes*. In International Conference on Industrial and Information Systems (ICIIS), pages 27–31, December 2018. (Cited on page 16.)

[Davies 2006] Jonathan J. Davies, Alastair R. Beresford and Andy Hopper. *Scalable, distributed, real-time map generation*. IEEE Pervasive Computing, vol. 5, no. 4, pages 47–54, October 2006. (Cited on pages 2, 8, 24, 25, 49, 50 and 70.)

[Deng 2011] Min Deng, Qiliang Liu, Tao Cheng and Yan Shi. *An adaptive spatial clustering algorithm based on delaunay triangulation*. Computers, Environment and Urban Systems, vol. 35, no. 4, pages 320–332, 2011. (Cited on page 22.)

[Deng 2018] Min Deng, Jincai Huang, Yunfei Zhang, Huimin Liu, Luliang Tang, Jianbo Tang and Xuexi Yang. *Generating urban road intersection models from low-frequency gps trajectory data*. International Journal of Geographical Information Science, vol. 32, no. 12, pages 2337–2361, 2018. (Cited on pages 2, 22 and 48.)

[Dey 2017] Tamal K. Dey, Jiayuan Wang and Yusu Wang. *Improved road network reconstruction using discrete morse theory*. In ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '17, pages 58:1–58:4, New York, NY, USA, 2017. ACM. (Cited on page 24.)

[Ding 2008] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang and Eamonn Keogh. *Querying and mining of time series data: experimental comparison of representations and distance measures*. VLDB Endowment, vol. 1, no. 2, pages 1542–1552, August 2008. (Cited on pages 9, 15 and 37.)

[Douglas 1973] David H. Douglas and Thomas K. Peucker. *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature*. Cartographica: the international journal for geographic information and geovisualization, vol. 10, no. 2, pages 112–122, 1973. (Cited on page 58.)

[Duran 2020] David Duran, Vera Sacristán and Rodrigo I. Silveira. Map construction algorithms: a local evaluation through hiking data. GeoInformatica, to appear, 2020. (Cited on pages 49, 60, 62, 63 and 70.)

[Edelkamp 2003] Stefan Edelkamp and Stefan Schrödl. *Route planning and map inference with global positioning traces*. In Rolf Klein, Hans-Werner Six and Lutz Wegner, editeurs, Computer Science in Perspective: Essays Dedicated to Thomas

Ottmann, pages 128–151, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cited on pages 2, 23, 49, 50, 70 and 78.)

[Eiter 1994] Thomas Eiter and Heikki Mannila. *Computing discrete Fréchet distance*. Technical report CD-TR 94/64, Christian Doppler Laboratories, University of Vienna, April 1994. (Cited on page 10.)

[Ester 1996] Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu. *A density-based algorithm for discovering clusters in large spatial databases with noise*. In International Conference on Knowledge Discovery and Data Mining, KDD'96, pages 226–231. AAAI Press, August 1996. (Cited on page 14.)

[Faloutsos 1994] Christos Faloutsos, Mudumbai Ranganathan and Yannis Manolopoulos. *Fast subsequence matching in time-series databases*. SIGMOD Rec., vol. 23, no. 2, pages 419–429, May 1994. (Cited on page 9.)

[Fathi 2010] Alireza Fathi and John Krumm. *Detecting road intersections from gps traces*. In Sara Irina Fabrikant, Tumasch Reichenbacher, Marc van Kreveld and Christoph Schlieder, editeurs, Geographic Information Science, pages 56–69, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. (Cited on pages 22, 25 and 78.)

[Fu 2005] Zhouyu Fu, Weiming Hu and Tieniu Tan. *Similarity based vehicle trajectory clustering and anomaly detection*. In IEEE International Conference on Image Processing, volume 2, pages II–602–5, September 2005. (Cited on pages 13 and 15.)

[Gariel 2011] Maxime Gariel, Ashok N. Srivastava and Eric Feron. *Trajectory clustering and an application to airspace monitoring*. IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 4, pages 1511–1524, December 2011. (Cited on pages 1, 11, 13 and 14.)

[Gonzalez 2006] Rafael C. Gonzalez and Richard E. Woods. Digital image processing (3rd edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. (Cited on pages 55 and 58.)

[Görnitz 2013] Nico Görnitz, Marius Kloft, Konrad Rieck and Ulf Brefeld. *Toward supervised anomaly detection*. Journal of Artificial Intelligence Research, vol. 46, pages 235–262, February 2013. (Cited on page 1.)

[Guo 1989] Zicheng Guo and Richard W. Hall. *Parallel thinning with two-subiteration algorithms*. Communications of the ACM, vol. 32, no. 3, pages 359–373, March 1989. (Cited on page 53.)

[Guo 2007] Tao Guo, Kazuaki Iwamura and Masashi Koga. *Towards high accuracy road maps generation from massive GPS Traces data*. In IEEE International Geoscience and Remote Sensing Symposium, pages 667–670, July 2007. (Cited on pages 2, 24 and 48.)

[Guo 2014] Yuejun Guo, Qing Xu, Yu Yang, Sheng Liang, Yu Liu and Mateu Sbert. *Anomaly detection based on trajectory analysis using kernel density estimation and information bottleneck techniques*. Technical report 108, University of Girona, 2014. (Cited on page 13.)

[Guo 2015] Yuejun Guo, Qing Xu, Sheng Liang, Yang Fan and Mateu Sbert. *XaIBO: an extension of aib for trajectory clustering with outlier*. In Sabri Arik, Tingwen Huang, Weng Kin Lai and Qingshan Liu, editeurs, Neural Information Processing, pages 423–431, Cham, 2015. Springer International Publishing. (Cited on pages 9 and 13.)

[Guo 2016] Yuejun Guo, Qing Xu, Yang Fan, Sheng Liang and Mateu Sbert. *Fast Agglomerative Information Bottleneck Based Trajectory Clustering*. In Akira Hirose, Seiichi Ozawa, Kenji Doya, Kazushi Ikeda, Minho Lee and Derong Liu, editeurs, Neural Information Processing, pages 425–433, Cham, 2016. Springer International Publishing. (Cited on page 13.)

[Guo 2017] Yuejun Guo, Qing Xu, Peng Li, Mateu Sbert and Yu Yang. *Trajectory shape analysis and anomaly detection utilizing information theory tools*. Entropy, vol. 19, no. 7, page 323, 2017. (Cited on pages 12, 13, 16, 19 and 40.)

[Guo 2018a] Yuejun Guo. *Matlab implementation*. http://gilabparc.udg.edu/trajectory/experiments/Experiments.zip, 2018. Online; accessed on 09 December 2019. (Cited on page 37.)

[Guo 2018b] Yuejun Guo. *Synthetic trajectories by Guo*. http://gilabparc.udg.edu/trajectory/data/SyntheticTrajectories.zip, 2018. Online; accessed on 09 December 2019. (Cited on page 42.)

[Guo 2018c] Yuejun Guo, Qing Xu, Xiaoxiao Luo, Hao Wei, Hongjuan Bu and Mateu Sbert. *A group-based signal filtering approach for trajectory abstraction and restoration*. Neural Computing and Applications, vol. 29, no. 9, pages 371–387, May 2018. (Cited on page 9.)

[Gupta 2014] Manish Gupta, Jing Gao, Charu C. Aggarwal and Jiawei Han. *Outlier detection for temporal data: a survey*. IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 9, pages 2250–2267, September 2014. (Cited on page 14.)

[Han 2011] Jiawei Han, Jian Pei and Micheline Kamber. Data mining: concepts and techniques. Elsevier, 2011. (Cited on pages 13 and 14.)

[Haritaoglu 2000] Ismail Haritaoglu, David Harwood and Larry S. Davis. *W4: real-time surveillance of people and their activities*. IEEE Transactions on Pattern Analysis & Machine Intelligence, vol. 22, no. 8, pages 809–830, August 2000. (Cited on page 1.)

