NOVEL ALGORITHMS FOR TRAJECTORY SEGMENTATION BASED ON INTERPOLATION-BASED CHANGE DETECTION STRATEGIES

by

Mohammad Etemad

Submitted in partial fulfillment of the requirements
for the degree of PhD of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
September 2020

Table of Contents

List of Tables

List of Figures

Abstract

An enormous number of mobility datasets for tracking animals, vehicles, vessels, individuals and moving objects are currently available, and this number continues to grow. Mobility data has diverse applications, including for transportation, marine navigation, tourism, and animal behaviour analysis. Processing such data requires reasonable pre-processing and cleaning efforts, owing to its velocity. Splitting the traces of mobility data movements into semantically related trajectory points is an essential task for trajectory mining pre-processing, called Trajectory Segmentation. Generating enriched graph representations, preserving the privacy of mobility data, and facilitating trajectory-tagging tasks are three critical reasons for the importance of this pre-processing task. Available solutions for trajectory segmentation utilize background knowledge more than just the movement captured knowledge and are specific to a particular domain. They do not provide a domain-independent solution that uses only the movement tracks.

We propose two TS methods that apply to two different types of trajectory applications: 1) when we only have access to geolocation and timestamp, but its point or segment feature is not accessible, and 2) when a dataset with at least a point feature is available.

Sliding Window Segmentation (SWS) splits trajectories into segments using behaviour change detection. We evaluated SWS on three datasets (fishing, hurricanes, and Geolife) by comparing it against four available solutions (SPD, GRASP-UTS, CBSMoT, and WKMeans). SWS discovered statistically significant higher-quality segments (higher Harmonic Mean) than SPD, GRASP-UTS, and WKMeans. The number of identified segments, amount of memory, and CPU consumption of algorithms, and v-measure indicated that SWS found more high-quality segments than CBSMoT. SWS cannot produce segments for a dataset with a low frequency of capturing.

Wise Sliding Window Segmentation (WSII) identifies the potential partitioning position using labelled data, a binary classifier, and a majority vote decision-making mechanism. It can be applied in a trajectory-tagging platform to assist in annotating trajectories. It boosted harmonic means on hurricanes and Geolife datasets compared to five other algorithms; however, WSII does not perform well on the fishing dataset. The dataset attributes and the nature of the movement are two major contributing factors (e.g. low frequency of sampling decreases the quality of segmentation).

List of Abbreviations Used

AIS	Automatic Identification System
CB-SMoT	Clustering Based Stops and Moves of Trajectories
DB-SCAN	Density-Based Spatial Clustering of Applications with Noise
DB-SMoT	Direction Based Stops and Moves of Trajectories
FDA	Functional Data Analysis
GPS	Global Positioning System
GRASP-UTS	Greedy Randomized Adaptive Search Procedure for Unsupervised Trajectory Segmentation
Harmonic Mean	Harmonic mean of Purity and Coverage
I.I.D	Independent and identically distributed
IoT	Internet of Things
MDL	Minimum Description Length
NOAA	National Oceanic and Atmospheric Administration
OPTICS	Ordering Points to Identify Clustering Structure
RGRASP-SemTS	Reactive Greedy Randomized Adaptive Search Procedure for semantic Semi-supervised Trajectory Segmentation
RTR	Robust Time-Referenced
S-AIS	Satellite Automatic Identification System

SeTraStream	Semantic-Aware Trajectory Construction over Streaming Movement Data
SID	Segment Identifier
SPD	Stay Point Detection
SWS	Sliding Window Segmentation
T-OPTICS	Trajectory Ordering Points to Identify Clustering Structure
TRACLUS	Trajectory Clustering
TS	Trajectory Segmentation
WKMeans	Warped K-Means
WSII	Wise Sliding Window Segmentation

Glossary

AISHUB is a FREE AIS data-sharing service which provides access to real time ship positions for vessel tracking systems. 122, 136, 137

Analytic is an active learning system for trajectory classification. 4

Atlantic hurricanes dataset (hurricanes dataset) is an asynchronous dataset with unidentifiable multiple moving object data. It is published by the National Oceanic and Atmospheric Administration (NOAA). 8, 43, 48, 61, 107, 121

Automatic Identification System (AIS) is a broadcast system utilized for vessels navigation with 27 defined report message that updates as often as every two seconds. 1, 18, 38, 48, 61, 137

Coverage is an evaluation metric for trajectory segmentation task. 37, 54, 68, 107

Cubic Method is a method that uses a third degree polynomial to interpolate or extrapolate values. 18

Extrapolation is an approach in which we construct new trajectory points outside of the range of a discrete set of known trajectory points. 15, 59

Fishing Dataset is a multi-object and asynchronous dataset that was compiled in contribution to the project called "Programa Nacional de Rastreamento de Embarcações Pesqueiras por Satelite" (PRES) in Brazil as part of the federal government program to supervise fishing activities on the Brazilian coast. 41, 48, 73, 107, 121

Fully Labeled Dataset is an attribute of a trajectory dataset that contains the behavior of a moving object. 37

Geolife Dataset is a frequently used benchmark dataset for mobility data research, is a synchronous dataset with identifiable multi objects, Microsoft Research Asia collected it from April 2007 to October 2011. 40, 48, 77, 107, 121

Geolocation is a tuple includes the latitude and longitude of the moving object. 2, 18, 59, 120

HURDAT2_processor is a Python script that performs the pre-processing necessary for using the hurricanes datasets published by NOAA. 8

Identified Trajectories Dataset is a type of trajectory dataset that includes segment features that identify the segmentation from a subject matter expert view. 37

Interpolation is an approach in which we construct new trajectory points within the range of a discrete set of known trajectory points. 5, 15, 31, 59, 100, 121

Interquartile range is the difference between 75th and 25th percentiles. 74

K-anonymity is a privacy model regularly employed to preserve data privacy in data sharing scenarios. 3

Kinematic Extrapolation is an extrapolation technique based on the dynamics of movement such as velocity and acceleration. 16

Label is a specific type of segment features. This may be called Segment Label. 4, 13, 19, 29, 48, 68, 99

Linear Interpolation is the simplest interpolation technique that using the average value between two value for interpolation of mid value. 15, 70, 112

Linear Regression is a linear method to represent the relationship between a dependent and an independent variable. 16, 70, 121

Non-overlapping segment is the situation where two consecutive segment in a trajectory do NOT have any overlapping trajectory points with similar semantic. 11

Over-segmentation is a case in which a trajectory segmentation algorithm produces considerably too many short segments. 56

Overlapping segment is the situation where two consecutive segment in a trajectory can have some overlapping trajectory points with similar semantic. 38

Partitioning Position is the index of a trajectory point that is positioned at the boundary of two segments. 5, 11, 12, 49, 97, 99, 121

Point Feature is a measured value, assigned to a trajectory point. A formal definition of a point feature is provided in Section 2.1. 12, 84

Point-based View is an approach to handle and process mobility data where all the points are stored in a media, and processing happens at the single point level. 3

Purity is an evaluation metric for trajectory segmentation task. 37, 54, 68, 107

Random Walk is an extrapolation/interpolation method for cases in which the moving object behaves randomly. it uses the recent distribution of distance and bearing of the movement for its interpolation/ extrapolation. 17

Raw trajectory is used interchangeably to trajectory. 10

Segment is a set of consecutive trajectory points belonging to a raw trajectory. 11

Segment Feature is a calculated or semantically assigned value that is assigned to a segment. 12

Segment Label is a specific type of segment features. 13

Segment Length is the number of trajectory points in a segment. 11

Segment-based View is an approaches to handle and process mobility data where the summarized version of data is stored in the media to be processed. 3

Semantic Segment is a tuple that assigns a set of segment features to a segment. A formal definition of a semantic segment is provided in Section 2.1. 15

Semantic Trajectory is a trajectory that all of its segments are *semantic segments* and these semantic segments have the same set of segment features. 15, 23, 31

Sub-Trajectory or segment is a set of consecutive trajectory points belonging to a raw trajectory. 11

Timestamp is the component that captures the temporal aspect of a trajectory point. 18, 19

Trajectory is a time-ordered sequence of trajectory points of a moving object. A formal definition of a raw trajectory is provided in Section 2.1. 10

Trajectory point is a data structure that contains the location of an object and the time of capturing the location. A formal definition is provided in Section 2.1. 10

Trajectory segmentation is a process for dividing a trajectory into sub parts. 11

Trajlib is a public Python library that aims to facilitate trajectory mining by generating segment and point features for trajectory data. 8

TrajSeg is a public library that includes our proposed trajectory segmentation algorithms. This library also includes all the other trajectory segmentation algorithms that we applied in our experiments. 8

Under-Segmentation is a case where a trajectory segmentation algorithm is producing a considerably lesser number of segments than expected. 56, 90

Unlabeled Dataset is a trajectory dataset devoid of any label data. 37

VISTA is a visual analytic platform for annotating trajectories. 4, 7, 22, 99, 115, 122, 143

Chapter 1

Introduction

The number of mobility datasets available for research and the applications of mobility datasets increase every moment. Some instances of these datasets include tracking animals, ships, cars, people, and moving IoT devices. Mobility data has all the characteristics of big data. The volume of this type of data is immense. For example, an Automatic Identification System (AIS) dataset for a year on Marine Traffic[1] contains 188,000,000,000 records of positioning reports. The velocity of the production of this type of data is very high so that a moving object can produce thousands of records in less than a day. Depending on the frequency of capturing data, it can be even more. A variety of data types can be merged to mine knowledge from such data stores. For instance, we have geographic shapefiles, tabular data, and graph representations of mobility data. The veracity of mobility data may sometimes be low because we have different types of noise and errors in data, such as GPS jumps or gaps. Therefore, to mine knowledge from this data requires the use of subject matter expert approaches.

Marine Traffic reported at almost 520,000,000 position reports per day. AIS messages[2], are captured from close to 3,000 active AIS stations around the globe [50]. Movebank[3], a free online database of animal tracking, reported 2.2 billion locations, 3.2 billion non-location events, with over 5,000 data owners that are participating in 7,320 studies [55]. According to the World Bank[4] dataset in 2017 for each group of 100 individuals', the number of mobile cellular subscriptions was 104.49. According to Market and market (MnM) report in 2018, the Smart Transportation market size is estimated near 75 Billion USD that is predicted to grow to 149.21 billion USD in 2023.

[1]Marine Traffic was a project started as an open, community-based project. This project formed a company called Marine Traffic (Founded 2007) which provides real-time information on the movements of ships and the current location of ships in harbours and ports. Their website is https://www.marinetraffic.com

[2]https://en.wikipedia.org/wiki/Automatic_identification_system

[3]https://www.movebank.org

[4]https://data.worldbank.org/

Mobility data has three extra dimensions (object identifier, time and geolocation) compared to conventional data, and it has properties such as *heterogeneity* and *auto-correlation*, making data mining tasks more challenging. The first property — heterogeneity — means that a moving object experiences different circumstances, such as low traffic or congestion: for example, when a vehicle moves from a rural area with little traffic to a downtown area with traffic congestion. The second property is auto-correlation, which means the location of a moving object is highly related to its location in nearby time and space. This is also in line with the first law of geography, which says that "everything is related to everything else, but near things are more related than distant things."[75]. The different types of correlation (e.g. temporal or spatial) makes an underlying assumption (i.e., Independent and identically distributed (I.I.D)) of most of the machine learning approaches invalid.

Processing mobility data, such as traces of individuals, vehicles, ships, and animals, has been the focus of researchers in science and industry. These traces of moving objects are called trajectory data and can be informally described as a time-ordered sequence of the geolocations of a moving object. Transportation mode detection [25], fishing detection [16], crime prediction [6], tourism [27], environmental science [74], and traffic dynamics [12, 17, 66] are a few examples of fields where trajectory mining techniques can be employed.

Splitting a trajectory into smaller parts is a primary task, and it is called trajectory segmentation. This partitioning is necessary because a mobility pattern, in general, is not the same for the entire trajectory, but it could be for some of its sub-parts (segments or sub-trajectories). Therefore, the segmentation process becomes one of the most critical pre-processing steps for trajectory data mining. The major benefit of trajectory segmentation is to decrease the effects of the presence of different types of correlation in mobility data, which makes the I.I.D assumption invalid. After a segmentation procedure, trajectory points in each segment are more related to each other and could be relatively unrelated to their segment neighbors.

Data summarization and data privacy are other key reasons to perform trajectory segmentation. For example, a navigation graph, including nodes (stop points) and edges (trajectory segments) which represents the voyages of a vessel, can be extracted from a trajectory dataset after a segmentation procedure. The segmentation identifies the nodes, their properties, and properties of each edge. From a privacy perspective, sharing the features of an edge disseminates less information than sharing the time and location of moving object during its movement. Also, the object itself may want to remain private: e.g. a fishing ship does not want the world to know where it goes fishing. Therefore, it is desired that their identity stays private when the data is analyzed. Since the periodic behavior of moving objects can represent some patterns, a trajectory can act as a fingerprint for a moving object to reveal its identity. A trajectory of a moving object can be attacked by a hacker to reveal its identity and related information can be discovered by applying background datasets [52]. Utilizing a trajectory segmentation method can protect the privacy of a moving object since the moving object only shares a summarized version of its data [1]. Many individuals' movements share the same patterns after segmentation, or they generate similar trajectories. They are thereby providing a form of anonymization, similar to K-anonymity [64].

There are two different approaches to handle and process mobility data. The first approach is a *point-based view* in which all the points are stored in a medium, and processing happens at the single point level. Usually, the number of points grows limitlessly and feeding this kind of data into an algorithm, even an algorithm with linear complexity can cause issues such as memory capacity limits and lengthy processing requirements. The second approach is a *segment-based view* in which the summarized version of data is stored in the media to be processed. This approach enables many methods to work reasonably well on this kind of data because it overcomes the complexity of storage and processing of a point-based view structure. A segment-based view provides knowledge more clearly related to a segment of a trajectory rather than a point-based view approach, which provides information about every single point of a trajectory. The researchers in the trajectory mining field are usually more interested in the behavior of a moving object through time rather than knowing the location of a moving object at every moment. Therefore, providing an aggregated representation of a moving object's behavior can be more beneficial to answer complex queries. For example, using this aggregated representation, the user can build a graph representation, defined in Section 2.1, of mobility data.

The segment-based view can be facilitated by a trajectory segmentation task, which is the process of splitting a given trajectory into several homogeneous segments according to some criteria. This task plays a pivotal role in trajectory mining since it affects the features of each segment, as the features may depend on the size of the trajectory segment, independently of the application domain, such as fishing detection [69], animal behavior [39, 38], tourism [27], traffic dynamics [25, 38, 78, 67], and vessel movement patterns [11]. There are several approaches to perform the trajectory segmentation task such as Clustering Based Stops and Moves of Trajectories (CB-SMoT) [58], Stay Point Detection (SPD) [85], Warped K-Means (WKMeans) [43], Greedy Randomized Adaptive Search Procedure for Unsupervised Trajectory Segmentation (GRASP-UTS) [69], and Trajectory Clustering (TRACLUS) [41].

While reviewing the related work, the following gaps were discovered: First, most of the segmentation algorithms are domain or application specific. For example, some of them are designed to process multi-sensor trajectory datasets. These methods are not applicable to single trajectories. Some algorithms require knowledge more than just the time and location of moving object to segment data. Finally, most of them suffer from a high complexity for processing a huge volume of data. This work has the main objective of searching for a segmentation algorithm that can segment trajectory data with low complexity while using only location and time of the moving object. As a requirement, such algorithm must not depend on the knowledge that can be extracted from collective behaviour of multiple trajectories. The benefit of proposing such an algorithm is that it can be executed on a moving object in which there is limitation of memory, connectivity, and processing power. Second, most proposed algorithms for trajectory segmentation are unsupervised or semi-supervised—no trajectory segmentation algorithm fully benefits from labelled data. A supervised trajectory segmentation may facilitate the process of trajectory annotation. Such an algorithm can gradually learn from subject matter experts and tune its performance based on the labelled data provided by users. A supervised approach can receive training data and be adjusted to a specific domain and application. Although there is a lack of training data with labels, trajectories can be annotated using platforms such as VISTA [68] or Analytic [38]. Another benefit of a supervised approach is that it can be tuned to a particular performance measure considered by an application or domain. Therefore, this work was guided by the following research questions:

- How could positions be detected in trajectory data where there were abrupt changes in its behavior the using minimum possible information (time and location of a single moving object)?
- How can these abrupt changes be used to segment trajectory data? The abrupt changes may be discovered using a predictor like an interpolation technique.
- In case of trajectory annotation, how could we detect partitioning positions in trajectory data applying previously annotated data collected in a training dataset in that domain?
- How would a new strategy based on those assumptions perform in well-known trajectory datasets?

1.1 Contributions

The contributions of this thesis are a supervised trajectory segmentation algorithm and an unsupervised trajectory segmentation algorithm using a sliding window to process trajectories. The effectiveness and computational performance of the unsupervised algorithm are evaluated against three data sets from three different domains (the fishing dataset, Atlantic hurricanes dataset, and Geolife dataset), four competitors segmentation algorithms (SPD, CBSMoT, GRASP-UTS, WKMeans), and under seven metrics (purity, coverage, harmonic means, memory consumption, CPU usage, number of segments, and v-measure). The results confirm that the algorithm outperforms competitor algorithms, even though GRASP-UTS and WKMeans had access to extra information. GRASP-UTS benefits from the wind speed information, and WKMeans is provided with an estimate of the number of segments. Moreover, the number of parameters to tune the supervised method is lower than the number of parameters for CBSMOT, GRASP-UTS and SPD. The results of parameter tuning for each fold of our trajectories indicated that the competitors' variation of parameter values is high. The proposed method had a very low variation of parameter values for each fold of trajectory data that shows the algorithm's robustness. Further investigation of the results showed that the unsupervised algorithm has a weakness in segment trajectories with multiple behaviour changes inside a movement course.

The effectiveness and computational performance of the supervised algorithm are evaluated against three data sets from three different domains (the fishing dataset, Atlantic hurricanes dataset, and Geolife dataset), five competitors segmentation algorithms (SWS, SPD, CBSMoT, GRASP-UTS, WKMeans), and under seven metrics (purity, coverage, harmonic means, memory consumption, CPU usage, number of segments, and v-measure). The supervised trajectory segmentation algorithm results demonstrate that the algorithm is at least competitive, and it outperforms competitor algorithms in many cases. Despite this algorithm's weakness in segmenting the fishing dataset, it surpassed competitors in the Atlantic hurricanes and Geolife datasets. The algorithm's further evaluation identified that the quality of capturing trajectories plays a vital role in the quality of segmentation.

Purity and Coverage are conventionally applied to measure the quality of trajectory segmentation algorithms. A trajectory segmentation algorithm that encourages short segments increases Purity, and a trajectory segmentation algorithm that promotes long segments increases Coverage. We introduced harmonic means of Purity and Coverage as a single measure representing the quality of segmentation tasks. At the end of chapter 6, we provide a guideline for the end-user that guides which trajectory segmentation algorithm is proper considering factors such as type of data, availability of extra information, and the rate of capturing trajectory points. Here we discussed the default values for our proposed approach and the best configuration for each experimented domain.

1.2 Outcomes

The outcomes of this study are two sliding window approaches and software packages to facilitate the trajectory segmentation of a moving object. The following are the publications which are the direct outcome of this work:

- The idea of interpolation and calculating the divergence of a moving object from an ordinary interpolation in that domain is explored in a conference paper titled "A Trajectory Segmentation Algorithm Based on Interpolation-based Change Detection Strategies." [22]. The primary sliding window approach that is carried out in this work is presented in a journal paper titled "SWS: An unsupervised trajectory segmentation algorithm based on change detection with interpolation kernels" [23].

- The supervised sliding window approach utilizes a binary classifier combined with a majority vote mechanism in order to place the partitioning position in the best position to generate high quality segments. We published the details of this work in a conference paper called "Wise Sliding Window Segmentation: A classification-aided approach for trajectory segmentation"[21].

During our study we explored some applications of trajectory segmentation and trajectory mining that are indirectly related to our work. The following publications are the works that are indirectly related to our study.

- "Using Deep Reinforcement Learning Methods for Autonomous Vessels in 2D Environments" is a conference paper in which we explore the possibility of generating a trajectory from moving object location to a target location [26].

- "A Network Abstraction of Multi-vessel Trajectory Data for Detecting Anomalies." is a conference paper with a focus on extracting a graph representation from trajectory data [78].

- "VISTA: A visual analytics platform for semantic annotation of trajectories." is a demo conference paper about an application called VISTA (see Section C.1) to segment and label trajectory data [68].

- "On feature selection and evaluation of transportation mode prediction strategies" is a paper that reviews segment and point features of trajectory data in transportation domain to select the most informative features [24].

- "Predicting transportation modes of GPS trajectories using feature engineering and noise removal" is a conference paper in which we introduced our TrajLib library. This paper received a positive attention from the research community because of the availability of the Python library [25].

- "Uncovering vessel movement patterns from AIS data with graph evolution analysis" is another conference paper that focuses on extracting a graph representation from trajectory data [11].

- "Building navigation networks from multi-vessel trajectory data" is a journal paper published in Geoinformatica that aims to extract graph representation from trajectory data.

During this research, we implemented two public Python libraries, a pre-processing script and a private Python repository as follows:

- Trajlib is a public Python library that aims to facilitate trajectory mining by generating segment and point features for trajectory data.[5]
- TrajSeg is a public library that contains our proposed trajectory segmentation algorithms. This library also contains all the other trajectory segmentation algorithms that we applied in our experiments[6].
- HURDAT2_processor is a Python script that does the preprocessing necessary for using the Atlantic hurricanes dataset. This code is available on Github[7].
- We developed a private repository to capture and collect AIS data. This private repository, available at CS GitLab, contains all the code for capturing, decoding and storing real-time AIS data [8].

Implementation of the last item resulted in creating the following datasets:

- A private AIS dataset which contains more than 10B AIS messages, accessible on MongoDB on a dedicated server on Dalhousie Datacenter named Bigdata4.
- A small AIS dataset that was extracted from the above dataset and made available to the general public for research. The details of this dataset are available in Chapter 4.

1.3 Outline

The overall organization and structure of the thesis are as follows. Chapter 2 explores the background, including definitions, models and interpolations. Chapter 3 reviews

[5]https://github.com/metemaad/TrajLib
[6]https://github.com/metemaad/TrajSeg
[7]https://github.com/metemaad/HURDAT2_processor
[8]https://git.cs.dal.ca/big-data-institute/ais-codebase

related work. It includes existing algorithms and methods for trajectory segmentation, plus some essential aspects of these algorithms. We provide details on data structure and datasets applied in this research in Chapter 4. In Chapter 5, we propose the Sliding Window Segmentation (SWS) algorithm, which is our first approach to solve the trajectory segmentation task and the preliminary concepts related to it. Here we provide the experiments and evaluations of this algorithm. We propose the Wise Sliding Window Segmentation (WSII), a classification-aided approach for trajectory segmentation, in Chapter 6. Furthermore, this chapter presents the experiments and evaluation of this algorithm. In Chapter 7, we provide a conclusion and review of potential future work to explore the trajectory segmentation task.

Chapter 2

Background

In this chapter, we discuss the background necessary to understand all the concepts related to the trajectory segmentation task, such as raw trajectories, trajectory point, segments, features, and partitioning positions.

2.1 Definitions

In this section, we formally define some key concepts that are essential for the trajectory segmentation task.

Trajectory Point

A *trajectory point*, l_i^o, is the location of object o at time i, and is defined as,

$$l_i^o = \langle x_i^o, y_i^o \rangle \tag{2.1}$$

where x_i^o is the longitude of the location which varies from $0°$ to $\pm 180°$, while y_i^o is the latitude which varies from $0°$ to $\pm 90°$.

Raw Trajectory

A *raw trajectory*, or simply *trajectory*, is a time-ordered sequence of trajectory points of some moving object o,

$$\tau^o = \langle l_0^o, l_1^o, .., l_n^o \rangle \tag{2.2}$$

Without loss of generality, when it is obvious from the context that we are referring to data from the trajectory of a single object, we may drop the superscript denoting the object ID to simplify our notation. So for instance, we may refer to a trajectory point as $l_i = \langle x_i, y_i \rangle$ instead of $l_i^o = \langle x_i^o, y_i^o \rangle$; or to a trajectory as $\tau = \langle l_0, l_1, ..., l_n \rangle$.

Segment or Sub-trajectory

A segment or sub-trajectory is a set of consecutive trajectory points belonging to a raw trajectory $\tau^o = \langle l_0^o, l_1^o, .., l_n^o \rangle$,

$$s^o = \langle l_j^o, \cdots, l_k^o \rangle, \ \ j \geq 0, \ k \leq n \text{ and } \ s^o \ is \ a \ subsequence \ of \ \tau^o \tag{2.3}$$

The process of generating segments from a trajectory is called *Trajectory Segmentation (TS)*. The most common way of defining TS involves splitting a raw trajectory into a set of non-overlapping segments. More formally:

Trajectory Segmentation

Given a raw trajectory $\tau^o = \langle l_0^o, l_1^o, .., l_n^o \rangle$, we define a sequence of segments $S^o = \langle s_0^o, \cdots, s_k^o \rangle$, such that

$$\forall_{s_i^o, s_{i+1}^o \in S} \ s_i^o = \langle l_p^o, \cdots, l_{p+t}^o \rangle, \ s_{i+1}^o = \langle l_{p+t+1}^o, \cdots, l_{p+t+u}^o \rangle \tag{2.4}$$

and

$$s_0^o = \langle l_0^o, \cdots, l_i^o \rangle, \ s_k^o = \langle l_j^o, \cdots, l_n^o \rangle \tag{2.5}$$

The input and output of the trajectory segmentation process are shown in Equation 2.6, where τ is a raw trajectory which contains n trajectory points, and S is the set of all segments generated from τ using TS.

$$TS : \tau \longrightarrow S \ \ , |\tau| = n + 1, \ \ |S| = k + 1 \tag{2.6}$$

In this notation, $n + 1$ is the number of trajectory points and $k + 1$ is the number of segments resulting from applying TS to the trajectory.

We designate a trajectory point at the end of each segment as *partitioning position*. This means that the result of applying TS to a trajectory, S contains k partitioning positions. We define *segment length* as the number of trajectory points in a segment. For example, $N_k^o = |S_k^o| = n - j$. To simplify for the case of single moving object we write $N_k = |S_k| = n - j$.

Partitioning Position

Let q be a partitioning position index and τ_1 be a raw trajectory generated by a single object. Then, $s_1 = \langle l_0, l_1, ..., l_q \rangle$ and $s_2 = \langle l_{q+1}, l_{q+2}, ..., l_n \rangle$ are two segments derived from τ_1. The set $S = \{s_1, s_2\}$ is the set of all generated segments and $k = |S| = 2$ is the number of elements of this set, which is the number of segments.

Graph Representation

We define l_i as a *boundary point of a segment* if l_i is the first or last trajectory point in that segment. Boundary points of a trajectory is a set of trajectory points that contains boundary points of all segments (B_{τ_n}).

We define a graph representation for a raw trajectory as an example application for a trajectory segmentation task. Each application can customize the graph representation.

Using a raw trajectory τ_n, we extract a set of segments, S^o, by applying a trajectory segmentation task. A graph representation of τ_n is a directed graph, $G = (V, E)$, where V is a set of clusters produced by clustering the boundary points of the raw trajectory, B_{τ_n}, and $E = V \times V$ is a set of edges so that each edge $e_i = (v_i, v_j)$ is a tuple that represents a segment in S^o starting from v_i and ending in v_j.

Point Features

A *point feature* is (i) a **measured value**, P_i, (ii) assigned to a trajectory point. The set of measurement functions is M which contains all of the available measurements and $M_p \in M$.

$$PF_p : L \longrightarrow \mathbb{R}, P_i = M_p(l_i), l_i \in L \tag{2.7}$$

For example, the instantaneous speed is considered a point feature since we can measure the speed of a moving object at each trajectory point.

Segment Features

A *segment feature* is (i) a **calculated** or (ii) **semantically assigned** value, Q_j , assigned to a segment. $SF_j \in SF$ is a function that computes the value of a segment feature. Equation 2.8 shows the domain and range of segment feature function where S is the set of all segments.

$$SF_j : S \longrightarrow Q_j, Q_j \in R \tag{2.8}$$

For example, the median, average, minimum or maximum value of any point feature in a segment can be a segment feature. In some trajectory mining tasks, we could have one or more segment features considered as a target value to be predicted by a model. We call these specific types of segment features as Segment Labels. A *segment label* , or label, $\lambda_i \in \Lambda_L$ is an annotation given to a segment assigned by an expert user in the studied domain, where Λ_L is the set of labels and L is the number of unique labels in Λ_L. We remark that any segment feature can be expanded as a point feature and we use the expanded version of label for ground truth. Equation 2.9 shows an example set of trajectory labels in the transportation domain.

$$\lambda_i \in \{Bus, Walk, Train, Bike\}, L = 4 \tag{2.9}$$

Similarly, we define a segment identifier as a special type of segment feature. A *segment identifier* or *SID*, $\psi_i \in \Psi_v$ is an annotation given to a segment by a subject matter expert, where Ψ_v is the set of segment identifiers and v is the number of unique segment identifiers in Ψ_v. We remark that any segment feature can be expanded as a point feature and we use the expanded version of segment identifier for ground truth. Equation 2.10 represents an example set of segment identifiers in the transportation domain.

$$\psi_i \in \{1001, 1002, 1003, 1004, 1005\}, v = 5 \tag{2.10}$$

Purity

Purity is the rate at which the discovered segments purely represent a semantic label for each ground truth segment. It is computed using Equation 2.11.

$$P(S, \Lambda_L) = \frac{1}{k}\left(\sum_{i=1}^{k} \underset{j\in[1,L]}{\operatorname{argmax}}\left(\frac{N_{ij}}{N_i}\right)\right) \tag{2.11}$$

where S is the set of segments discovered by segmentation algorithm,

Λ_L is the set of labels (a point feature) provided by subject matter expert,

k is the number of discovered segments,

L is the number of expert labels,

N_{ij} is the number of trajectory points inside segment s_i with label λ_j,

and N_i is the total number of points found for the segment s_i.

Coverage

Coverage is the rate at which the discovered segment covers the ground truth segment suggested by a subject matter expert. It is calculated using Equation 2.12.

$$C(S, \Psi_v) = \frac{1}{v}(\sum_{i=1}^{v} \underset{j\in[1,k]}{\text{argmax}}(\frac{N_{\psi_i \cap s_j}}{N_i})) \tag{2.12}$$

where S is the set of segments discovered by segmentation algorithm,

Ψ_v is the set of segments by a subject matter expert,

$N_{\psi_i \cap s_j}$ is the number of trajectory points of the segment s_j that belongs to the ψ_i segment,

and N_i is the total number of points of the identified segment with segment identifier equals to ψ_i.

Harmonic Mean of Purity and Coverage

Harmonic mean is one of several kinds of average mostly applied for situations when the average of rates is desired. The harmonic mean of two numbers a and b is equal to $(2 * ab)/(a + b)$. An example of this measure in machine learning is F_measure, which is the harmonic mean of precision and recall.