[Hu 2007] Weiming Hu, Dan Xie, Zhouyu Fu, Wenrong Zeng and Steve Maybank. *Semantic-based surveillance video retrieval*. IEEE Transactions on Image Processing, vol. 16, no. 4, pages 1168–1181, April 2007. (Cited on pages 9 and 11.)

[Hu 2018] Kaixi Hu, Pan Duan, Bei Hu and Qichang Duan. *IBTOD: an isolation-based method to detect outlying sub-trajectories on multi-factors*. In IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference, pages 1912–1918, May 2018. (Cited on pages 11 and 18.)

[Huang 2018] Jincai Huang, Min Deng, Jianbo Tang, Shuling Hu, Huimin Liu, Sembeto Wariyo and Jinqiang He. *Automatic generation of road maps from low quality gps trajectory data via structure learning*. IEEE Access, vol. 6, pages 71965–71975, 2018. (Cited on pages 2, 25, 48 and 78.)

[Ismail 2015] Anas Ismail and Antoine Vigneron. *A new trajectory similarity measure for gps data*. In ACM SIGSPATIAL International Workshop on GeoStreaming, IWGS '15, pages 19–22, New York, NY, USA, 2015. ACM. (Cited on page 12.)

[Izakian 2016] Zahedeh Izakian, Mohammad Saadi Mesgari and Ajith Abraham. *Automated clustering of trajectory data using a particle swarm optimization*. Computers, Environment and Urban Systems, vol. 55, pages 55–65, 2016. (Cited on page 10.)

[Jiang 2004] Sheng-Yi Jiang and Yu-Ming Xu. *An efficient clustering algorithm*. In International Conference on Machine Learning and Cybernetics, volume 3, pages 1513–1518, August 2004. (Cited on page 15.)

[Jiang 2008] Sheng-Yi Jiang and Qing-Bo An. *Clustering-based outlier detection method*. In International Conference on Fuzzy Systems and Knowledge Discovery, volume 2, pages 429–433, October 2008. (Cited on page 15.)

[Jiang 2009] Fan Jiang, Ying Wu and Aggelos K. Katsaggelos. *A dynamic hierarchical clustering method for trajectory-based unusual video event detection*. IEEE Transactions on Image Processing, vol. 18, no. 4, pages 907–913, April 2009. (Cited on pages 13, 16 and 19.)

[Karagiorgou 2012] Sophia Karagiorgou and Dieter Pfoser. *On vehicle tracking data-based road network generation*. In International Conference on Advances in Geographic Information Systems, SIGSPATIAL '12, pages 89–98, New York, NY, USA, 2012. ACM. (Cited on pages 1, 2, 8, 22, 25, 26, 49, 50, 70, 71 and 78.)

[Karagiorgou 2013] Sophia Karagiorgou, Dieter Pfoser and Dimitrios Skoutas. *Segmentation-based road network construction*. In ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL'13, pages 460–463, New York, NY, USA, 2013. ACM. (Cited on pages 2, 22, 50 and 78.)

[Kaufman 1987] Leonard Kaufman and Peter J. Rousseeuw. *Clustering by means of medoids*. In Statistical Data Analysis Based on the L_1–Norm and Related Methods, pages 405–416. North Holland / Elsevier, 1987. (Cited on page 13.)

[Keogh 2004] Eamonn Keogh, Stefano Lonardi and Chotirat Ann Ratanamahatana. *Towards parameter-free data mining*. In ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '04, pages 206–215, New York, NY, USA, 2004. ACM. (Cited on pages 2 and 29.)

[Keogh 2005] Eamonn Keogh, Jessica Lin and Ada Fu. *HOT SAX: efficiently finding the most unusual time series subsequence*. In IEEE International Conference on Data Mining, pages 226–233, November 2005. (Cited on page 16.)

[Kong 2018] Xiangjie Kong, Menglin Li, Kai Ma, Kaiqi Tian, Mengyuan Wang, Zhaolong Ning and Feng Xia. *Big trajectory data: a survey of applications and services*. IEEE Access, vol. 6, pages 58295–58306, 2018. (Cited on pages 1 and 8.)

[Kruskal 1952] William H. Kruskal and W. Allen Wallis. *Use of ranks in one-criterion variance analysis*. American Statistical Association, vol. 47, no. 260, pages 583–621, 1952. (Cited on page 40.)

[Kumar 2017] Dheeraj Kumar, James C. Bezdek, Sutharshan Rajasegarar, Christopher Leckie and Marimuthu Palaniswami. *A visual-numeric approach to clustering and anomaly detection for trajectory data*. The Visual Computer, vol. 33, no. 3, pages 265–281, 2017. (Cited on pages 15 and 19.)

[Lam 1992] Louisa Lam, Seong-Whan Lee and Ching Y. Suen. *Thinning methodologies-a comprehensive survey*. IEEE Transactions on Pattern Analysis and Machince Intelligence, vol. 14, no. 9, pages 869–885, September 1992. (Cited on page 53.)

[Laxhammar 2010] Rikard Laxhammar and Göran Falkman. *Conformal prediction for distribution-independent anomaly detection in streaming vessel data*. In International Workshop on Novel Data Stream Pattern Mining Techniques, StreamKDD '10, pages 47–55, New York, NY, USA, 2010. ACM. (Cited on page 32.)

[Laxhammar 2011] Rikard Laxhammar and Göran Falkman. *Sequential conformal anomaly detection in trajectories based on Hausdorff distance*. In International Conference on Information Fusion, pages 1–8. IEEE, July 2011. (Cited on pages 9, 17, 19, 32 and 33.)

[Laxhammar 2013] Rikard Laxhammar. *Synthetic trajectories by Laxhammar*. https://www.researchgate.net/publication/236838887_Synthetic_trajectories, 2013. Online; accessed on 09 December 2019. (Cited on page 41.)

[Laxhammar 2014a] Rikard Laxhammar. *Conformal anomaly detection: detecting abnormal trajectories in surveillance applications*. PhD thesis, University of Skövde, School of Informatics, 2014. (Cited on page 31.)

[Laxhammar 2014b] Rikard Laxhammar and Göran Falkman. *Online learning and sequential anomaly detection in trajectories*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 36, no. 6, pages 1158–1173, June 2014. (Cited on pages 3, 9, 17, 19, 30, 31, 33, 40, 41, 42 and 43.)

[Lazarević 2007] Aleksandar Lazarević. *First set of recorded video trajectories*. https://www-users.cs.umn.edu/~lazar027/inclof/, 2007. Online; accessed on 09 December 2019. (Cited on page 40.)

[Lee 2008] Jae-Gil Lee, Jiawei Han and Xiaolei Li. *Trajectory outlier detection: a partition-and-detect framework*. In International Conference on Data Engineering, pages 140–149. IEEE, April 2008. (Cited on pages 1, 11 and 17.)

[Li 2012] Jun Li, Qiming Qin, Chao Xie and Yue Zhao. *Integrated use of spatial and semantic relationships for extracting road networks from floating car data*. International Journal of Applied Earth Observation and Geoinformation, vol. 19, pages 238–247, 2012. (Cited on pages 21 and 78.)

[Li 2016] Hengfeng Li, Lars Kulik and Kotagiri Ramamohanarao. *Automatic generation and validation of road maps from gps trajectory data sets*. In ACM International on Conference on Information and Knowledge Management, CIKM '16, pages 1523–1532, New York, NY, USA, 2016. ACM. (Cited on pages 2, 23, 25, 48 and 78.)

[Li 2017] Lin Li, Daigang Li, Xiaoyu Xing, Fan Yang, Wei Rong and Haihong Zhu. *Extraction of road intersections from gps traces based on the dominant orientations of roads*. ISPRS International Journal of Geo-Information, vol. 6, no. 12, 2017. (Cited on pages 23, 25 and 78.)

[Li 2018] Huanhuan Li, Jingxian Liu, Kefeng Wu, Zaili Yang, Ryan Wen Liu and Naixue Xiong. *Spatio-temporal vessel trajectory clustering based on data mapping and density*. IEEE Access, vol. 6, pages 58939–58954, 2018. (Cited on page 12.)

[Lin 1991] Jianhua Lin. *Divergence measures based on the Shannon entropy*. IEEE Transactions on Information Theory, vol. 37, no. 1, pages 145–151, January 1991. (Cited on page 12.)

[Liu 2012a] Xuemei Liu, James Biagioni, Jakob Eriksson, Yin Wang, George Forman and Yanmin Zhu. *Mining large-scale, sparse gps traces for map inference: comparison of approaches*. In ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, pages 669–677, New York, NY, USA, 2012. ACM. (Cited on pages 2 and 25.)

[Liu 2012b] Xuemei Liu, Yanmin Zhu, Yin Wang, George Forman, Lionel M. Ni, Fang Yu and Minglu Li. *Road recognition using coarse-grained vehicular traces*. Technical report HPL-2012-26, HP Labs, 2012. (Cited on pages 23, 25 and 78.)

[Luo 2015] Xiaoxiao Luo, Qing Xu, Yuejun Guo, Hao Wei and Yimin Lv. *Trajectory abstracting with group-based signal denoising*. In Sabri Arik, Tingwen Huang, Weng Kin Lai and Qingshan Liu, editeurs, Neural Information Processing. Lecture Notes in Computer Science, volume 9491, pages 452–461, Cham, 2015. Springer International Publishing. (Cited on page 9.)

[Mach 2014a] Paul Mach. *Implementation of slide*. https://github.com/paulmach/slide, 2014. Online; accessed on 09 December 2019. (Cited on pages 51 and 52.)

[Mach 2014b] Paul Mach. *Strava slide tool*. https://labs.strava.com/slide/, 2014. Online; accessed on 09 December 2019. (Cited on pages 51 and 57.)

[MacQueen 1967] J. MacQueen. *Some methods for classification and analysis of multivariate observations*. In Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, pages 281–297, Calif. Berkeley, 1967. University of California Press. (Cited on page 13.)

[Majecka 2009] Barbara Majecka. *Statistical models of pedestrian behaviour in the forum*. PhD thesis, MSc Dissertation, School of Informatics, University of Edinburgh, 2009. (Cited on pages 1 and 8.)

[Mariescu-Istodor 2018] Radu Mariescu-Istodor and Pasi Fränti. *CellNet: inferring road networks from gps trajectories*. ACM Transactions Spatial Algorithms Systems, vol. 4, no. 3, pages 8:1–8:22, September 2018. (Cited on page 22.)

[Maurer 2003] Calvin R. Maurer, Rensheng Qi and Vijay Raghavan. *A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 2, pages 265–270, February 2003. (Cited on page 81.)

[Mazimpaka 2016] Jean Damascène Mazimpaka and Sabine Timpf. *Trajectory data mining: a review of methods and applications*. Journal of Spatial Information Science, vol. 2016, no. 13, pages 61–99, 2016. (Cited on page 8.)

[Mcfadyen 2016] Aaron Mcfadyen, Mark O'Flynn, Terrance Martin and Duncan Campbell. *Aircraft trajectory clustering techniques using circular statistics*. In IEEE Aerospace Conference, pages 1–10, March 2016. (Cited on page 12.)

[Meng 2018] Fanrong Meng, Guan Yuan, Shaoqian Lv, Zhixiao Wang and Shixiong Xia. *An overview on trajectory outlier detection*. Artificial Intelligence Review, February 2018. (Cited on pages 14 and 29.)

[Morris 2009a] Brendan Morris and Mohan Trivedi. *Learning trajectory patterns by clustering: experimental studies and comparative evaluation*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 312–319, June 2009. (Cited on pages 9, 11, 13 and 39.)

[Morris 2009b] Brendan Morris and Mohan Trivedi. *Trajectory clustering datasets*. http://cvrr.ucsd.edu/bmorris/datasets/dataset_trajectory_clustering.html, 2009. Online; accessed on 09 December 2019. (Cited on page 39.)

[Nandeshwar 2011] Ashutosh Nandeshwar, Tim Menzies and Adam Nelson. *Learning patterns of university student retention*. Expert Systems with Applications, vol. 38, no. 12, pages 14984–14996, 2011. (Cited on pages 19 and 42.)

[Niehöfer 2009] Brian Niehöfer, Ralf Burda, Christian Wietfeld, Franziskus Bauer and Oliver Lueert. *GPS community map generation for enhanced routing methods based on trace-collection by mobile phones*. In International Conference on Advances in Satellite and Space Communications, pages 156–161, July 2009. (Cited on pages 21, 25 and 78.)

[Ord 1995] J. K. Ord and Arthur Getis. *Local spatial autocorrelation statistics: distributional issues and an application*. Geographical Analysis, vol. 27, no. 4, pages 286–306, 1995. (Cited on page 22.)

[Parmar 2017] Jagruti D. Parmar and Jalpa T. Patel. *Anomaly detection in data mining: a review*. International Journal of Advanced Research in Computer Science and Software Engineering, vol. 7, no. 4, pages 32–40, 2017. (Cited on page 14.)

[Pfoser 2016] Dieter Pfoser and Carola Wenk. *Map construction algorithms*. http://mapconstruction.org/, 2016. Online; accessed on 09 December 2019. (Cited on pages 60 and 71.)

[Piciarelli 2006] Claudio Piciarelli and Gian Luca Foresti. *On-line trajectory clustering for anomalous events detection*. Pattern Recognition Letters, vol. 27, no. 15, pages 1835–1842, 2006. Vision for Crime Detection and Prevention. (Cited on page 11.)

[Piciarelli 2008a] Claudio Piciarelli, Christian Micheloni and Gian Luca Foresti. *Synthetic trajectory dataset*. https://avires.dimi.uniud.it/papers/trclust/, 2008. Online; accessed on 09 December 2019. (Cited on pages 39, 57 and 58.)

[Piciarelli 2008b] Claudio Piciarelli, Christian Micheloni and Gian Luca Foresti. *Synthetic trajectory generator*. https://avires.dimi.uniud.it/papers/trclust/create_ts2.m, 2008. Online; accessed on 09 December 2019. (Cited on page 42.)

[Piciarelli 2008c] Claudio Piciarelli, Christian Micheloni and Gian Luca Foresti. *Trajectory-based anomalous event detection*. IEEE Transactions on Circuits and Systems for Video Technology, vol. 18, no. 11, pages 1544–1554, November 2008. (Cited on pages 16 and 19.)

[Piciarelli 2011] Claudio Piciarelli and Gian Luca Foresti. *Surveillance-oriented event detection in video streams*. IEEE Intelligent Systems, vol. 26, no. 3, pages 32–41, May 2011. (Cited on page 16.)

[Powell 2001] Mark D. Powell and Sim D. Aberson. *Accuracy of United States tropical cyclone landfall forecasts in the Atlantic basin (1976–2000)*. Bulletin of the American Meteorological Society, vol. 82, no. 12, pages 2749–2768, 2001. (Cited on page 1.)

[Prati 2008] Andrea Prati, Simone Calderara and Rita Cucchiara. *Using circular statistics for trajectory shape analysis*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8, June 2008. (Cited on pages 13, 16 and 19.)

[Qiao 2002] Yan Qiao, XW Xin, Yang Bin and S. Ge. *Anomaly intrusion detection method based on HMM*. Electronics letters, vol. 38, no. 13, pages 663–664, 2002. (Cited on page 16.)

[Ramaswamy 2000] Sridhar Ramaswamy, Rajeev Rastogi and Kyuseok Shim. *Efficient algorithms for mining outliers from large data sets*. SIGMOD Record, vol. 29, no. 2, pages 427–438, May 2000. (Cited on page 15.)