The Harmonic mean of Purity and Coverage is derived from purity and coverage, computed using Equation 2.13. Since we are interested in achieving both high average purity and average coverage, we defined harmonic mean of them.

$$H(S, \Lambda_L, \Psi_v) = \frac{2 * P(S, \Lambda_L) * C(S, \Psi_v)}{P(S, \Lambda_L) + C(S, \Psi_v)} \tag{2.13}$$

where S is the set of segments discovered by segmentation algorithm,

Λ_L is the set of labels provided by subject matter expert,

Ψ_v is the set of segments by subject matter expert,

P is the purity,

and C is the coverage.

Semantic Trajectory

A segment can be enriched by assigning a set of segment features to it. $(S_j, \mathbf{Q_j})$ is a *semantic segment* , where S_j is a segment in S and $\mathbf{Q_j}$ is a set of segment features assigned to S_j. A *semantic trajectory* is a trajectory that all of its segments are *semantic segments* and these semantic segments have the same set of *segment features.*

2.2 Interpolation and Extrapolation Methods

Since the main focus of this thesis is to propose trajectory segmentation methods based on interpolation / extrapolation methods, we review interpolation and extrapolation techniques. *Interpolation* is an approach in which we construct new trajectory points within the range of a discrete set of known trajectory points. *extrapolation* is an approach in which we construct new trajectory points outside of the range of a discrete set of known trajectory points. In this work, we use interpolation for constructing a point within the range of known trajectory points to measure the deviation from the norm. Also, we use extrapolation to predict the next point outside of the range of known trajectory points to use it as an auxiliary point to calculate interpolation.

In this section, we review one basic interpolation technique, Linear Interpolation, and four basic extrapolation techniques, namely Linear Regression, Kinematic, Random walk, and Cubic; however, there are many more techniques and enhanced versions of them that can be used such as enhanced kinematic approach [35, 48], advanced random walk [72], and Functional Data Analysis (FDA) [36, 60, 28, 71, 87, 28]. We selected these four techniques because each of them is a representative of a category of approaches.

2.2.1 Linear Interpolation

Linear interpolation is the simplest interpolation technique. Given two trajectory points, $l^o_{t_1} = \langle x^o_{t_1}, y^o_{t_1} \rangle$ and $l^o_{t_3} = \langle x^o_{t_3}, y^o_{t_3} \rangle$, $t_1 < t_3$, and an unknown value of $l^o_{t_2} =$

$\langle x^o_{t_2}, y^o_{t_2} \rangle$, we use Equation 2.14 to calculate the value of $l^o_{t_2}$.

$$\begin{aligned} t_2 &= t_1 + \frac{t_3 - t_1}{2} \\ x^o_{t_2} &= x^o_{t_2} + \frac{x_{t_3} - x_{t_1}}{2} \\ y^o_{t_2} &= y^o_{t_2} + \frac{y^o_{t_3} - y^o_{t_1}}{2} \end{aligned} \tag{2.14}$$

We apply this technique to compute the location of the moving object at t_2 when we have the location of the moving object, $\langle x^o_{t_1}, y^o_{t_1} \rangle$ and $\langle x^o_{t_3}, y^o_{t_3} \rangle$, at times t_1 and t_3. The minimum number of points we need for Linear interpolation is two points.

2.2.2 Linear Regression Extrapolation

Linear regression is a linear method to represent the relationship between a dependent variable (x and y for two models LR_x and LR_y) and one or more independent variables (t). Given trajectory points of a single object $l_{t_1}, l_{t_2}, ..., l_{t_p}$, where $t_1 < t_2 < ... < t_p$, and the value of $l_{t_{p+1}}$ is unknown, we fit two linear regression models to the $x(t)$ and $y(t)$ of the known points. $LR_x = x(t)\beta_x + \epsilon_x$ is a representation of y_t and y_t so that β_x and ϵ_x are defined in a way such that the LR_x is the line with minimum distance to all data points, (x, t). Similarly, $LR_y = y(t)\beta_y + \epsilon_y$ is a representation of y_t and y_t so that β_y and ϵ_y are defined in a way such that the LR_y is the line with minimum distance to all data points, (y, t). $X(t) = LR_x(t)$ and $Y(t) = LR_y(t)$ are two trained linear regression models estimated using known trajectory points that can be used to predict $X(t_{p+1})$ and $Y(t_{p+1})$.

2.2.3 Kinematic Extrapolation

Kinematic extrapolation method is based on the dynamics of movement. We need to calculate the speed (v_x, v_y) and acceleration (a_x, a_y) of a moving object for each trajectory point and use Equation 2.15. In order to calculate the extrapolated value we need at least three trajectory points, $l_{t_1}, l_{t_2}, l_{t_3}$ to extrapolate l_{t_4}.

$$\begin{aligned} v_x(t+1) &= \frac{x_{t+1}-x_t}{t_{t+1}-t_t} \\ v_y(t+1) &= \frac{y_{t+1}-y_t}{t_{t+1}-t_t} \\ a_x(t+1) &= \frac{v_x(t+1)-v_x(t)}{t_{t+1}-t_t} \\ a_y(t+1) &= \frac{v_y(t+1)-v_y(t)}{t_{t+1}-t_t} \\ x(t+1) &= \frac{1}{2}a_x(t+1)^2 + v_x(t+1) + x_t \\ y(t+1) &= \frac{1}{2}a_y(t+1)^2 + v_y(t+1) + y_t \end{aligned} \tag{2.15}$$

2.2.4 Random Walk

Random Walk is an extrapolation/ interpolation method for cases in which the moving object behaves randomly [72]. In this method, we calculate (i) the interval of direction variations, $\bar{\theta} \pm \sigma_\theta$, which is the mean of direction variations in each known point calculated using Equation 2.16 where $l^o_{t_1} = \langle x^o_{t_1}, y^o_{t_1} \rangle$ is the first point, $l^o_{t_2} = \langle x^o_{t_2}, y^o_{t_2} \rangle$ is the second point (Δx is the difference in longitude), and (ii) the interval of the step of a moving object, $\bar{l} \pm \sigma_l$, which is the mean of the distance between known points[1] [72]. The number of points in this extrapolation is very important and the frequency of capturing points can affect the determination of the number of points needed for extrapolation. This method is useful to extrapolate animal movement [72]. Note that x and y are in radian. To convert x_t and y_t from degree to radian, we multiply their value by $\frac{\pi}{180}$.

$$\theta = atan2(\sin(\Delta x)\cos(y_2),\ \cos(y_1)\sin(y_2) - \sin(y_1)\cos(y_2)\cos(\Delta x)) \tag{2.16}$$

$$l_\theta = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{y_2-y_1}{2}\right) + \cos(y_1)\cos(y_2)\sin^2\left(\frac{x_2-x_1}{2}\right)}\right) \tag{2.17}$$

where r is the radius of the earth.

For example, assume a moving object travels from point $A = \langle 0, 0 \rangle$ to point

[1]For calculating distance we used haversine distance [13]

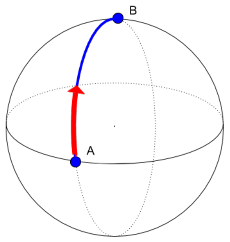


Figure 2.1: An example of calculating θ and l_θ when a moving object travels from point A to point B using random walk kernel.

$B = \langle 90, 90 \rangle$, shown in Figure 2.1. We calculate θ and l_θ as follows.

$\theta = atan2(\sin(\frac{\pi}{2})\cos(\frac{\pi}{2}), \cos(0)\sin(\frac{\pi}{2}) - \sin(0)\cos(\frac{\pi}{2})\cos(\frac{\pi}{2})) = atan2(0,0) = 0$ which means the moving object headed to the north.

$$l_\theta = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\pi}{4}\right) + \cos\left(0\right)\cos\left(\frac{\pi}{2}\right)\sin^2\left(\frac{\pi}{4}\right)}\right) = \frac{\pi r}{2}$$

2.2.5 Cubic Method

Cubic method is a method that uses a third degree polynomial. It can be used as both an interpolation method or extrapolation method because it fits a curve to the known points and returns its function. Given n trajectory points, where there is no duplicate time value, $t_i \neq t_j$, we can estimate two functions $S_x(t)$ and $S_y(t)$ to predict the $x(t)$ and $y(t)$ values of unknown trajectory points. The details of calculation and implementation are available in [45, 79]. Some Cubic interpolation variations, such as Hermite and Spline, are confirmed to be useful for interpolating the Automatic Identification System. (AIS) data [83]. The cubic spline is determined to reconstruct AIS data better than Hermite interpolation [83]. However, considering the fact that behaviour of a cargo vessel and a small fishing vessel are vastly different, we might be interested to look into more general approaches such as the methods selected in this research.

2.3 Data Structure

A *trajectory point* of a single moving object comprises two core components, namely timestamp and geolocation. Moreover, it can be related to some optional domain-related elements such as local and global features that we call point features. A sample trajectory with one point-feature is shown in Table 2.2.

Symbol	Description
a_x	Acceleration of moving object in direction x
a_y	Acceleration of moving object in direction y
k	Number of segments in a trajectory (τ)
L	The set of all trajectory points
l_θ	The distance step of movement in a random walk movement
n	Number of trajectory points in a trajectory (τ)
N	The set of all measurement functions for measuring segment feature
N_q	Measurement q for measuring segment feature
Q_j	Measured value for segment feature using N_q
S	Set of all segments in a trajectory (τ)
SF_q	Segment Feature
τ	Raw trajectory
θ	Direction variation in a random walk movement from the north.
v_x	Velocity of moving object in direction x
v_y	Velocity of moving object in direction y

Table 2.1: The symbols and notations used in this chapter

The *geolocation* includes the latitude and longitude of the moving object. In some domains, altitude can be part of the geolocation.

The *timestamp* is the component that captures the temporal aspect of a trajectory point. Although date-time is the most common example of a timestamp, using an ordinal number or chain of events can be other representations. Whenever we talk about timestamps, we use the conventional example — date-time — unless otherwise stated.

The *point features* are the optional assignments to a trajectory point that measure an attribute of a moving object at particular geolocation and timestamp, a formal definition of point feature is presented in Section 2.1. Table 2.2 presents six samples of trajectory points with one point-feature, speed, and two segment features, namely mode and Segment Identifier (SID).

A label or a set of tags can be assigned to each trajectory point, which is a particular case of a segment feature. This feature is a categorical variable that can provide semantic knowledge attached by a user to each segment. Since we will frequently utilize this specific feature, we define it as an independent entity, and we call it Label. An example of of trajectory label is displayed in Table 2.2 (column Mode).

Another remarkable point feature is Segment Identifier (SID). SID distinguishes the parent segment of a trajectory point. All the trajectory points in a segment have the same segment-id. Due to the particular role that this feature plays in the segmentation task, we consider it as an independent entity, and we call it **Segment Identifier (SID)**. A raw trajectory with two segments (Walk, and Bus) is shown in Table 2.2.

Timestamp	**Geolocation**		**Point Features**	**Segment Features**	
Time	**Longitude**	**Latitude**	**Measured Speed**	**Mode**	**SID**
2008-04-01 00:48:32	80.297195	41.144058	0.04	Walk	1000
2008-04-01 00:49:32	80.286858	41.14214	0.19		
2008-04-01 00:50:32	80.282905	41.152378	0.18		
2008-04-01 00:53:32	80.276862	41.167892	0.80	Bus	1001
2008-04-01 00:54:32	80.274478	41.17449	0.77		
2008-04-01 00:56:32	80.269932	41.172747	0.42		

Table 2.2: A sample trajectory data with a point feature (measured speed), and two segment features (mode, SID)

After segmenting a raw trajectory, we can calculate some features related to that segment; we call them segment features, detailed in Section 2.1. These features can be calculated globally or locally with respect to a segment. An example of extracted segment features (duration, minimum speed, day of week, average speed, mode, and segment ID) is shown in Table 2.3.

Duration	**Min Speed**	**Day of week**	**Average speed**	**Mode**	**SID**
3:00	0.04	Tue	0.136	Walk	1000
4:00	0.42	Tue	0.663	Bus	1001

Table 2.3: A sample segment feature

2.3.1 Spatial-temporal Data Structure VS. Trajectory Data Structure

In the literature on spatial-temporal data, we identified some research that considers trajectory data structure as a particular case of spatial-temporal data. However, there is a nuanced difference that makes trajectory data different and more complex than spatial-temporal data. Spatial-temporal data describes an entity in a specific time and location, such as the amount of precipitation in particular geolocation and a specific time. Therefore spatial-temporal data describes a static object, while a trajectory describes the behaviour of a moving entity. A moving object does not have a static location. The action of a moving object changes according to many factors, such as the environmental situation, or local circumstances. For instance, a moving object starting the movement from a rural area and continuing the action in a metropolitan area have different reactions to achieve the same objectives.

2.3.2 Single Moving Object VS. Multiple Moving Objects

A dataset of trajectories can be collected by a single moving object (single-sensor) or multiple moving objects (multi-sensor). When the whole dataset is about the movement of one moving object, we call it a single moving object, such as a vessel. This attribute means the moving object cannot be in two different locations at the same time. Multiple moving objects are the cases in which there are more than one moving object in the dataset. This concept adds more complexity to our dataset. For example, the AIS dataset contains a history of more than one vessel.

2.3.3 Synchronous VS. Asynchronous Data Collection

The data can be collected synchronously or asynchronously. When we receive data using a device such as a smartphone or GPS device, all that data can be recorded with a disciplined pattern, such as a fixed sampling frequency. However, when a moving object broadcasts its data, and the terrestrial towers or satellites collect the broadcast data, there is a chance of introducing some uncertainty to the data and missing part of the data. This uncertainty is due to factors such as receiver coverage, transmitter signal power, or the environment condition. An example of asynchronous data is Satellite Automatic Identification System (S-AIS) data that is broadcasted by the vessels and captured by satellites. Sometimes we combine the received data to enrich the dataset. For example, the AIS data collected by terrestrial towers can be merged by S-AIS data collected by the satellites. In such cases, finding duplicate AIS messages can be a challenging task.

2.3.4 Trajectory Annotation or Trajectory Tagging

In some applications such as animal behaviour studies, a subject matter expert assigns a label or SID to each trajectory point. This practice is called trajectory annotation or trajectory tagging. This task is time-consuming and costly because a subject matter expert is required to review trajectory points and decide on the features assigned to each trajectory point. Some platforms, such as VISTA, are available for trajectory tagging, which facilitate trajectory annotation using visual analytic and visualization tools [68].

Chapter 3

Related Work

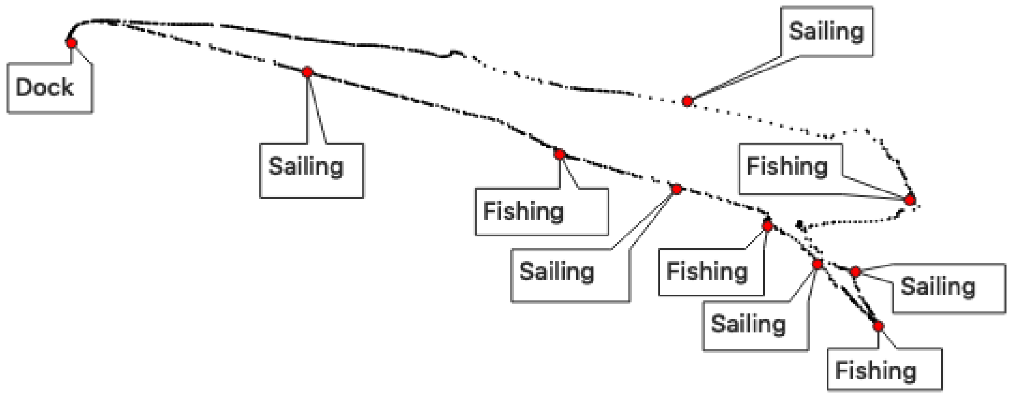
Trajectory segmentation has long received much attention in the mobility data mining field because trajectory patterns may not hold during the entire trajectory but these patterns might hold on trajectory parts. This process can summarize trajectories and render a segment-based view of mobility data as a representation of a trajectory. This perspective generates the field of semantic trajectory mining [59, 70]. Trajectory segmentation is the sine qua non condition for the migration from trajectory mining to semantic trajectory mining. More details on semantic trajectory are presented in Section 2.1. In this chapter, we review the literature on the trajectory segmentation topic. Then, the objectives and other aspects of existing approaches are discussed.

A semantic trajectory can be generated from two approaches. In the first approach, a trajectory is enriched with some background datasets such as maps and geographical layers [51]. For example, a trajectory of an individual and a geographic layer that shows shopping centers, schools, workplaces, and residential areas may produce a semantic trajectory for a person which represents their daily activities like *domestic activities* $\longrightarrow$ *shopping* $\longrightarrow$ *work*, as illustrated in Figure 3.1 (a).

In the second approach, a semantic trajectory is generated from a trajectory with no help from other background datasets. For instance, processing marine navigation data to create a semantic trajectory for fishing activities like *Docking* $\longrightarrow$ *Sailing* $\longrightarrow$ *Fishing* $\longrightarrow$ *Sailing* $\longrightarrow$ *Fishing* $\longrightarrow$ *Sailing* $\longrightarrow$ *Fishing* $\longrightarrow$ *Sailing* $\longrightarrow$ *Fishing* $\longrightarrow$ *Sailing* $\longrightarrow$ *Docking*, illustrated in Figure 3.1 (b). The second approach shows behavioural changes in movement devoid of the use of geographical semantic information and this is the focus of our review. There have been some domain-specific approaches to behavioural changes in moving objects [15, 14]. Detecting stop and move behaviors in traffic management systems [8], discovering fishing and non-fishing activities [16], analyzing the migratory pattern of animals [15], and transportation mode prediction [25, 14, 18] are some examples of investigating behavioral changes in different domains.

Work
Home
Shopping

(a) The moving object here is an individual who commutes from work (domestic activities) to home and stops by a shopping center on the way. The semantic trajectory is domestic activities-shopping-home.



(b) The moving object here is a fishing boat starting a voyage from a harbour and go for fishing. Then come back to the harbour.

Figure 3.1: Samples of semantic trajectories.

3.1 Objectives of Trajectory Segmentation Algorithms

Trajectory segmentation algorithms have one or more of the four following objectives, as we discuss in the related work. First, some trajectory segmentation algorithms, like the one in [14], only aim to make a uniform data structure. These types of algorithms prepare trajectory data for a specific task, such as feeding a trajectory to a neural network model with fixed input size [14]. The second objective of trajectory segmentation is maximizing the homogeneity of points belonging to a segment. For example, Greedy Randomized Adaptive Search Procedure for Unsupervised Trajectory Segmentation (GRASP-UTS) is a greedy algorithm that utilizes Minimum Description

Length (MDL) to discover segments with the highest possible homogeneity on specific features [69]. The third objective of trajectory segmentation is to identify interesting points or hot spots, such as SPD [85], DB-SMoT [61], and CB-SMoT [58], and discover hot spots [56]. Forth, another approach that we observed in the reviewed literature is finding patterns, including temporal patterns, in trajectory data. SeqScan is an example of discovering ordered clusters and temporal patterns in mobility data [15].

3.2 Aspects of Trajectory Segmentation Algorithms

Having provided a general perspective about varieties of trajectory segmentation methods, we are ready to compare significant aspects of them. The works were evaluated regarding four main categories aspects named: Learning, Features, Performance, and Others.

In the learning category, we explore the objectives of algorithm and how the algorithms learn how to create trajectory segments. This includes what methodology the algorithms uses, the algorithm's capability of being executed in a parallel environment, and how data is presented to the algorithm. This category includes the following aspects: classes of segmentation algorithms, nature of algorithms, and type of supervision. These aspects are summarized in Table 3.1 in page 30.

The feature aspects covers properties of algorithm to identify how the algorithms apply features in the process of trajectory segmentation. This category includes the type of extractable knowledge, the number of features, and type of features. Table 3.2 provides a compact version of this information.

The complexity of algorithm, amount of memory requirement for execution and the algorithm robustness to noise are discussed in the performance aspects, and are summarized in Table 3.3.

The rest of aspects for segmentation algorithms such as number of segments, whether the algorithm benefits from collective behaviour, methods of evaluation and whether the segments can overlap will be covered under the category of others aspects, and are summarized in Table 3.4.

First, we detail each aspect, and after, we compare related work based on each of them.

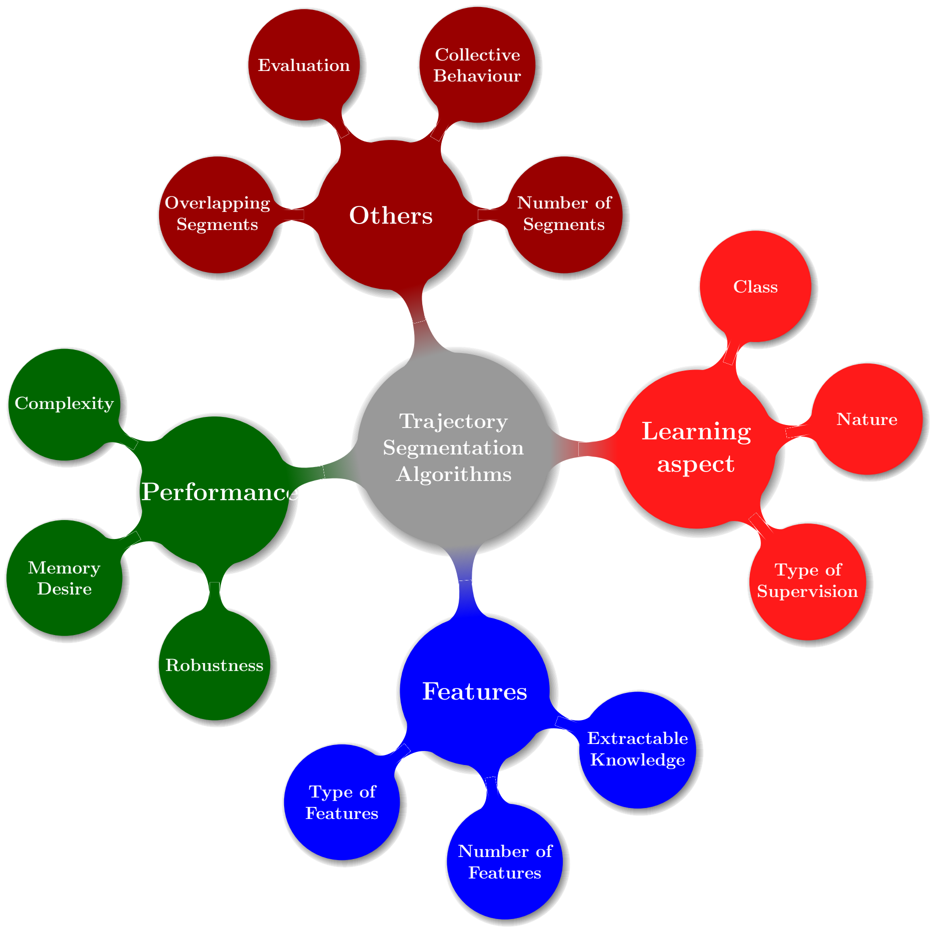


Figure 3.2: An overview of trajectory segmentation aspects studied in this research

3.2.1 Learning Aspects

Classes of Segmentation Algorithms

Trajectory segmentation algorithms can be classified into four groups. The first group consists of algorithms that generate a fixed uniform size of trajectories. These methods are useful for situations in which processing of a point-based view of trajectory points is performed with a *fixed size* of input structure. For example, processing raw trajectories needs a fixed size of inputs to train a deep neural network model. This method has been applied in some research as a part of the pre-processing method [14].

Second, some methods focus on performing the trajectory segmentation task as a solution to an optimization problem. There is usually a *cost function* in these methods which they aim to minimize. For instance, GRASP-UTS falls into this category since it uses the MDL concept to minimize the heterogeneity in each trajectory segment [69]. Another example of this method is the SeTraStream in which the RV-Coefficient has been used in as optimization method [81].

Third, trajectory segmentation can been viewed as a *clustering method*. In these approaches, clustering techniques have been employed to find clusters of trajectory points. One subcategory of clustering methods is the K-Means family. A specific version of K-Means which is called Warped K-Means was customized for trajectory segmentation [44]. The problem with these methods is that the algorithms require the number of clusters which usually is not available in the trajectory segmentation task. Having iterations to find the number of optimal clusters is also very costly.

Another approach in clustering methods is the family of density-based clustering algorithms [7, 58, 61, 19, 76, 47]. This is one of the preferred algorithms for trajectory segmentation since there is no need to provide the number of segments; these algorithms can find it automatically. Different versions of DB-SCAN such as CB-SMoT, DB-SMoT, and SeqScan belong to this category and have been proposed to find interesting points and patterns. There are a few caveats regarding the use of DB-SCAN. First, the border points, see Definition 1 in [20], can belong to more than one cluster, and the order of processing data plays an essential role in generating clusters with slightly different borders. This characteristic makes DB-SCAN to some degree unfit for this particular situation. There are some improvements on the basic DB-SCAN to solve this issue; however, we did not find the application of these improvements in the trajectory segmentation literature [10]. The second caveat, which is general to clustering methods, is that the distance measure plays a vital role. Selecting an inappropriate distance measure can profoundly affect the result of clustering. For example, if trajectory data belongs to different geographical zones, applying Euclidean distance decreases the quality of clustering. Third, DB-SCAN became less favorite in situations that clusters have different density.

For example, using CB-SMoT, which is inspired by the DB-SCAN method and is based on speed in the transportation mode domain, could not be the best choice to find consecutive segments with the same transportation mode (for example, two consecutive walk segments). This is because two consecutive segments with same transportation mode maintain the same speed distribution.

Another approach in clustering methods is the TRACLUS algorithm [41]. This method employs two sub-tasks: 1) partitioning and 2) clustering line segments. The partitioning itself can be considered as a trajectory segmentation method which is based on MDL. The second step does not segment trajectory points, while it extracts the collective behaviour patterns to provide a semantic method of segmentation. The algorithm performs this information extraction by clustering the lines generated in the first step.

The last class of algorithms involves methods based on a *sliding window*. This class of algorithms has a few advantages over the other methods. Since processing a sliding window requires only a small amount of memory and computational power, they can be used in embedded systems and real-time processing environment. Moreover, since processing each window can be independently performed, the likelihood of taking advantage of parallel processing is very high. The first step of TRACLUS (partitioning) is an example of a sliding window method.

Nature of Algorithms

The *nature* of an algorithm is a characteristic that shows to what degree we can implement it on a parallel processing environment using methods such as map-reduce. If a method has a sequential logic, with *sequential nature*, in which each steps requires results of the previous step, it can not easily be implemented in the parallel environment. On the other hand, if we have processes with independent steps we can implement the method on a parallel platform, *parallel nature*, to achieve functionalities such as real-time processing.

Types of Supervision

The type of supervision of trajectory segmentation is an important aspect representing the high-level structure of the input of the methods. To be consistent with the machine learning literature we define three major categories.

An *unsupervised method* is a method that does not have access to labeled data and processes a raw trajectory. This type of segmentation method is a function of a raw trajectory, i.e., $f(\tau)$. This type is more useful than supervised methods because of the complexity of the trajectory tagging task. This task has a time and complexity cost because an expert is required to label the data. An unsupervised trajectory segmentation task may have some parameters which can be tuned by the help of some labeled data. This type of algorithm can work without tuning with lower performance. Therefore, they do not depend on the existence of labelled data. Warped K-mean, CB-SMoT, DB-SMoT, and GRASP-UTS are a few examples of unsupervised methods.

The second type of method is a *semi-supervised method* in which the algorithm requires some labeled data to process unlabeled data to do the segmentation task. The segmentation is a combination of a supervised and an unsupervised function, i.e., $f1(\tau_1, l)$ and $f2(\tau_2)$, where $\tau = \tau_1 \cup \tau_2$ where τ is the whole trajectory dataset, τ_1 is the labeled part of data and τ_2 is the unlabeled part of data. This type of method is useful when we have a small subset of labeled data to expand on the knowledge of the subject matter expert. RGRASP-SemTS, and RB-SAT are two examples from this family. RGRASP-SemTS is a semi-supervised method because it uses a supervised method to extract semantic landmarks, $f1$, and an unsupervised approach to process the similarity of trajectory points and calculate cost function, $f2$.

Third, *supervised trajectory segmentation* is a method that has trajectory data plus a feature called *label*. This type formally defines a trajectory segmentation method as a function $f(\tau, l)$ where τ is a raw trajectory and l is the label for each trajectory point assigning each point to segment[1], i.e., segment identifier.

These types of trajectory segmentation learn a pattern from labeled data and use the extracted knowledge, known as a model, to segment unlabeled data in the future. Despite the fact that we did not find a supervised example of trajectory segmentation, we define this class to show the gap in the literature and possible future work.

Most of the related work is focused on trajectory segmentation task as an unsupervised task, shown in Table 3.1. However, there are applications that benefit from semi-supervised trajectory segmentation and supervised trajectory segmentation. A tagging system that interacts with a subject matter expert in a specific domain to label the trajectories benefits from semi and fully supervised trajectory segmentation methods using active learning [38, 68].

Method	**Class**	**Nature**	**Type of supervision**
SPD	Clustering	Sequential	Unsupervised
Warped K-Means	Clustering	Sequential	Unsupervised
CB-SMoT	Clustering	Sequential	Unsupervised
DB-SMoT	Clustering	Sequential	Unsupervised
SeqScan	Clustering	Sequential	Unsupervised
Time-based	Fixed window	Parallel	Unsupervised
Distance-based	Fixed window	Parallel	Unsupervised
GRASP-UTS	Cost function based	Sequential	Unsupervised
SeTraStream	Cost function based	Sequential	Unsupervised
RGRASP-SemTS	Cost function based	Sequential	Semi-Supervised
TRACLUS	Sliding Window	Sequential	Unsupervised
RTR Family	Sliding Window	Sequential	Unsupervised
Global Voting	Sliding Window	Sequential	Unsupervised
StopFinder	Clustering	Sequential	Unsupervised
TrajDBSCAN	Clustering	Sequential	Unsupervised
SPET	Clustering	Sequential	Unsupervised

Table 3.1: Comparing characteristics of trajectory segmentation approaches in terms of Learning aspects

[1]Note that this type of data can be utilized for evaluation of the trajectory segmentation algorithms.

3.2.2 Feature Aspects

Extractable Knowledge

Depending on which methods are applied for trajectory segmentation, we can infer different knowledge from data. The *extractable knowledge* from data can be categorized into five categories. First, trajectory segmentation can extract *interesting points*. Methods such as CB-SMoT, DB-SMoT, and StopFinder that are interested in finding stop points belong to this category. Second, it can obtain a *uniform data structure with a weak semantic* [14]. These methods are interested in changing the shape of mobility data to a uniform shape to be able to pass it to some existing models such as neural networks [14]. Combining such methods with other approaches can add a miscellaneous semantic to trajectory segments. Third, it can produce *homogeneous segments*. A cost function based segmentation methods aim to get the most homogeneous segments possible. The way they define homogeneity varies since they apply different features. Fourth, it can categorize segments based on a *similar pattern* of movement. These methods require information from several trajectories to find similar patterns between trajectories. Thus, this approach is computationally expensive. Fifth, it can detect the *behavior changes* in moving object. This is ideal that we segment trajectory data based on behaviour change since generating semantic trajectory conveys the meaning of changing the behavior of a moving object.