[Román 2018] Ignacio San Román, Isaac Martín de Diego, Cristina Conde and Enrique Cabello. *Outlier trajectory detection through a context-aware distance*. Pattern Analysis and Applications, pages 1–9, August 2018. (Cited on page 12.)

[Sander 2010] Joerg Sander. *Density-based clustering*. In Claude Sammut and Geoffrey I. Webb, editeurs, Encyclopedia of Machine Learning, pages 270–273, Boston, MA, 2010. Springer US. (Cited on page 14.)

[Schroedl 2004] Stefan Schroedl, Kiri Wagstaff, Seth Rogers, Pat Langley and Christopher Wilson. *Mining gps traces for map refinement*. Data Mining and Knowledge Discovery, vol. 9, no. 1, pages 59–87, July 2004. (Cited on pages 23, 25 and 78.)

[Shi 2009] Wenhuan Shi, Shuhan Shen and Yuncai Liu. *Automatic generation of road network map from massive gps, vehicle trajectories*. In International IEEE Conference on Intelligent Transportation Systems, pages 1–6, October 2009. (Cited on pages 24, 25 and 58.)

[Singh 2017] Pradeep Singh and Prateek A. Meshram. *Survey of density based clustering algorithms and its variants*. In International Conference on Inventive Computing and Informatics, pages 920–926, November 2017. (Cited on page 14.)

[Smith 1981] P.R. Smith. *Bilinear interpolation of digital images*. Ultramicroscopy, vol. 6, no. 2, pages 201–204, 1981. (Cited on page 52.)

[Stanojevic 2018] Rade Stanojevic, Sofiane Abbar, Saravanan Thirumuruganathan, Sanjay Chawla, Fethi Filali and Ahid Aleimat. *Robust road map inference through network alignment of trajectories*. In SIAM International Conference on Data Mining, pages 135–143, 2018. (Cited on pages 23, 26 and 78.)

[Tan 2007] Pang-Ning Tan, Michael Steinbach and Vipin Kumar. Introduction to data mining. Pearson Education India, 2007. (Cited on page 37.)

[Tang 2017] Luliang Tang, Chang Ren, Zhang Liu and Qingquan Li. *A road map refinement method using delaunay triangulation for big trace data*. ISPRS International Journal of Geo-Information, vol. 6, no. 2, 2017. (Cited on pages 21, 23 and 78.)

[Tang 2019] Jianbo Tang, Min Deng, Jincai Huang and Huimin Liu. *A novel method for road intersection construction from vehicle trajectory data*. IEEE Access, vol. 7, pages 95065–95074, July 2019. (Cited on page 22.)

[Vovk 2005] Vladimir Vovk, Alex Gammerman and Glenn Shafer. Algorithmic learning in a random world. Springer Science & Business Media, January 2005. (Cited on pages 30 and 31.)

[Wang 2011] Shuliang Wang, Wenyan Gan, Deyi Li and Deren Li. *Data field for hierarchical clustering*. International Journal of Data Warehousing and Mining, vol. 7, no. 4, pages 43–63, 2011. (Cited on page 13.)

[Wang 2015] Suyi Wang, Yusu Wang and Yanjie Li. *Efficient map reconstruction and augmentation via topological methods*. In 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '15, pages 25:1–25:10, New York, NY, USA, 2015. ACM. (Cited on pages 2, 24, 48, 49, 51, 60, 70 and 76.)

[Wang 2016] Jiayuan Wang. *Graph reconstruction*. https://github.com/wangjiayuan007/graph_recon_DM, 2016. Online; accessed on 09 December 2019. (Cited on page 60.)

[Wang 2017] Jing Wang, Chaoliang Wang, Xianfeng Song and Venkatesh Raghavan. *Automatic intersection and traffic rule detection by mining motor-vehicle gps trajectories*. Computers, Environment and Urban Systems, vol. 64, pages 19–29, 2017. (Cited on page 22.)

[Worrall 2007] Stewart Worrall and Eduardo Nebot. *Automated process for generating digitised maps through gps data compression*. In Australasian Conference on Robotics and Automation. Brisbane: ACRA, 2007. (Cited on page 23.)

[Wu 2013] Junwei Wu, long Zhu Yun, Tao Ku and Liang Wang. *Detecting road intersections from coarse-gained gps traces based on clustering*. Journal of Computers, vol. 8, no. 11, pages 2959–2965, November 2013. (Cited on pages 22 and 78.)

[Xie 2017] Xingzhe Xie, Wenzhi Liao, Hamid Aghajan, Peter Veelaert and Wilfried Philips. *Detecting road intersections from gps traces using longest common subsequence algorithm*. ISPRS International Journal of Geo-Information, vol. 6, no. 1, 2017. (Cited on page 23.)

[Yankov 2008] Dragomir Yankov, Eamonn Keogh and Umaa Rebbapragada. *Disk aware discord discovery: finding unusual time series in terabyte sized datasets*. Knowledge and Information Systems, vol. 17, no. 2, pages 241–262, November 2008. (Cited on page 17.)

[Yim 2015] Odilia Yim and Kylee T. Ramdeen. *Hierarchical cluster analysis: comparison of three linkage measures and application to psychological data*. The Quantitative Methods for Psychology, vol. 11, no. 1, pages 8–21, 2015. (Cited on page 13.)

[Ying 2009] Xia Ying, Zhang Xu and Wang Guo Yin. *Cluster-based congestion outlier detection method on trajectory data*. In International Conference on Fuzzy Systems and Knowledge Discovery, volume 5, pages 243–247, August 2009. (Cited on page 14.)

[Yu 2018] Qingying Yu, Yonglong Luo, Chuanming Chen and Xiaohan Wang. *Trajectory outlier detection approach based on common slices sub-sequence*. Applied Intelligence, vol. 48, no. 9, pages 2661–2680, September 2018. (Cited on pages 1 and 18.)

[Yuan 2017a] Guan Yuan, Penghui Sun, Jie Zhao, Daxing Li and Canwei Wang. *A review of moving object trajectory clustering algorithms*. Artificial Intelligence Review, vol. 47, no. 1, pages 123–144, January 2017. (Cited on pages 9, 13 and 14.)

[Yuan 2017b] Yuan Yuan, Dong Wang and Qi Wang. *Anomaly detection in traffic scenes via spatial-aware motion reconstruction*. IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 5, pages 1198–1209, May 2017. (Cited on pages 18, 19 and 20.)

[Zhang 2006] Zhang Zhang, Kaiqi Huang and Tieniu Tan. *Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes*. In International Conference on Pattern Recognition, pages 1135–1138, August 2006. (Cited on page 9.)

[Zhang 2010] Lijuan Zhang, Frank Thiemann and Monika Sester. *Integration of gps traces with road map*. In International Workshop on Computational Transportation Science, IWCTS '10, pages 17–22, New York, NY, USA, 2010. ACM. (Cited on pages 21 and 25.)

[Zhang 2011] Daqing Zhang, Nan Li, Zhi-Hua Zhou, Chao Chen, Lin Sun and Shijian Li. *iBAT: detecting anomalous taxi trajectories from gps traces*. In International Conference on Ubiquitous Computing, UbiComp '11, pages 99–108, New York, NY, USA, 2011. ACM. (Cited on pages 17, 19 and 20.)

[Zheng 2015a] Yu Zheng. *Trajectory data mining: an overview*. ACM Transactions on Intelligent Systems Technology, vol. 6, no. 3, pages 1–41, May 2015. (Cited on page 1.)

[Zheng 2015b] Yu Zheng. *Trajectory data mining: an overview*. ACM Transactions on Intelligent Systems and Technology, vol. 6, no. 3, pages 29:1–29:41, May 2015. (Cited on page 8.)

[Zheng 2018] Ke Zheng and Dunyao Zhu. *A novel clustering algorithm of extracting road network from low-frequency floating car data*. Cluster Computing, pages 1–10, January 2018. (Cited on pages 2, 21, 25, 48 and 78.)