We define behaviour change as a deviation from a norm in this research; and the norm is defined differently in each domain. For example, in animal behaviour studies, there is a method called random walk [72] that shows the normal behavior. Moreover, the cubic spline interpolation method is a norm for marine traffic [83]. Furthermore, movement of a fast moving object can be modeled by a kinematic interpolation as a norm [48]. In case of absence of a norm, we define norm as a collective behaviour that can be learned from the mobility data warehouse.

Number of Features

The *number of features* to be processed in a trajectory segmentation method differs. Some methods use a *fixed* number of features to find the best segmentation while others are able to employ a *variable* range of features. An algorithm with a variable number of features can process any number of features provided by the user. In contrast, an algorithm with a fixed number of features is not flexible to process any information beyond its requirements. The techniques with a fixed number of features such as CB-SMoT, DB-SMoT, and StopFinder have a fixed complexity and accuracy; however, utilizing a variety of features can increase the complexity and accuracy of the method.

Types of Features

We define two types of features: i) intrinsic and ii) semantic. An intrinsic feature means feature is generated from movement dynamics. Therefore, a raw trajectory is the only knowledge that is required to compute these features. Semantic means the feature requires more information than a raw trajectory. For example, distance to shore can be a feature that needs a geographical semantic layer in addition to a raw trajectory in order to be computed.

If an algorithm is able to apply semantic features, it can be more specific to a domain. On the other hand, if an algorithm requires intrinsic features, it can be used in more general fields. Some algorithms are able to employ both types of feature which make them more powerful and sophisticated. However, calculating semantic features requires their algorithm to access some background knowledge and require more processing power.

3.2.3 Performance Aspects

Complexity

The *complexity* of an algorithm has been one of the important metrics to compare two algorithms. In this research, we have a focus on reporting the complexity of the segmentation tasks and have a comparison between them. When we apply an algorithm on a machine with limited capabilities of memory and processing power, the algorithms with less complexity can feasibility produce their promised results; however, methods with high complexity can crash due to the lack of memory or timeout error. Moreover, when running these algorithms on a machine that relies on a battery, the higher complexity means higher energy consumption.

Method	**Extractable knowledge**	**Number of Features**	**Type of Features**
SPD	Interesting Points	Fixed	intrinsic
Warped K-Means	Similar pattern	Variable	intrinsic
CB-SMoT	Interesting Points	Fixed	intrinsic
DB-SMoT	Interesting Points	Fixed	intrinsic
SeqScan	Interesting Points	Fixed	intrinsic
Time-based	Uniform dataset	Fixed	intrinsic
Distance-based	Uniform dataset	Fixed	intrinsic
GRASP-UTS	Homogeneous Segmentation	Variable	knowledge
SeTraStream	Homogeneous Segmentation	Variable	knowledge
RGRASP-SemTS	Homogeneous Segmentation	Variable	knowledge
TRACLUS	Similar pattern	Fixed	intrinsic
RTR Family	Homogeneous Segmentation	Fixed	intrinsic
Global Voting	Homogeneous Segmentation	Fixed	knowledge
StopFinder	Interesting Points	Fixed	intrinsic
TrajDBSCAN	Interesting Points	Fixed	intrinsic
SPET	Interesting Points	Fixed	knowledge

Table 3.2: Comparing characteristics of trajectory segmentation approaches in terms of Features aspects

The class of reviewed segmentation algorithms has two primary clustering methods. K-means is known to be of the complexity of $O(ntk)$ where n is the total number of objects, k is the number of clusters (unknown), and t is the number of iterations [33]. However, imposing some criteria on the K-mean can reduce the complexity to some extent [57]. By the way, we need to include the complexity of finding k which is of $O(n)$. The fact that we need to know the number of clusters makes this method less interesting since this fact guarantees that the complexity can be greater than $O(n)$.

Another family of clustering class of trajectory segmentation methods is based on the density-based clustering concepts and modifications of DB-SCAN. The complexity of DB-SCAN without using any indexing methodology is $O(n^2)$ [33]. This complexity can be reduced to $O(n\log(n))$ by using indexing methods such as r-tree [33, 31, 7].

Obviously, we require to add the complexity of creating r-tree index as well. In very optimistic circumstances with heuristic approaches, some of the variations of DB-SCAN, like CB-SMoT and DB-SMoT, can get to the complexity of $O(kn)$ which is still high because n is a big number in mobility data, usually a data stream. We observed some improvements on DB-SCAN, such as RT-DBSCAN, in which the clustering algorithm benefits from parallel processing structures; however, these approaches were not employed for trajectory segmentation tasks [29].

Fixed window methods are fast regarding complexity; however, they are not appropriate methods to segment trajectories because the behavior changes can happen in the middle of the fixed window and the algorithm is not capable of handling this situation. These algorithms can be useful when we utilized them in combination with other methods. They can be useful for data augmentation purposes.

The complexity of methods with cost function in the best scenario is $O(tn)$ where t is the number of iteration to converge [69]. The sliding window class of algorithms is more favorite concerning complexity since they can do parallel processing on each window. These methods may achieve a complexity of less than $O(n)$ in case of utilizing the parallel processing platform.

Robustness to Noise

Robustness to noise is another characteristic of a trajectory segmentation methods. Some methods can handle noisy data while others are sensitive to noise. For example, DB-SCAN is designed to handle noise while K-Means is sensitive to noise [33]. If the algorithm is sensitive to noise, it needs a pre-processing noise removal step. For cleaning the GPS data, there are different filtering methods including but not limited to hampel filter [32], Kalman filter [37, 54], and Savitzky-Golay filter [65]. Although an extension of a Kalman filter provides the best results for removing noise [37], it consumes a lot of computational power since it has an iterative nature (expectation - maximization). Moreover, most GPS devices perform a kind of embedded Kalman filter as their pre-processing before capturing data [37]. The Savitzky-Golay filter fits a polynomial function to a fixed window and the hampel filter is a simple method works based on the median of a fixed window [14]. In most of the devices, there is an embedded Kalman filter method to handle noise. This pre-processing step can be necessary in some cases but assuming the device already performs a Kalman filter makes us not to worry about this characteristic too much.

Memory Consumption Desire

Memory consumption desire is another aspect that shows which methods are capable of running on a system with limited memory space. We define two levels. First, *low memory need* is for algorithms that can release the memory they used in each step. It is highly related to the nature of the algorithm in which memory can be released. Second, *high memory desire* is for algorithms that do not have a mechanism to release the memory in each step or need the memory until the end of processing.

Method	Robustness	Memory Desire	Complexity
SPD	Yes	High	$O(n^2)$
Warped K-Means	No	High	$O(kn^2)$
CB-SMoT	Not Applied	High	$O(n\log(n))$
DB-SMoT	Not Applied	High	$O(n\log(n))$
SeqScan	Yes	High	$O(n\log(n))$
Time-based	No	Low	$O(n)$
Distance-based	No	Low	$O(n)$
GRASP-UTS	No	High	$O(in)$
SeTraStream	No	High	$O(n)$
RGRASP-SemTS	No	High	$O(kn)$
TRACLUS	No	High	$O(n\log(n))$
RTR Family RTR-DT, RTR-BU, RTR-SW	No, No, Yes	High, High, Low	$O(kn^2)$, $O(kn\log(n))$, $O(kn)$
Global Voting	Yes	High	$O(n\log(n))$
StopFinder	Yes	High	$O(n^2)$
TrajDBSCAN	Yes	High	$O(n\log(n))$
SPET	No	High	$O(kn)$

Table 3.3: Comparing characteristics of trajectory segmentation approaches in terms of Performance aspects

3.2.4 Other Aspects

Number of Segments

The number of segments is another aspect of a trajectory segmentation that needs to be considered in an algorithm. Some techniques such as K-Means-based [44] algorithms require the number of segments or in a more general way the number of clusters. Some other methods do not need this information, like the DB-SCAN-based algorithms. When the number of groups or segments is required, it imposes a limitation on the algorithm and increases the complexity of an algorithm by adding a step to estimate the best number of clusters. When a trajectory segmentation method expects number of segments, we call it *parametrized*, and when an algorithm does not require the number of segments, that is called *automatic* [69].

Collective Behavior

The *collective behavior* of a trajectory segmentation algorithm is a characteristic that shows the capability of the method to infer the segmentation using the trajectory of a moving object collected by a *single sensor* or *multiple sensors*. By the term sensor we mean a device embedded in a moving object that collects trajectory points information. When a technique processes single sensor trajectories, it can be run on the client side due to the independency of processing from other moving objects. When a method needs to extract knowledge from the patterns created by multiple trajectories, the processing is not efficient to be run on a client side and needs to be on a server side. This aspect of trajectory segmentation plays a vital role in the capability of a method for executing on sensor devices that rely on a battery. The algorithms with multi-sensor input cannot process single-sensor data because they rely on the summarized information from multiple sensors. Whereas, the single sensor algorithms can group the multi sensor data based on sensor identifiers and process each sensors' information.

Evaluation

Although evaluation of an algorithm has nothing to do with its properties, some properties impose limitations on the evaluation approach. We discuss evaluation of each method here to show what is most popular for evaluation of a trajectory segmentation algorithm. The *evaluation* methods are very diverse since we have different types of trajectory segmentation methods. For supervised methods, metrics such as accuracy, F-score, and AUC are generally approved methods. For unsupervised methods, we have two parameters called purity and coverage. *Purity* of a segment shows how much of a trajectory segment is divided correctly in comparison to the subject matter expert segmentation. The *coverage* shows how much the algorithm can cover the segments tagged by a subject matter expert. The Harmonic mean of Purity and Coverage (Harmonic Mean) represent both metrics since they are orthogonal if we assume there is no two consecutive segments belong to the same semantic. We have seen some research that compares the number of clusters generated by their algorithm with the number of clusters a subject matter expert proposed for that data. For general clustering tasks we have metrics including homogeneity, completeness, and v-score [62]. The purity, coverage, and harmonic mean are customized versions of these metrics for trajectory segmentation task.

Type of Dataset

Selecting a dataset for testing an algorithm is not a feature of algorithm; however, these features impose some limitation on capability of algorithms to perform a high quality results. Therefore, we find it useful to highlight that selecting a dataset with specific features can focus on limited aspects of a segmentation algorithm. Since we are covering different types of trajectory segmentation, we have seen a few different *datasets* for evaluation. Each dataset can have different characteristics such as whether it has labeled data or not. This characteristic can lead to three types of datasets: first, datasets devoid of any label data, *Unlabeled Dataset*. Second, datasets with features that identify the segmentation of a subject matter expert, *Identified Trajectories Dataset*. Third, data sets that have an attribute that shows the behavior of a moving object, e.g., transportation mode, *Fully Labeled Dataset*. Selecting a proper dataset for evaluation is highly vital since some datasets can not produce some kinds of evaluations. For example, unlabeled data cannot drive the purity and coverage.

Most of the datasets applied in the literature for evaluation were small datasets. For example, the StopFinder algorithm has been evaluated on a dataset of fewer than five hours of trajectory data in only two days [86]. The TRACLUS divided the full animal datasets based on the type of animals and reported the experiment on them [41]. This makes sense but the dataset is small for the collective behaviour. Also, the SeqScan discovering less than ten clusters in trajectory data [15]. This does not decrease the value of their work and shows that a segmentation method that can work on these datasets can be an acceptable method from the academic perspective. Two of the high quality datasets applied for evaluation are the dataset of fishing activities and hurricane dataset for the following reasons. First, they are a fully labeled datasets which means the trajectory id and activity type (fishing, sailing, level of hurricane) are available from the subject matter perspective. Second, the datasets have some level of uncertainty because of the noise in GPS devices. Third, both of them have considerable amount of trajectories and segments; however, they are not big datasets in comparison to Automatic Identification System (AIS) dataset.

Overlapping Segments

When we segment a trajectory, we divide it into consecutive segments. This means that there is no overlapping between two consecutive segments. Although all of the reviewed research make the assumption that segments do not have overlap, we noticed it is possible to have overlapping segments in real world. For example, a person who walks inside a train. This assumption can prevent an algorithm from wrong segmentation. A situation in which a trajectory point belongs to more than one segment with a membership function can be explored as a future work.

3.3 Trajectory Segmentation Algorithms

In this section, we present details on each algorithm related to our work and provide a general understanding of the trajectory segmentation algorithms. Then we review the aspects of trajectory segmentation algorithms. In order to perceive fully detailed explanations about each algorithm, we encourage the reader to read the original references.

Method	**Number of Segments**	**Collective Behaviour**	**Evaluation**	**Overlapping Segments**
SPD	Automatic	Single	# of Segments	No
Warped K-Means	Parametrized	Single	# of Segments	No
CB-SMoT	Automatic	Single	# of Segments	No
DB-SMoT	Automatic	Single	# of Segments	No
SeqScan	Automatic	Single	# of Segments, Purity	* Yes
Time-based	Automatic	Single	Not Applied	No
Distance-based	Automatic	Single	Not Applied	No
GRASP-UTS	Automatic	Single	Purity Coverage	No
SeTraStream	Automatic	Single	# of Segments	No
RGRASP-SemTS	Automatic	Single	AUC	No
TRACLUS	Automatic	Multi	# of Segments	No
RTR Family	Automatic	Single	Homogeneity (STHM)	No
Global Voting	Automatic	Multi	# of Segments	No
StopFinder	Automatic	Multi	Ground Truth	No
TrajDBSCAN	Automatic	Multi	# of Segments	No
SPET	Automatic	**Multi**	Not Applied	No

# of Segments: Number of Segments
* SeqScan can be considered as an overlapping-segments algorithm if we assume the noise segment detected between two clusters can belong to adjacent segments.

Table 3.4: Comparing characteristics of trajectory segmentation approaches in terms of the rest of the aspects.

3.3.1 Stay Point Detection (SPD)

Stay Point Detection (SPD) is a useful and straightforward algorithm that is used to segment the Geolife dataset [85]. This algorithm assumes that there is a stay point between each two segments. The algorithm uses a distance threshold (θ_d) and a time threshold θ_t. If a moving object spends more than θ_t time in the vicinity of θ_d the segment is called a stay point. Therefore, each stay point indicates a new segment, and the trajectory points between two stay points are also created as segments.

3.3.2 Warped K-Means (WKMeans)

WKMeans is a general purpose segmentation method that follows the idea of the legendary K-Means algorithm proposed by Macqueen [49]. This algorithm minimizes a cost function (e.g., quadratic error) and modifies the original K-means to deal with temporal constraints. The value of k determines the number of segments to be found by the algorithm which is an input for this algorithm. In this method, the initialization values were generated by employing a procedure called Trace Segmentation (TS) boundary [40] to produce the initial partitions (i.e., first segments). The constraint that points are allowed to move between adjacent clusters is practiced, and a cost function decides whether the movement is beneficial or not [44, 42]. Three different datasets have been evaluated in their experiments to make comparisons with five different algorithms in their literature. They reported that this algorithm could achieve up to 97% accuracy in these domains, obtaining more than 66% accuracy concerning the worst algorithm evaluated. This unsupervised algorithm uses intrinsic trajectory as input with variable features. The main drawback of this algorithm is the need for the number of segments. The algorithm processes single sensor trajectories, and it has a sequential nature. This method is sensitive to noise, and it requires high memory.

3.3.3 Clustering Based Stops and Moves of Trajectories (CB-SMoT)

CB-SMoT intends to discover stops and moves and segment a trajectory to these two clusters. This algorithm is an extension of DB-SCAN [58]. The original definitions of an ϵ-neighborhood and minimum points in DB-SCAN are modified so that the algorithm can benefit from temporal aspects of data. This algorithm operates based on the speed feature, and the stop points are trajectory points where the moving object has a lower speed. This clustering method has a sequential nature and aims to detect interesting points. The advantage of this algorithm in comparison to the Warped K-Means is that the algorithm will determine the number of segments automatically. This algorithm was evaluated on educational game in Amsterdam with 487 trajectories. The reported results show a high quality of detecting stop points in comparison to the first version of the Stop and Move (SMoT) algorithm [3]. Since the CB-SMoT performance was meaningfully better than the SMoT with similar concept we omitted the SMoT from the related work.