[Zhu 2015] Jie Zhu, Wei Jiang, An Liu, Guanfeng Liu and Lei Zhao. *Time-dependent popular routes based trajectory outlier detection*. In Jianyong Wang, Wojciech Cellary, Dingding Wang, Hua Wang, Shu-Ching Chen, Tao Li and Yanchun Zhang, editeurs, Web Information Systems Engineering, pages 16–30, Cham, 2015. Springer International Publishing. (Cited on page 17.)

[Zhu 2018] Ye Zhu, Kai Ting Ming and Maia Angelova. *A distance scaling method to improve density-based clustering*. In Dinh Phung, Vincent S. Tseng, Geoffrey I. Webb, Bao Ho, Mohadeseh Ganji and Lida Rashidi, editeurs, Advances in Knowledge Discovery and Data Mining, pages 389–400, Cham, 2018. Springer International Publishing. (Cited on page 15.)

A 10-Fold Cross Validation Results on 65 Time Series Datasets

As interpreted in [Antunes 2001], “A sequence composed by a series of nominal symbols from a particular alphabet is usually called a *temporal sequence* and a sequence of continuous, real-valued elements, is known as a *time series*”. According to this definition, trajectory is a specific kind of time series. We test the performance of DHD(ω) based on 65 public time series datasets which are collected by Chen *et al.* [Chen 2015]. The average and standard deviation of classification error ratio on the datasets are given in Table A.1, where the best performance of each dataset is shown in bold. The results show that DHD(ω) outperforms than DHD.

Table A.1: Classification Error Ratio (%) on Time Series Datasets

No.	**Dataset**	**Data Size**	**Clusters Size**	**DHD**		**DHD(ω)**	
				Average	**Std**	**Average**	**Std**
1	50words	905	50	86.52	**0.0300**	**40.77**	0.0391
2	Adiac	781	37	71.33	0.0603	**36.48**	**0.0452**
3	ArrowHead	210	3	50.00	0.0985	**10.00**	**0.0613**
4	Beef	60	5	55.00	**0.2364**	**50.00**	0.2606
5	BeetleFly	40	2	40.00	**0.1748**	**22.50**	0.2486
6	BirdChicken	40	2	22.50	0.3217	**20.00**	**0.1581**
7	Car	120	4	58.33	0.1521	**29.17**	**0.0982**
8	CBF	930	3	60.54	0.0608	**3.44**	**0.0167**
9	Coffee	56	2	25.33	0.1501	**1.67**	**0.0527**
10	Computers	500	2	**26.40**	**0.0610**	37.80	0.0614
11	Cricket_X	780	12	78.08	**0.0293**	**48.33**	0.0897
12	Cricket_Y	780	12	80.51	0.0550	**50.51**	**0.0510**
13	Cricket_Z	780	12	78.33	**0.0333**	**48.08**	0.0397
14	DiatomSizeReduction	322	4	8.39	0.0392	**0.00**	**0.0000**
15	DistalPhalanxOutlineAgeGroup	539	3	33.19	0.0637	**23.38**	**0.0583**
16	DistalPhalanxOutlineCorrect	876	2	34.60	0.0509	**23.29**	**0.0425**
17	DistalPhalanxTW	539	6	43.21	0.0563	**29.68**	**0.0488**
18	Earthquakes	461	2	**32.34**	**0.0735**	36.04	0.0797
19	ECG200	200	2	31.50	0.1180	**11.50**	**0.0914**
20	ECG5000	5000	5	13.70	0.0173	**6.68**	**0.0114**
21	ECGFiveDays	884	2	3.51	0.0125	**0.00**	**0.0000**
22	FaceAll	2247	14	64.49	0.0278	**6.54**	**0.0149**
23	FaceFour	112	4	50.91	0.1621	**17.05**	**0.0911**
24	FacesUCR	2247	14	64.53	0.0337	**6.19**	**0.0135**
25	FISH	350	7	69.43	**0.0687**	**17.43**	0.0888
26	Gun_Point	200	2	39.50	0.1012	**2.00**	**0.0258**
27	Ham	214	2	45.28	0.0894	**27.62**	**0.0831**
28	Haptics	463	5	69.77	**0.0424**	**64.37**	0.0510
29	Herring	128	2	**45.45**	**0.1015**	46.79	0.1380
30	InsectWingbeatSound	2200	11	87.91	**0.0221**	**41.50**	0.0263
31	ItalyPowerDemand	1096	2	32.39	0.0485	**5.75**	**0.0206**
32	LargeKitchenAppliances	750	3	**51.33**	**0.0594**	60.53	0.0623
33	Lighting2	121	2	36.28	0.1149	**35.64**	**0.1137**
34	Lighting7	143	7	**52.43**	**0.0966**	54.48	0.1559
35	Meat	120	3	10.83	0.0883	**7.50**	**0.0730**
36	MedicalImages	1140	10	47.72	0.0310	**25.88**	**0.0262**
37	MiddlePhalanxOutlineAgeGroup	554	3	44.42	0.0928	**31.41**	**0.0662**
38	MiddlePhalanxOutlineCorrect	891	2	39.73	0.0648	**27.84**	**0.0480**
39	MiddlePhalanxTW	553	6	49.19	**0.0419**	**45.94**	0.0434
40	MoteStrain	1272	2	27.51	0.0384	**12.57**	**0.0285**
41	OliveOil	60	4	26.67	0.1610	**16.67**	**0.1361**
42	OSULeaf	442	6	65.16	**0.0443**	**38.71**	0.0926
43	PhalangesOutlinesCorrect	2658	2	37.77	0.0409	**24.08**	**0.0261**
44	Plane	210	7	21.90	0.0784	**2.86**	**0.0246**
45	ProximalPhalanxOutlineAgeGroup	605	3	29.40	**0.0554**	**23.64**	0.0620
46	ProximalPhalanxOutlineCorrect	891	2	30.18	**0.0493**	**18.30**	0.0541
47	ProximalPhalanxTW	605	6	33.88	**0.0560**	**26.42**	0.0824
48	RefrigerationDevices	750	3	**35.73**	**0.0439**	63.60	0.0507
49	ScreenType	750	3	**51.20**	**0.0467**	65.60	0.0474
50	ShapeletSim	200	2	**38.50**	0.1435	47.00	**0.0919**

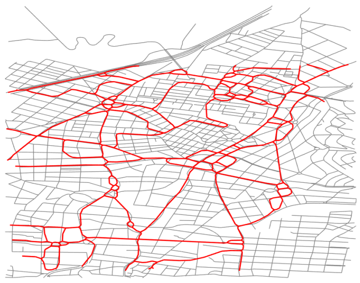
continued on next page

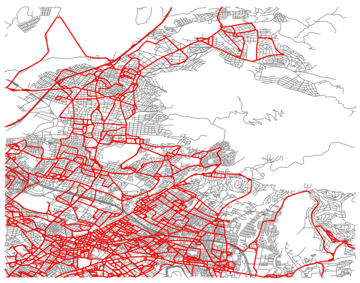
continued from previous page

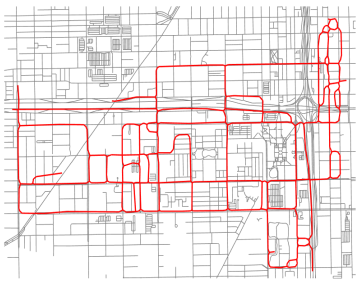
No.	Dataset	Data Size	Clusters Size	DHD		DHD(ω)	
				Average	Std	Average	Std
51	ShapesAll	1198	60	83.48	**0.0225**	**21.37**	0.0302
52	SmallKitchenAppliances	750	3	**49.87**	0.0706	59.33	**0.0587**
53	SonyAIBORobotSurface	621	2	18.69	0.0346	**1.45**	**0.0119**
54	Strawberry	983	2	7.63	0.0183	**4.17**	**0.0170**
55	SwedishLeaf	1122	15	67.21	0.0525	**17.02**	**0.0359**
56	Symbols	1000	6	77.90	0.0370	**4.40**	**0.0222**
57	synthetic_control	600	6	73.50	0.0552	**3.33**	**0.0192**
58	ToeSegmentation1	252	2	30.51	**0.0697**	**26.09**	0.0912
59	ToeSegmentation2	166	2	**19.85**	**0.0784**	21.25	0.1409
60	Trace	200	4	25.00	0.0943	**6.50**	**0.0626**
61	TwoLeadECG	1162	2	8.86	0.0269	**0.95**	**0.011**
62	Wine	111	2	5.45	0.0878	**4.55**	**0.0643**
63	WordsSynonyms	905	25	82.75	**0.0458**	**38.12**	0.0489
64	Worms	225	5	**59.57**	**0.0839**	69.94	0.1074
65	WormsTwoClass	225	2	**40.45**	0.1212	50.61	**0.0490**
Average of all datasets				44.36	0.0744	**26.50**	**0.0640**