3.3.4 Direction Based Stops and Moves of Trajectories (DB-SMoT)

DB-SMoT aims to find stops and moves and creates two types of clusters: (i) stops, (ii) moves [61]. The DB-SMoT, similar to CB-SMoT, is an extension of DB-SCAN that considers the direction changes of a moving object to detect interesting points. The definitions of a $\epsilon-$neighborhood and minimum points are adjusted so that the algorithm can benefit from temporal aspects of data. This algorithm runs based on the direction feature. Similar to CB-SMoT, this clustering method has a sequential nature and consumes high memory space. Since this method is based on the DB-SCAN, it might handle some noise, but there is no discussion in this regard. This algorithm was evaluated using the fishing dataset for 22 days. The results find 25 interesting points over the fishing regions (the ground truth in their experiment). The comparison between this method and the CB-SMoT revealed that when the vessel was very heavy in the way back to harbor, the CB-SMoT detected a stop point which was not a correct segment. In reviewing this paper, we observed another method called Intersection Based Stops and Moves of Trajectories (IB-SMoT) that was conceptually similar to the CB-SMoT and DB-SMoT [4]. Therefore, we omitted this work from our comparative analysis.

3.3.5 SeqSCAN

SeqSCAN is a framework for the trajectory segmentation based on spatial density and temporal patterns [15]. This research reviewed the segmentation from the time series perspective and improve it for trajectory segmentation [73]. The SeqSCAN generates a collection of temporally separated clusters interleaved by segments of un-clustered trajectory points. This model proposes a noise detection model based on the distinction between local noise and transition. The SeqSCAN is an expansion of DBSCAN. This model is robust to noise and can distinguish two types of noise: (i) local noise, and (ii) transition. An essential contribution of this paper is the consideration of periodic patterns and transition part between segments. This algorithm is evaluated on two datasets: (i) animal migration dataset, (ii) an artificial dataset. This clustering sequential algorithm aims to extract interesting patterns and more specifically periodic patterns. This algorithm requires a considerable amount of memory and uses the intrinsic features of single sensor trajectories.

3.3.6 Time-based Approach

Some machine learning models have a fixed input data structure such as neural network, and convolutional neural network. When we decide to use these types of models, we need to do some pre-processing to convert our data into the models' input data structure, shape of input data. In the literature on trajectory mining, some papers are applying these methods [18, 14]. If the timestamp between two consecutive points is more than a threshold, they generate a new segment which is called a trip. Also, each trip divided into segments based on the trajectory labels. Then, each segment divided into a pre-defined number of trajectory points which is a fixed time or fixed number of trajectory points. This method proposed as a pre-processing and since it was not the focus of their paper, there is no evaluation for the segmentation part. Although this method is very trivial and cannot produce a good coverage, it can be very useful in data augmentation.

3.3.7 Distance-based Approach

In these methods, trajectories are split based on a fixed value that shows the trip distance [46]. This approach segments trajectory data by grouping consecutive trajectory points based on their spatial vicinity to identify activities and places. This segmentation can be accomplished by combining all consecutive trajectory points that are within a certain distance from each other (10m in this paper). They suggested associating trajectory points to a street map. There is no formal evaluation of the segmentation method since it was not the focus of their research. Another distance based approach can be the idea of hexagonal grids or square grids. We have not found any paper on trajectory segmentation using grids; however it can be considered as a trajectory segmentation method.

3.3.8 Greedy Randomized Adaptive Search Procedure for Unsupervised Trajectory Segmentation (GRASP-UTS)

GRASP-UTS is an unsupervised trajectory segmentation that applies the Minimum Description Length (MDL) principle to generate the most homogeneous segments considering a set of provided features. This algorithm generates random landmarks at the first step. Then, it produces the most homogeneous segments by moving the trajectory points and adjusting the landmarks based on a cost function value [69]. The advantage of this method is the ability to use knowledge more than just raw trajectory by adding features to the segmentation task. This algorithm is evaluated by two datasets: (i) the fishing dataset, and (ii) the Atlantic hurricanes dataset. The average purity and coverage have been applied to assess the quality of segments. The best configuration for this algorithm produces segments with the average purity and coverage of 90.90%, 87.34% respectively on the hurricane dataset. For the fishing dataset the average purity of 91.50% and average coverage of 87.18% for the best configuration are reported [69].

3.3.9 Semantic-Aware Trajectory Construction over Streaming Movement Data (SeTraStream)

SeTraStream is an unsupervised trajectory segmentation framework with a cost function method in the kernel that works based on the RV Coefficient of single sensor trajectories [81]. This framework is responsible for four tasks: (i) trajectory cleansing; (ii) compression; (iii) segmentation; and (iv) adding the labels to the segments using classification. This method has a sliding window and uses the RV Coefficient to decide on splitting a trajectory. This optimization approach divides trajectories to segments so that produced segments are highly homogeneous. The authors evaluated SeTraStream using Taxi dataset and Phone data provided by Nokia Research Center.

3.3.10 Reactive Greedy Randomized Adaptive Search Procedure for semantic Semi-supervised Trajectory Segmentation (RGRASP-SemTS)

RGRASP-SemTS is a semi-supervised method for trajectory segmentation using MDL principle. This algorithm allows a user to provide a small labeled trajectory dataset and this algorithm benefits from this small dataset to segment the rest of trajectories. This approach has been evaluated on two real-world datasets, and the results are competitive in comparison with the state-of-the-art methods.

3.3.11 TRACLUS

TRACLUS is a clustering approach for trajectory mining to detect dense regions with the same line segment characteristics. This algorithm has two steps: (i) partitioning of the trajectory in line segments; and (ii) clustering these lines into similar clusters. The partitioning step utilizes a cost function based on the Minimum Description Length (MDL) principle to split trajectory to line segments into three attributes related to the trajectory segments: (i) parallel distance; (ii) perpendicular distance and (iii) angular distance. In the second step, a density-based clustering method (DB-SCAN) is applied to cluster line segments [41]. This unsupervised clustering-based algorithm uses a fixed number of features and automatically calculates the number of segments. The algorithm uses the collective behavior and similar patterns of trajectories as a mined knowledge. Therefore it categorizes as a multi-sensor method. The nature of the algorithm is sequential since it applies the cost function and aims to minimize the value of the cost function. This algorithm is evaluated on deer and elk movement dataset and hurricane dataset. Although the algorithm is claimed that is robust to noises because of the DBSCAN core, the partitioning part which is the trajectory segmentation does not have any evaluation on robustness to noise. Since DBSCAN core performs clustering on extracted line instances, this algorithm is robast to noisy lines.

3.3.12 Robust Time-Referenced (RTR) Family

The family of three unsupervised trajectory segmentation methods (RTR-TopDown, RTR-BottomUp, and RTR-SlidingWindow) is presented to consider the spatial and temporal structure of trajectory data. The process is robust against noise, and three real-world datasets were applied to test the algorithm [82]. In this method a fixed number of variables is used and the number of segments automatically generates from trajectory. This method uses a sliding window and processes single sensor trajectories. The Spatio-Temporal Homogeneity Measure (STHM) was proposed as a evaluation method for this approach. The method is robust against noise. The complexity of RTR-TopDows is $O(kn^2)$ that can be reduced to $O(kn\log(n))$ using priority queues. The complexity of RTR-BottomUp is $O(kn)$ can be reduced to $O(k\log(n))$ using priority queues. And the complexity of RTR-SlidingWindow is $O(kn)$ where k is the length of sliding window. The authors of RTR conduct their experiments on three datasets: i) bus dataset with 108 segments, ii) truck dataset with 273 segments, iii) hand dataset with 660 segments. The experiments show the algorithm performs well on creating homogeneous segments; however, there is no comparison against other methods.

3.3.13 StopFinder

The StopFinder algorithm is a trajectory segmentation method based on finding stops and moves [86]. This work is an extension of Ordering Points to Identify Clustering Structure OPTICS [5] to deal with temporal proximity of trajectory points based on elapsed time and speed and the authors called it T-OPTICS [86]. The algorithm estimates the parameters for T-OPTICS in the first step. Then it runs the T-OPTICS and extracts the stops. After that, it generates the visualization. StopFinder was evaluated on a trajectory of a user using different transportation modes (e.g., walking, car and bicycle) for two consecutive days. The results showed that, unlike CB-SMoT, StopFinder could find stops with varying values of speed.

3.3.14 TrajDBSCAN

The TrajDBSCAN is a hierarchical trajectory segmentation that tackles three tasks: i) finding stop and moves in trajectories. ii) detecting custom stops (generated by a single user) and shared stops (created from collective behaviour), and iii) organizing the stops and moves in a hierarchical manner. The algorithm first generates the custom stops based on single user behaviour. Then if there is a minimum number of custom stops in an area, a shared stop is created. Then the algorithm produces a hierarchy for all the detected stops [80].

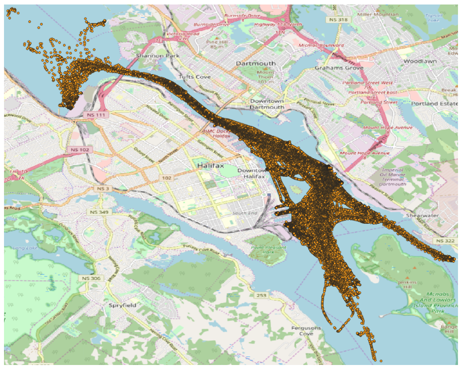
3.3.15 Stop and Proximity Episodes in Trajectories

The Stop and Proximity Episodes in Trajectories (SPET) is a trajectory segmentation algorithm that detects stops and episodes that occur close to a particular geographic area [53]. The SPET detects the stops based only on the time the moving object intersects a region of interest. In contrast, CB-SMoT and DB-SMoT identify stops using the speed and direction variation attributes during a specified time interval. This algorithm requires background knowledge about the region of interest to be able to find stops. There is no evaluation regarding the robustness to noise for this algorithm. One strength of this research is finding the transition segments that a moving object switches from one segment to another segment.

3.4 Summary and Discussion

A summary of the reviewed related work methods and their different characteristics is provided in Tables 3.1, 3.2, 3.3. We observe the gap of an unsupervised single sensor algorithm with low use of memory and low complexity. Moreover, we were not able to find a supervised segmentation algorithm. A supervised segmentation method can become domain-specific by using the labelled data in that domain.

The gaps in the literature motivate us to work on a specific situation that has not been well-studied. All the segmentation algorithms are pre-processing data to create a new representation for facilitating data mining approaches. Summarizing trajectories brings benefits such as improving privacy protection and the effective and efficient performance of sensor devices. Furthermore, a graph representation of mobility data can be extracted from mobility data only after the segmentation graph structure is an enriched representation that can provide high-level insights about data and answer more complex queries [11]. AIS messages, 766,671 trajectory points, generated by one vessel on Halifax waters are displayed in Figure 3.3.a. The graph representation (with 13 nodes) of this trajectory is displayed in Figure 3.3.b. This graph is extracted from the trajectory after segmentation. Each edge can carry segment features of the trajectories. Using this graph, we can answer which ports are the most crowded ones or where we have low traffic.


(a) Plotting 766,671 trajectory points generated by a vessel voyaging in Halifax waters.

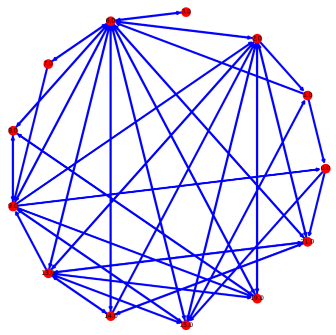

(b) Graph representation of trajectory points in the left figure after segmentation.

Figure 3.3: A raw trajectory of 766,671 trajectory points generated by a vessel voyaging in Halifax waters and its graph representation after segmentation.

There is a considerable number of studies on interpolation to reconstruct missing parts of mobility data in different domains, such as animal behaviour [15], transportation management [25], and marine traffic [16]. Utilizing this concept, we propose two segmentation algorithms. First, we propose an unsupervised algorithm that segment data using an interpolation kernel called octal window segmentation. We enhance this algorithm by a few changes to have a flexible sliding window, and we called the new version sliding window segmentation (SWS). Second, we introduce a supervised algorithm to segment trajectory data utilizing octal window segmentation kernel, a binary classifier and a majority vote, named WS-II. In the next chapter, details are provided about the datasets used in this thesis.

Chapter 4

Mobility Data and Its Properties

4.1 Datasets

Four datasets were selected for our work: (i) the fishing dataset (5190 trajectory points, 153 segments), (ii) the Atlantic hurricanes dataset (1990 trajectory points, 182 segments), (iii) a subset of the Geolife dataset (32046 trajectory points, 258 segments) and (v) Automatic Identification System (AIS) dataset (507,354 trajectory points, without ground truth). Our team collected the AIS dataset at the Institute for Big Data Analytics, and we prepared a subset of this dataset for debugging our algorithms. Therefore we have not used this dataset for comparing our algorithm with other algorithms. These datasets represent mobility data in transportation, meteorology, and marine navigation domains, which form the major part of the mobility data domains. The moving objects in the fishing dataset and the AIS dataset are vessels with segments representing their voyages. The Geolife dataset is the trajectory of human movements while using different transportation modes and the segments are the transportation modes. The Atlantic hurricane trajectories show the movement of the hurricanes' eye with the segments representing different levels of hurricanes.

4.1.1 Fishing Dataset

This dataset can be identified as a multi-object and asynchronous dataset. The fishing dataset was compiled in contribution to the project called "Programa Nacional de Rastreamento de Embarcações Pesqueiras por Satelite" (PRES) in Brazil as part of the federal government program to supervise fishing activities on the Brazilian coast. This dataset has 35 trajectories, 153 segments, 5190 trajectory points, and two labels (Fishing, and Non-Fishing) annotated by subject matter experts. To use this dataset in our study, which has the focus of a single sensor, we had to pre-process this dataset and solve the time overlaps by sequencing trajectories of different vessels. The average period of capturing in this dataset is about 105 minutes. This dataset is collected using AIS messages, and we expected to see the rate of capturing near to 2 to 5 minutes; However, the dataset is sparse, and each trajectory point is captured nearly 105 minutes after the previous one. During this 105 minutes gap, the vessel could make some behaviour changes, which makes the identification of partitioning positions challenging. An overview of the fishing dataset on a map is shown in Figure 4.1. The QGIS 3.4.2 were applied to generate this map. The dataset was originally divided into ten folds to apply cross-validation. Since trajectory data points are related to their adjacent trajectory points, we cannot randomly select a subset of this dataset to create folds for cross-validation. These folds are provided by Soares et al. [69] and we applied the same folds in our research. Each segment in these folds is shown with a color in Figure 4.1. This dataset is a *fully-labeled* dataset (contains both segment identifier and semantic label in its ground-truth).

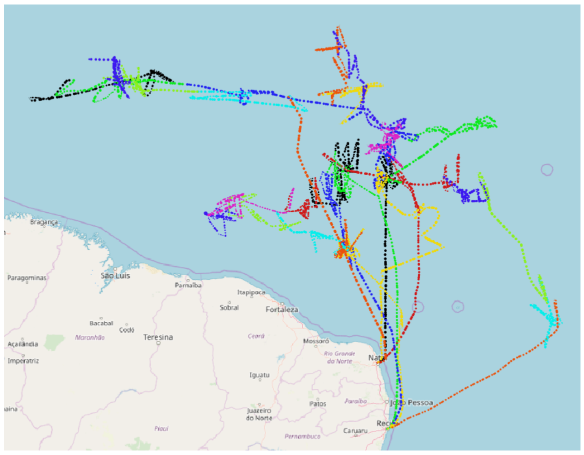


Figure 4.1: An overview of fishing dataset in which each colour represents a voyage's trajectory. Each colour symbolizes a segment that can be a fishing activity or non-fishing activity.