B Generated Maps by the Three-Step Framework and by Wang *et al.*

In this section, we show the generated maps by the three-step framework and by Wang *et al.*. Figure B.1 and Figure B.4 plots the generated maps of four urban datasets along with the groundtruth. Besides, for the visual inspection of data coverage, Figure B.2, Figure B.5, Figure B.3and Figure B.6 show the generated maps with trajectory data. In each figure, the maps by Wang *et al.* and by our three-step framework are listed in the first and second columns, respectively.


(a) Athens small


(b) Athens large


(c) Chicago

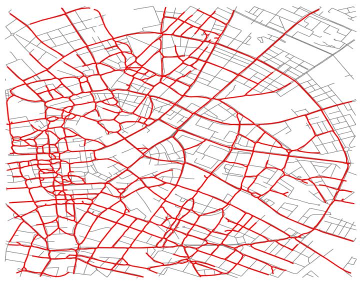
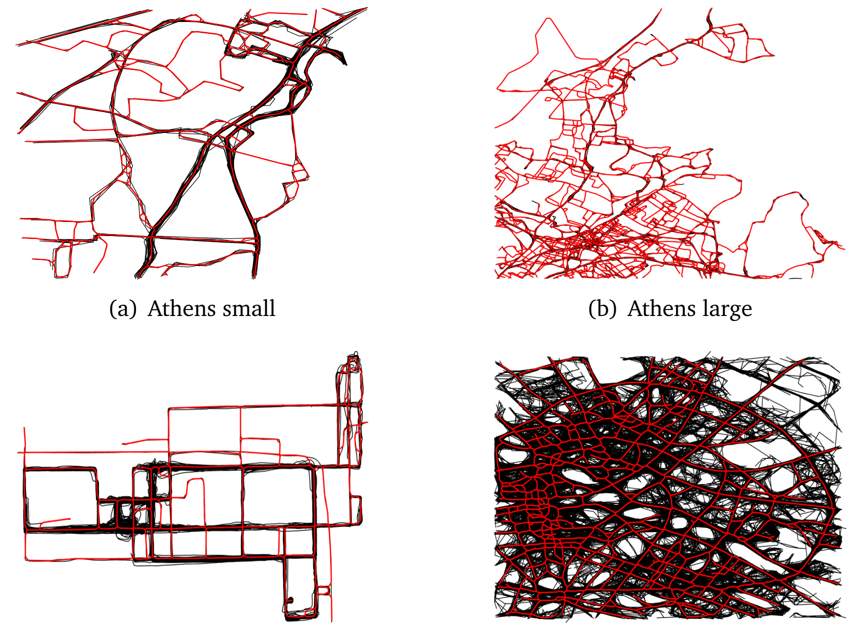
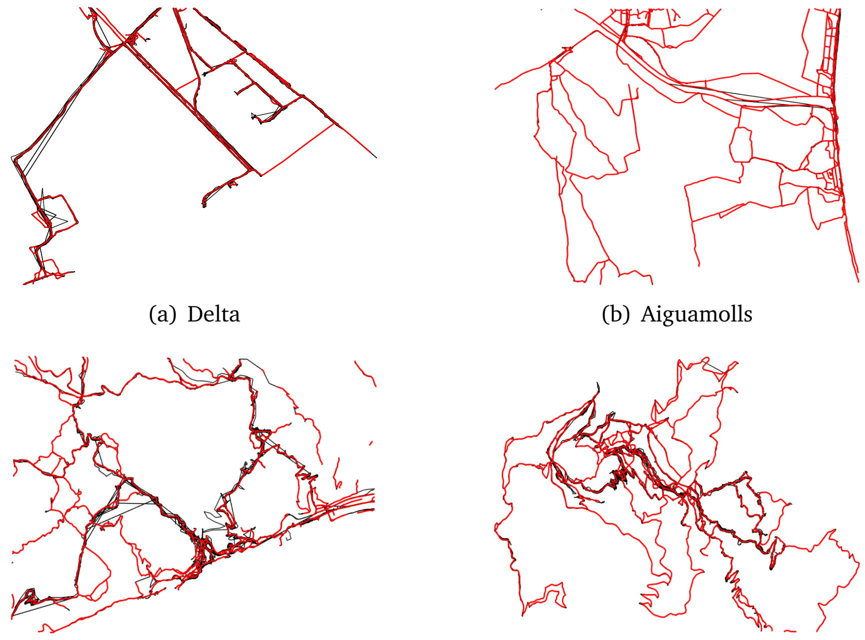

(d) Berlin

Figure B.1: Plots of generated maps (red) on the groundtruth (black) of urban GPS data by our three-step framework.


(a) Athens small (b) Athens large

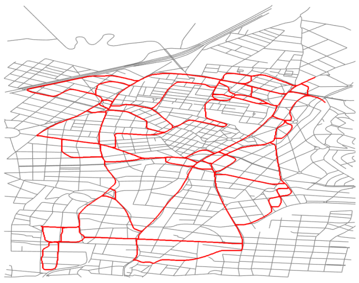
(c) Chicago (d) Berlin

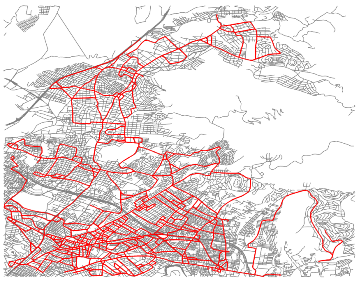
Figure B.2: Plots of generated maps (red) with trajectories (black) of urban data by our three-step framework.

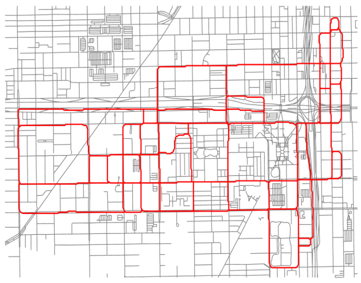

(a) Delta (b) Aiguamolls

(c) Garraf (d) Montseny

Figure B.3: Plots of generated maps (red) with trajectories (black) of hiking data by our three-step framework.


(a) Athens small


(b) Athens large


(c) Chicago

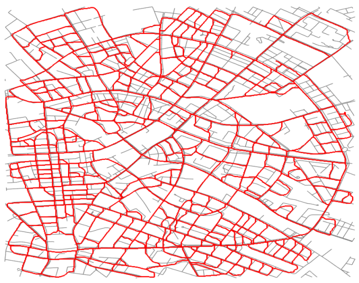
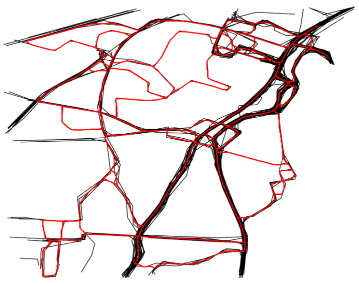
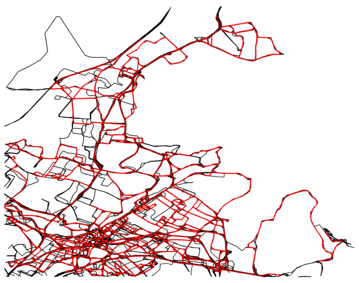
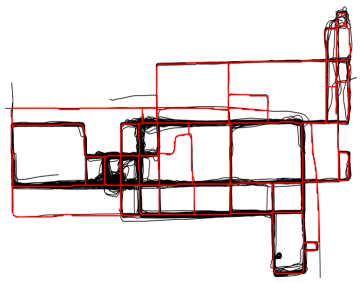

(d) Berlin

Figure B.4: Plots of generated maps (red) on the groundtruth (black) of urban GPS data by Wang *et al.*


(a) Athens small


(b) Athens large


(c) Chicago

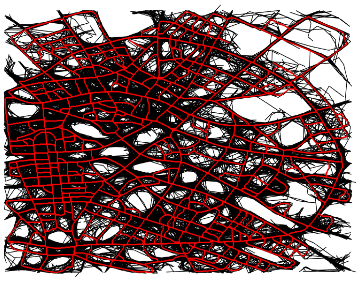

(d) Berlin

Figure B.5: Plots of generated maps (red) with trajectories (black) of urban data by Wang *et al.*

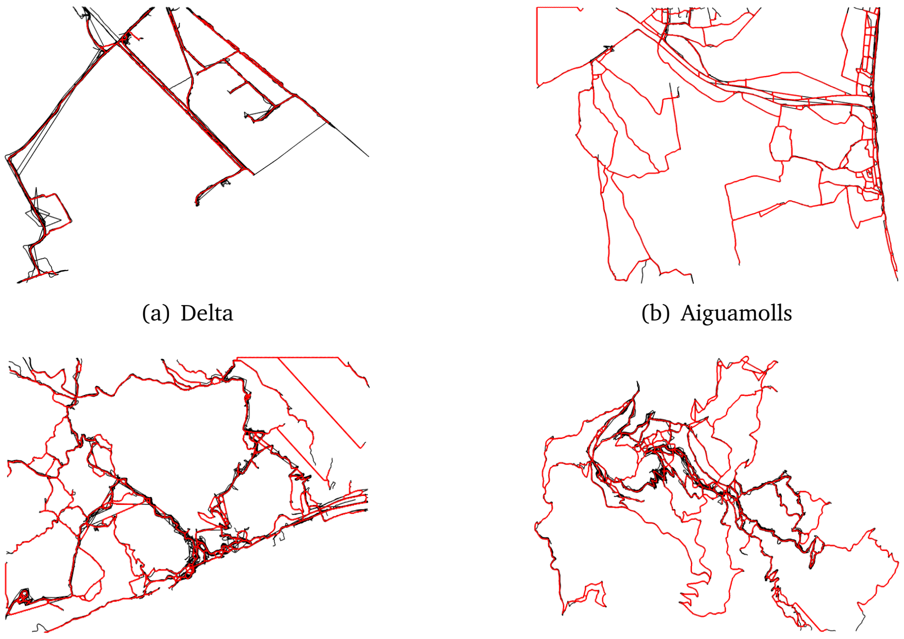

(a) Delta (b) Aiguamolls
(c) Garraf (d) Montseny

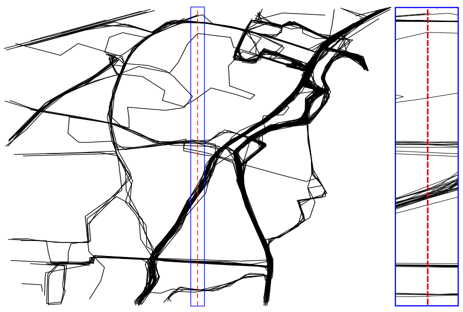
Figure B.6: Plots of generated maps (red) with trajectories (black) of hiking data by Wang *et al.*

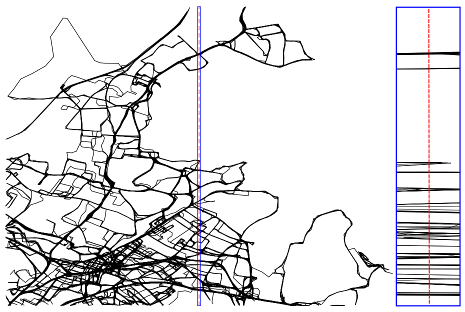
C Map Construction With the Split-and-Merge Strategy

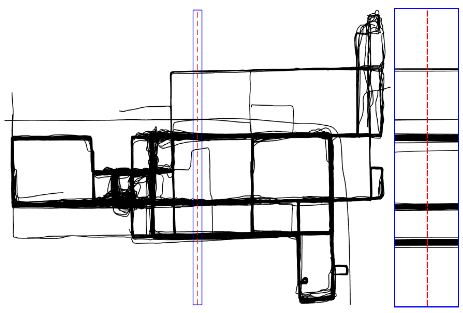
In this section, we show the datasets and generated maps.

C.1 GPS Datasets With Splitting Boundaries

To perform the split-and-merge strategy, we split the datasets into two regions. Figure C.1 and Figure C.2 show the datasets with the spatial splitting. The overlapping zone is highlighted with a blue frame, and the red dotted line indicates the midline.


(a) Athens small


(b) Athens large


(c) Chicago

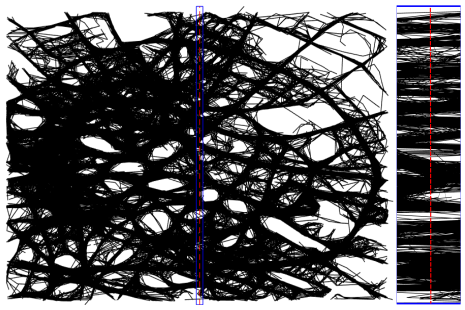
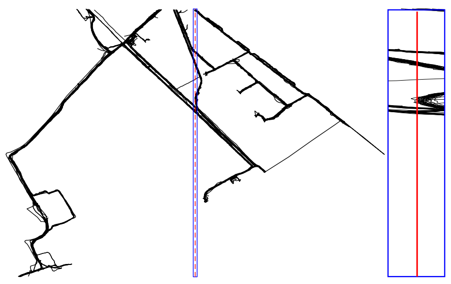
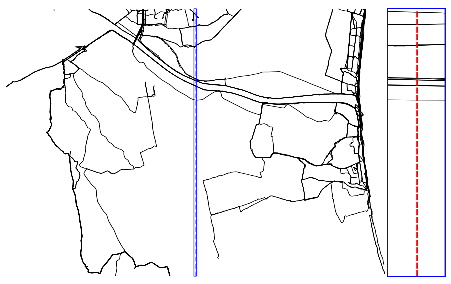
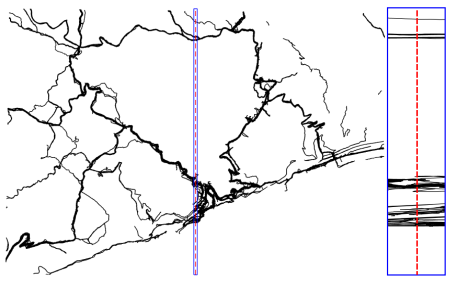

(d) Berlin

Figure C.1: Plots of urban GPS datasets with splitting boundaries and midline. A zoom in view of the overlapping zone is shown in the right side.