We provide some statistical information about this dataset in Table 4.1 because this statistical information offers insight into each method's performance. For example, a technique that benefits from a sliding window of size k might have a hard time detecting trajectories shorter than k.

Fishing Dataset	**Trajectory points**	**Number of segments**	**Segment length***			**percentage of segments shorter than 7**
			mean	**minimum**	**maximum**	
Fold 0	630	19	33.15	2	96	19
Fold 1	634	18	35.22	4	139	20
Fold 2	560	22	25.45	2	69	22
Fold 3	605	20	30.25	3	90	11
Fold 4	535	10	53.5	3	154	8
Fold 5	437	16	27.31	4	77	20
Fold 6	432	14	30.93	4	114	20
Fold 7	607	16	37.93	2	105	22
Fold 8	317	8	39.62	2	108	5
Fold 9	433	10	43.3	5	104	11
Total	**5190**	**153**	**33.92**	**2**	**154**	**158**

* Segment length is the number of trajectory points in each segment.

Table 4.1: Statistics of the fishing dataset

4.1.2 Atlantic Hurricanes Dataset

Hurricanes Dataset is an asynchronous dataset with unidentifiable multiple object data. This dataset is published by the National Oceanic and Atmospheric Administration (NOAA) [1], applied in the evaluation of GRASP-UTS, and it contains 52 trajectories, 1990 trajectory points, 182 segments, and two trajectory labels (hurricane level one and two). An overview of this dataset is presented in Figure 4.2. The dataset was divided into ten folds for applying cross-validation; each segment is shown by different colours in Figure 4.2, which is provided by Soares et al. [69]. The average period of capturing in this dataset is 498 minutes or about 8 hours. Atlantic Hurricane dataset is a *fully-labeled* dataset because segment identifiers (SID) and trajectory labels (label) are available. Statistical information related to the hurricanes dataset is presented in Table 4.2.

4.1.3 Geolife Dataset

The Geolife dataset, a frequently used benchmark dataset for mobility data research, is a synchronous dataset, a dataset that trajectory points are collected by moving object, (See Section 2.3.3) with identifiable multi objects, Microsoft Research Asia collected it from April 2007 to October 2011 [84]. The dataset has 5,504,363 records

[1]https://coast.noaa.gov/hurricanes/

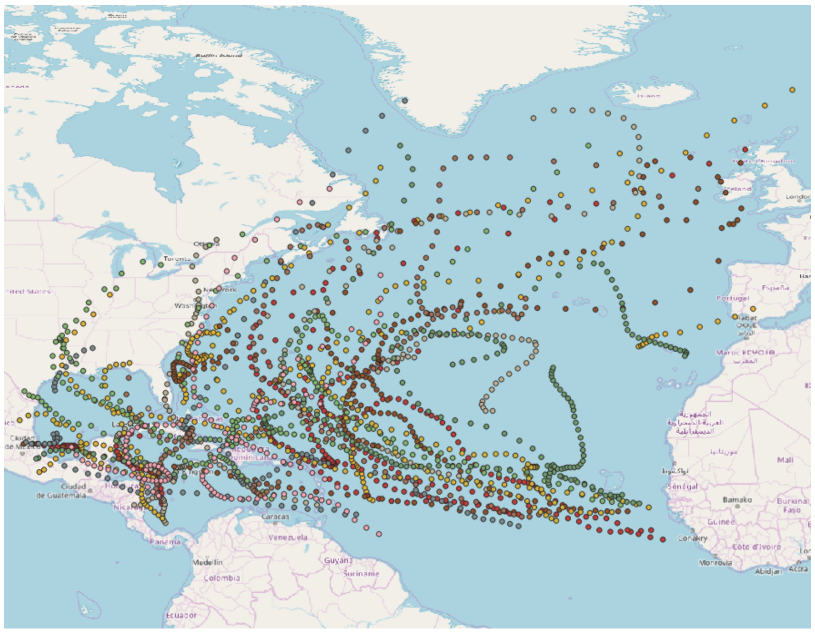


Figure 4.2: An overview of the Atlantic hurricanes dataset

labeled with eleven transportation modes: taxi (4.41%); car (9.40%); train (10.19%); subway (5.68%); walk (29.35%); airplane (0.16%); boat (0.06%); bike (17.34%); run (0.03%); motorcycle (0.006%); and bus (23.33%). We conducted our experiments on a subset of the Geolife dataset to see the effects of trajectory segmentation algorithms on longer trajectories with a higher frequency of capturing in transportation domain. Because all the algorithms received a trajectory of single sensor data, we create ten folds; each includes trajectories of one user. We avoid selecting very short trajectories because we have short trajectories in the fishing dataset and hurricanes dataset. We also avoid very long trajectories because some of the available trajectory segmentation could not provide their segmentation results for days. The average period of capturing in this dataset is about 22 minutes.

An overview of this subsection of dataset is displayed in Figure 4.3. We intentionally select some trajectories like the one in the red bounding box to see the effect of our proposed method on a mountain road with a lot of behaviour changes. Some statistics of this dataset are presented in Table 4.3. In this research, when we talk about the Geolife dataset, we refer to this part of the whole the Geolife dataset unless we explicitly refer to the entire of the Geolife dataset.

Hurricanes Dataset	Trajectory points	Number of segments	Segment length*			percentage of segments shorter than 7
			mean	minimum	maximum	
Fold 0	187	24	7.7	1	27	36
Fold 1	211	15	14.06	1	32	13
Fold 2	214	17	12.58	1	36	18
Fold 3	121	15	8.06	2	36	28
Fold 4	242	19	12.73	2	40	29
Fold 5	168	17	9.88	1	28	24
Fold 6	209	21	9.95	1	32	41
Fold 7	214	15	14.26	1	44	16
Fold 8	145	19	7.63	1	25	54
Fold 9	279	20	13.95	1	44	24
Total	**1990**	**182**	**33.92**	**1**	**44**	**283**

* Segment length is the number of trajectory points in each segment.

Table 4.2: Statistics of the Atlantic hurricanes dataset

Geolife Dataset*	Trajectory points	Number of segments	Segment length*			percentage of segments shorter than 7
			mean	minimum	maximum	
Fold 0	972	17	57.17	10	202	0
Fold 1	2587	45	57.48	4	158	4
Fold 2	5271	33	159.72	4	1329	4
Fold 3	1271	17	74.76	3	483	10
Fold 4	5390	52	103.65	4	928	16
Fold 5	2426	11	220.54	10	1123	0
Fold 6	6017	56	107.44	6	859	6
Fold 7	2587	45	57.48	4	158	4
Fold 8	1173	12	97.75	3	478	7
Fold 9	4352	15	290.13	1	771	1
Total	**32046**	**258**	**124.20**	**1**	**1329**	**44**

* Segment length is the number of trajectory points in each segment.

Table 4.3: Statistics of the Geolife dataset

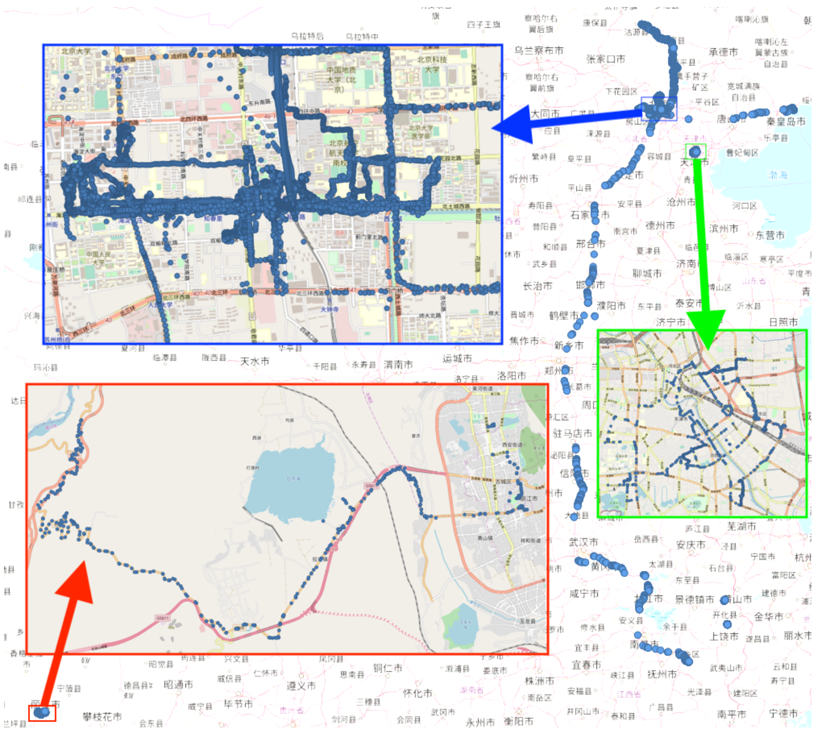


Figure 4.3: An overview of the Geolife dataset which is a subset of the whole Geolife data

4.1.4 Automatic Identification System (AIS)

This dataset is an asynchronous dataset with identifiable multi objects, including 507,354 trajectory points. An overview of this dataset is displayed in Figure 4.4. The data was captured using a Shakespeare Galaxy Antenna, dAISy HAT board, and a Raspberry Pi using a Python library called “aislib” with some modifications. Details of data collection architecture is provided in Appendix A. The antenna was set up at the Centre for Ocean Ventures & Entrepreneurship in Dartmouth, NS and the project financially supported by Innovacorp. In this subset of AIS data, we selected ten folds; each includes the movement of one vessel. This dataset was applied in our research for the debugging purpose; therefore, we did not use it to compare our algorithm with other trajectory segmentation algorithms. The average period of capturing in this dataset is about three minutes which is a normal capturing rate for AIS messages.

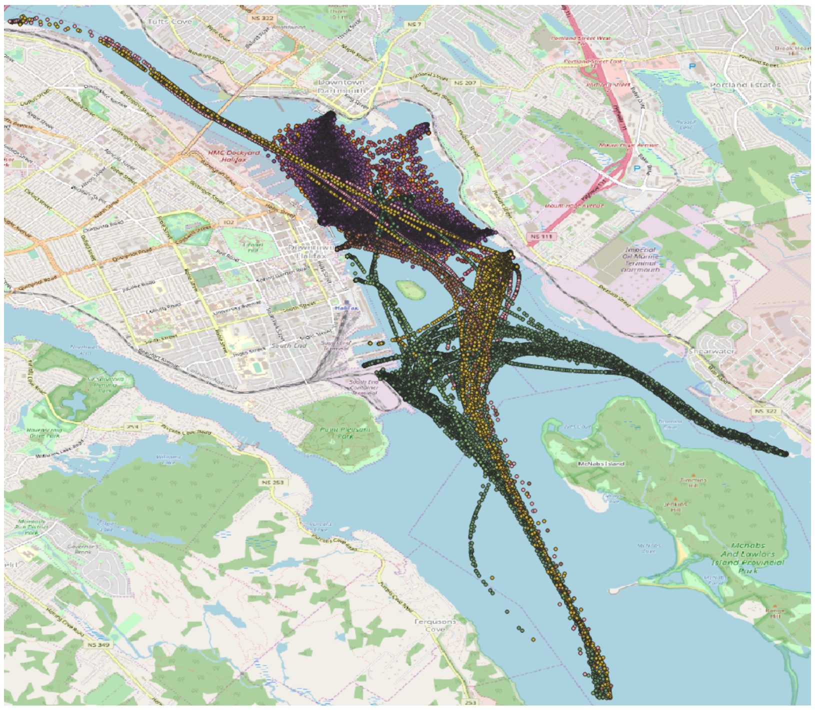


Figure 4.4: An overview of the AIS dataset which is a subset of the AIS messages captured near Halifax

4.2 Metrics

Although there are many metrics available for supervised and unsupervised machine learning methods, there are only two metrics that consider two factors of trajectory labels (label) and trajectory segment id (SID) annotated by subject matter expert applied by Soares et al. [69]: i) purity [69], ii) coverage [69]. The Harmonic mean of Purity and Coverage is derived from purity and coverage. Since we are interested in achieving both high purity and coverage, we report harmonic mean of them. Therefore we applied the definition of purity and coverage, provided by Soares et al. [69], to evaluate our method. We utilized the harmonic mean to fine-tune all algorithms. In this work, we have used two metrics. First, *purity* (P), as seen in Equation 4.1, where S is the set of segments discovered by segmentation algorithm, Λ_L is the set of labels provided by subject matter expert, k is the number of discovered segments, L is the number of expert labels, N_{ij} is the number of trajectory points inside segment s_i with label λ_j, and N_i is the total number of points found for the segment s_i. Second, *coverage* (C), as seen in Equation 4.2, where S is the set of segments discovered by

Symbol	Description
$\overline{C}$	Average Coverage
H	Harmonic means of Purity and Coverage
$\overline{P}$	Average Purity
$C(S, \Psi_v)$	Coverage of a segment according to ground truth.
$H(S, \Lambda_L, \Psi_v)$	Harmonic mean of Purity and Coverage
Λ_L	The set of labels by subject matter expert
L	The number of expert labels
N_{ij}	N_{ij} is the number of trajectory points inside segment s_i with label λ_j
N_i	N_i is the total number of points found for the segment s_i
$P(S, \Lambda_L)$	Purity of a segment according to ground truth.
Ψ_v	The set of segments by subject matter expert
v	The number of segments in Ψ_v
S	The set of segments discovered by segmentation algorithm
k	The number of discovered segments.

Table 4.4: The symbols and notations used in this chapter

segmentation algorithm, Ψ_v is the set of segments by subject matter expert, $N_{\psi_i \cap s_j}$ is the number of trajectory points of the segment s_j that belongs to the segment with ψ_i segment identifier, and N_i is the total number of points of the identified segment with segment identifier equals to ψ_i. These concepts were applied in the context of trajectory segmentation [69, 22, 23, 21] and we repeat it here to review the basis of our comparisons. These two metrics were designed to be orthogonal under the assumption that there are no adjacent segments with the same label; i.e., when one tends to increase, the other tends to decrease. Therefore, we defined *the harmonic mean*, *(H)*, of purity (P) (shown in Equation 4.1), and coverage (C) (shown in Equation 4.2), as shown in Equation 4.3, as the primary metrics for our experimental analysis. The harmonic mean simplifies the plots and comparison of the segmentation algorithms because it is a summarized version of both purity and coverage. However, we report both purity and coverage as well.

$$P(S, \Lambda_L) = \frac{1}{k}(\sum_{i=1}^{k} \underset{j \in [1,L]}{\operatorname{argmax}}(\frac{N_{ij}}{N_i})) \tag{4.1}$$

$$C(S, \Psi_v) = \frac{1}{v}(\sum_{i=1}^{v} \underset{j \in [1,k]}{\operatorname{argmax}}(\frac{N_{\psi_i \cap s_j}}{N_i})) \tag{4.2}$$

In addition to the base concepts of purity and coverage above, we provide more details concerning the purity and coverage calculation. *Purity* of a segment ($s_i \in S$) is defined as follows: Assuming the length of s_i is N_i. We count the number of trajectory points in s_i annotated as λ_j for all $\lambda_i \in \Lambda_L$. Now we select λ_j as the label with maximum occurrence and assign its occurrence to N_{ij}. Therefore, the purity of segment S_i is $\frac{|N_i|}{N_{ij}}$. The average of purity values for all segments generated by a trajectory segmentation algorithm ($s_i \in S$) is called *purity*, P. In order to calculate *coverage* for a segment in ground truth (SID), $\psi_i \in \Psi_v$, we identify all segments discovered by algorithm ($S_i \in S$) that overlap with ψ_i, where S is the set of discovered segments by algorithm. We exclude trajectory points with segment identifier ψ_i that do not belong to S_i. Now, we find the longest discovered segment $\hat{S}_i$ and calculate the coverage by $\frac{|\hat{S}_i|}{|S_i|}$.

$$H(S, \Lambda_L, \Psi_v) = \frac{2 * P(S, \Lambda_L) * C(S, \Psi_v)}{P(S, \Lambda_L) + C(S, \Psi_v)} \tag{4.3}$$

The average for *coverage* of all ground truth segments (segments with segment identifier $\psi_i \in \Psi_v$) is called C.