(a) Delta


(b) Aiguamolls


(c) Garraf

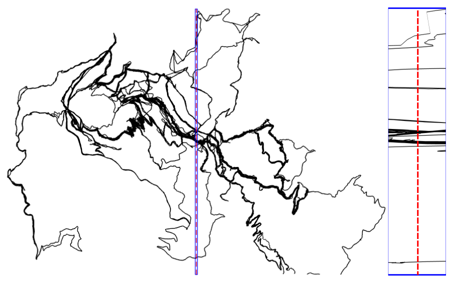
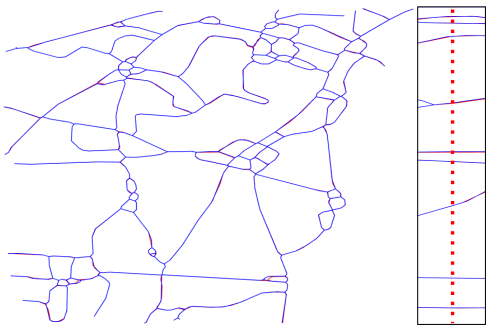
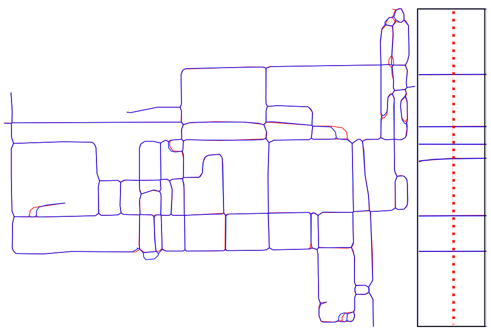

(d) Montseny

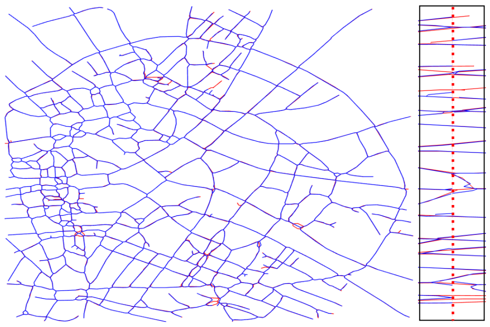
Figure C.2: Plots of four hiking GPS datasets with splitting boundaries and midline. A zoom in view of the overlapping zone is shown in the right side.

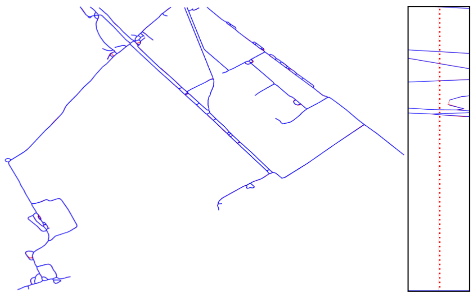
C.2 Generated Maps by the Split-and-Merge strategy

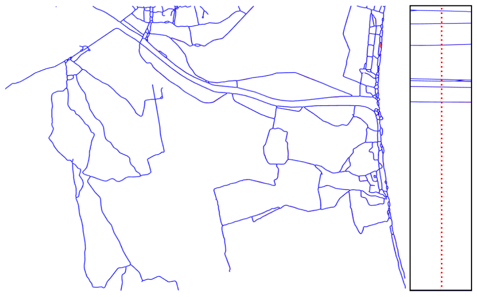
In Figure C.3, we show the generated maps by the three-step-framework with (in blue) and without (in red) using the split-and-merge strategy. A zoom in view of the overlapping zone with the midline in red is shown on the right side.


(a) Athens small


(b) Chicago


(c) Berlin


(d) Delta


(e) Aiguamolls

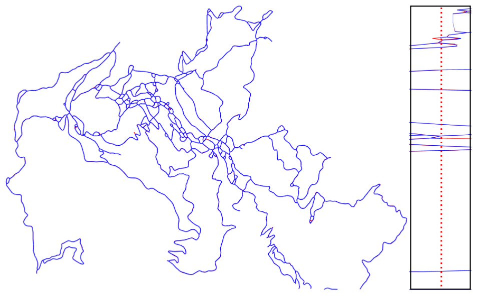
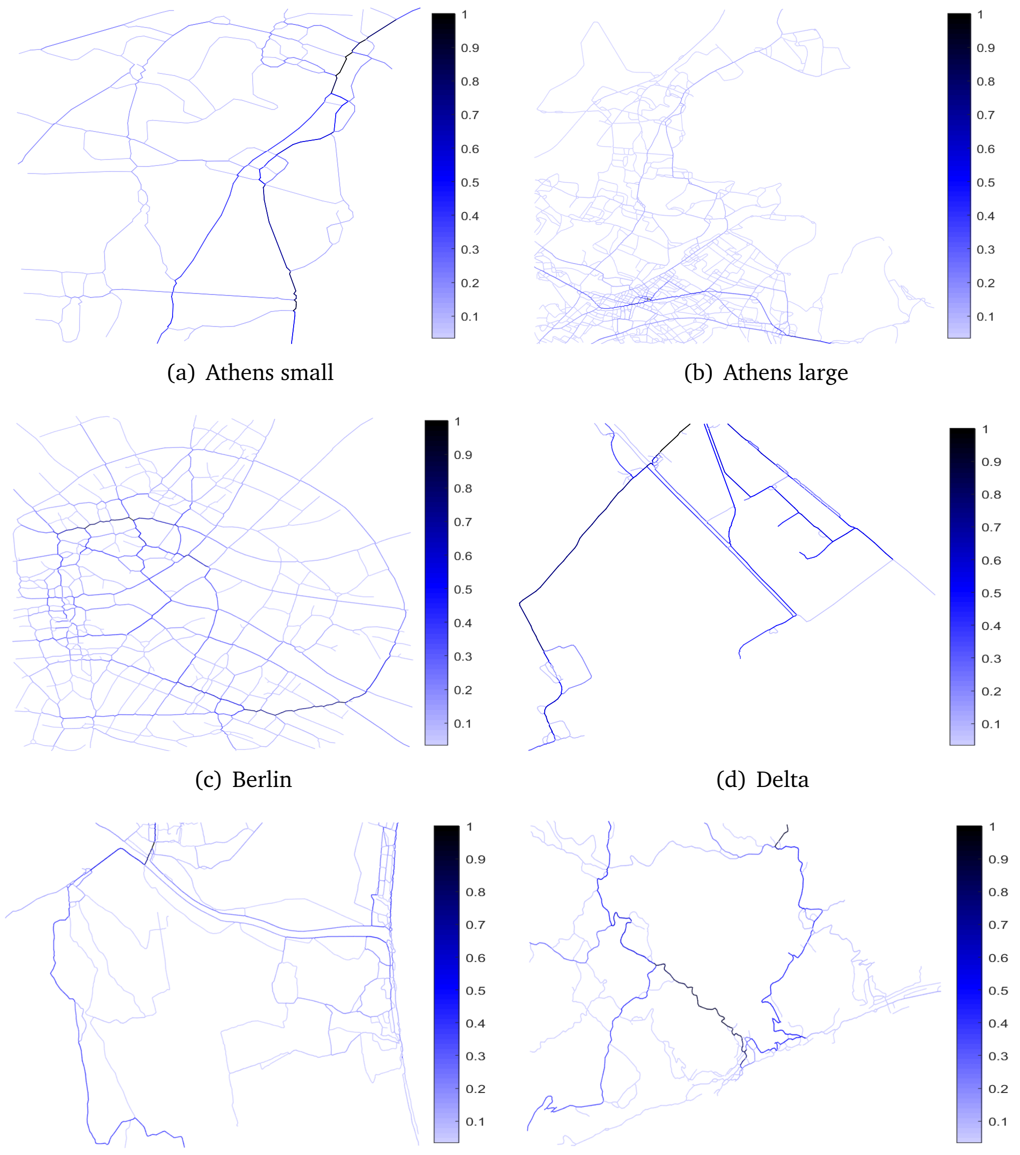

(f) Garraf

Figure C.3: Plots of generated maps utilizing the split-and-merge strategy.

D Map Visualization With Edge Weight

In Figure D.1, we show the filtered maps by edge weight. The frequencies of edges are normalized to assign the colors from blue to black with the increasing frequency.



(a) Athens small (b) Athens large

(c) Berlin (d) Delta

(e) Aiguamolls (f) Garraf

Figure D.1: Plots of filtered maps visualized with colormap.