We define two concepts that identify two ends of an spectrum for evaluating a trajectory segmentation algorithm, called over-segmentation and under-segmentation. we note that both situation are undesired.

The more **over-segmented** a trajectory is, the lower the value of expected *purity*. The worst case for over-segmentation is when each trajectory point is identified as a segment when the ground truth has only one segment. For a trajectory with n trajectory points, $P = \frac{1}{n}, C = \frac{n * \frac{1}{1}}{n} = 1, H = \frac{2}{(n+1)}$.

The same conflicting result occurs if the trajectory is **under-segmented**, i.e. *coverage* values tend to decrease, but *purity* tends to increase. The worst case for under-segmentation is when only one segment is discovered by algorithm when the ground truth indicates that each trajectory point is an independent segment. For a trajectory with n trajectory points, $P = \frac{n * \frac{1}{1}}{n} = 1, C = \frac{1}{n}, H = \frac{2}{(n+1)}$. We measure the rate at which the produced segment covers the ground truth segment suggested by the subject matter expert using coverage. This metric is important because a trajectory segmentation algorithm should not make short segments while the subject matter expert is expecting longer segments. We encourage the algorithm to generate longer segments by selecting an algorithm with higher coverage. We measure the rate in which the produced segments purely represent a semantic (label) using purity. A trajectory segmentation algorithm should not mix two semantics in one segment since trajectory segmentation aims to reduce the inter-dependency between segments in the trajectory data. By selecting an algorithm with higher purity, we encourage selecting the algorithm that generates shorter segments. Since we cannot generate short segments and long segments simultaneously, we introduced a new measure called harmonic mean that considers a balance between both purity and coverage. Therefore harmonic mean is a single wise measure to measure the quality of segmentation.

Chapter 5

Sliding Window Segmentation

This chapter describes our unsupervised trajectory segmentation algorithm called Sliding Window Segmentation (SWS), which uses an interpolation kernel and a flexible size sliding window [23]. Our segmentation approach can be categorized as an unsupervised trajectory segmentation method because of using just the geo-locations and the time tags of a moving object. SWS is an enhancement of our segmentation method with a fixed sliding window, called Octal Window Segmentation (OWS) [22]. The difference between SWS and OWS is that OWS was the initial idea with a fixed-size sliding window. In SWS, we expanded OWS and change the algorithm to receive the sliding window size as a parameter of the algorithm. In this chapter, first, we explain the proposed method called SWS. We continue this chapter by providing a running example of the proposed trajectory segmentation algorithm. After proposing our method, we detail our algorithm's parameter selection procedure (Section 5.3.1). We present our experimental evaluation in Section 5.4. We discussed our experimental setup in Section 5.4.1. The interpolation methods analysis (kernels) obtained by Sliding Window Segmentation (SWS) algorithm is detailed in Section 5.4.2. Evaluation of SWS window size is presented in Section 5.4.3. We compared our approach with other algorithms and provide the details in Section 5.4.4. Sections 5.4.5 to 5.4.7 compare our proposed method and some other segmentation algorithms in terms of harmonic mean, number of discovered segments, memory and CPU requirements, and v-measure. A discussion on the strengths and weaknesses of our approach is provided in Section 5.5 and we conclude this chapter by a summary presented in Section 5.6.

5.1 Preliminaries

Trajectory segmentation can be performed on the moving object's hardware or a centralized distributed machine. In both situations, the memory and computational power are of concern. The Sliding Window Segmentation (SWS) method is motivated by using (i) limited knowledge (just time and geolocation of moving objects), (ii) limited memory, and (iii) limited computational power (CPU time). The idea behind this algorithm is that the behaviour of a single moving object in several domains has been well-studied. There are interpolation/extrapolation methods available that can represent and reconstruct the movement behaviour for those domains [48, 72, 83]. It is possible to memorize the general behaviour of a moving object from available large trajectory datasets using methods such Functional Data Analysis (FDA) [36, 60, 28, 71, 87, 28]. Any deviation from normal behaviour can be considered as a signal to segment a trajectory (i.e., place a partitioning position). To detect this deviation from the norm, we use a sliding window to calculate error values (distance between each point and their interpolated midpoint) for each trajectory point by using forward and backward extrapolation. We call these values *error signal, E*, presented in line one in Algorithm 5.1. Then we process this error signal to detect abrupt changes in moving an object's behaviour, line two in Algorithm 5.1. We start with defining extrapolation directions in the next section and Sliding Window in Section 5.1.2. Then, we explain two important components of the Sliding Window Segmentation (SWS) in Sections 5.2 and 5.3.

Algorithm 5.1 Sliding Window Segmentation Algorithm

```
Require: $\tau_n$ {The raw trajectory}
Require: $ws$ {Size of sliding window (an odd number)}
Require: $kernel$ {Interpolation kernel}
Require: $\epsilon$ {Epsilon threshold}
Require: $flag \leftarrow$Ture { Choose the output to be the segment IDs (True) or error
    signal (False)}
Ensure: output
1: $E \leftarrow GenerateErrorSignal(\tau, ws, kernel)$
2: if $flag$ is $True$ then
3:     $SegmentIDs \leftarrow SegmentationProcedure(E, \epsilon)$
4:     $output \leftarrow SegmentIDs$
5: else
6:     $output \leftarrow E$
7: end if
8: return $output$
```

5.1.1 Extrapolation and Interpolation

Our approach benefits from two types of methods: interpolations and extrapolations. We applied the interpolation method to estimate a trajectory point inside a set of trajectory points. The extrapolation is applied to generate the next trajectory point that comes after a set of trajectory points. Figure 5.1 shows that l^B and l^F are generated using extrapolation. The I^C is calculated using interpolation between two points of l^B and l^F, shown in Figure 5.1, in page 62.

We define two types of extrapolations based on the direction of movement: (i) Forward extrapolation, and (ii) Backward extrapolation. These two methods are very similar so that they can be categorized as one extrapolation method with changing the direction as a parameter; however, we define these two types to make the concept more precise and more comprehensible for the reader. The marriage of these two extrapolation methods forms an interpolation approach.

Assuming we have trajectory points from i to j, $\langle l_i, .., l_j \rangle$, *forward extrapolation* computes the location of moving object in l_{j+1} when the direction of movement goes from i to j. Similarly, *backward extrapolation* calculates the location where the moving object should have been there at $i-1$ if the moving object went from j to i. We emphasize that we construct trajectory points *within* a defined range in the interpolation with the help of Forward and Backward extrapolation. Each extrapolation method constructs a trajectory point *outside* of its provided range but *inside* of our interpolation range. The use of Forward and Backward extrapolation together, to interpolate the midpoint of a sliding window, facilitates the method to be neutral regarding movement direction.

5.1.2 The Sliding Window

A *Sliding Window* of size seven, $S_{w,7} = \langle l_{w-3}, l_{w-2}, l_{w-1}, l_{mid}, l_{w+1}, l_{w+2}, l_{w+3} \rangle$, is a segment, defined in Section 2.1 where $l_i \in S_{w,7}$ is a trajectory point defined in Section 2.1, with a minimum of seven trajectory points, by which three new trajectory points are created using interpolation/extrapolation techniques: (i) forward extrapolated point, l^F, (ii) backward extrapolated point, l^B, and (iii) interpolated midpoint, l^C. The indexes are relative for each window so that they can slide over a raw trajectory and represent different windows. The decision to use seven trajectory points on a window was motivated by the fact that it is necessary to use just three points to forward/backward extrapolate in the kinematic extrapolation of a trajectory point immediately before or after them. However, this limitation can be removed when we apply other interpolation methods such as cubic or random walk. For example, since we use forward and backward extrapolations here, we have used three points at the beginning of the Sliding Window to predict forward the position of the fourth point (l^F) and three points at the end of the Sliding Window to predict backward another fourth point (l^B). We expect that this point is very close to the result of the forward extrapolation. Therefore, we are dealing with eight points (three first points, one interpolated midpoint, one actual midpoint, and three last points), hence the name of the Octal Window Segmentation (OWS) method.

We deliberately applied two notations: i) a subscript, l_i, symbolizes the points belong to the trajectory and ii) a superscript, l^i, indicates the points are calculated and do not belong to the trajectory. We then use these two extrapolated trajectory points to create an interpolated midpoint (l^C) and calculate its geographical distance from the actual midpoint, l_{mid}. We use the term *current sliding window* to refer to the sliding window under process by our procedure at a given moment. After processing a sliding window, we move the current sliding window by one trajectory point and process the next sliding window.

5.2 Generating the Error Signal

The distance between an interpolated point and the midpoint of the current sliding window (the error of that window) can be calculated using any distance measure suitable for each specific application. Because we work with Automatic Identification System (AIS) and the Atlantic hurricanes dataset covering more than one geographical zone, we choose to use Haversine distance, which is a very accurate way of computing distances between two points on the surface of a sphere, also known as great-circle or spherical distance which is the shortest distance between two points on the surface of a sphere, using the latitude and longitude of the two points ($haversine(x) = sin^2(\frac{x}{2})$, where x is the central angle [1] between two points on the earth [77]. We remark

[1]A **central angle** is an angle whose apex is the center of a circle and whose sides are intersecting the circle in two distinct points.

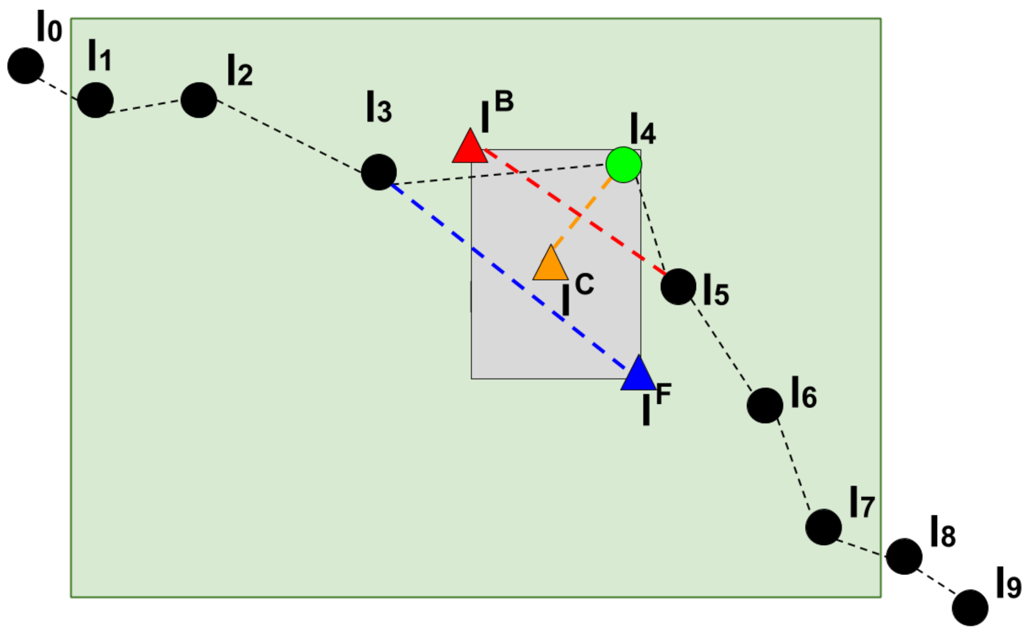


Figure 5.1: An example of error calculation in Octal Window Segmentation where seven trajectory points $(l_1, ..., l_7)$ are selected as the *current octal window*. Here, the midpoint, l_4, assumed as a missing point. l^B and l^F (triangle red and blue points) are generated using extrapolation on first three (l^F) and last three points (l^B). The l^C, triangle orange point, is generated as a middle point of l^B and l^F. The distance between midpoint and l^C is called the *error value* of this octal window.

that the use of haversine distance is a more appropriate measure for trajectory mobility data on the earth since it gives more accurate short distance value between two points on the earth. In cases of a smaller dataset, covering only a geographical zone, the algorithm can be simplified by using the euclidean distance. We set the index of each window equal to the index of the midpoint of that window. When we process a trajectory with n trajectory points, we assume that the error for the $[ws/2]$ points at the start of trajectory and $[ws/2] + 1$ points at the end of trajectory are equal to their calculated adjacent error value in SWS, where ws is the number of trajectory points in the sliding window or window size. The rest of the error values are calculated based on the distance between actual midpoint (l_4) and interpolated midpoint (l^C). Therefore, we generate an array with n elements of errors for a trajectory with n trajectory points, and we call this array *error signal*.

As we discussed earlier, SWS has two major components. The first procedure that composes the Sliding Window Segmentation algorithm is detailed in Algorithm 5.2.

This procedure creates an error signal by sliding a Sliding Window over a raw trajectory of τ_n.

Algorithm 5.2 Generate Error Signal

```
Require: $\tau_n$ {the raw trajectory}
Require: $n$ {number of trajectory points in $\tau_n$}
Require: $ws$ {Size of sliding window (an odd number)}
Ensure: E {Error signal for all trajectory points in $\tau_n$}
1: $mid \leftarrow int(ws/2)$
2: $ws \leftarrow mid * 2 + 1$
3: $E \leftarrow []$
4: $E.append([0] * mid)$ {$A * n$, where $A$ is an array and $n$ is an integer, copies array $A$ for $n$ times. For example, $[0] * 3 = [0, 0, 0]$}
5: for $(i \leftarrow mid; i < n - mid; i++)$ do
6:     Create Sliding Window $S_{ow} = \langle l_{i-mid}, ..., l_{i+mid} \rangle$
7:     $l^F \leftarrow$ extrapolate forward $S_{ow}$
8:     $l^B \leftarrow$ extrapolate backward $S_{ow}$
9:     $l^C \leftarrow$ extract midpoint from $l^F$ and $l^B$, Equation 5.1
10:    $\epsilon_i \leftarrow Haversine(l_i, l^C)$
11:    $E.append(\epsilon_i)$
12: end for
13: $E.append([0] * mid)$
14: $E[0 : mid] \leftarrow E[mid] * mid$ {$A[i : j]$ is a slice of array $A$ from index $i$ to index $j$.}
15: $E[n - mid - 1 : n] \leftarrow E[n - mid - 2] * (mid + 1)$
16: return $E$
```

The procedure of generating error signal starts with an empty array of error of sliding window, E, in line three. In line four, this empty signal set to $[0] * [ws/2]$ (which means an array of $[ws/2]$ of zero elements, e.g. $[0, 0, 0]$ for $ws = 7$) is appended to this array and represents the error for the first $[ws/2]$ trajectory points of the raw trajectory (τ_n). The algorithm explores all the Sliding Windows from lines five to eleven as follows: First, the *Current Sliding Window*(S_{ow}) is created in line six. The *forward extrapolated point* (l^F) is calculated in line seven by performing forward extrapolation. In this method, we assume that l_i in the current sliding window is missing and will be extrapolated using points $l_{i-mid}, .., l_{i-2},$ *and* l_{i-1}. The forward extrapolated point at time $t^i = t_{mid-1} + \frac{t_{mid+1} - t_{mid-1}}{2}$ is called l^F. After, the *backward extrapolation* method calculates the *backward extrapolated point* l^B in line eight. In this method, it is also assumed that l_i, the midpoint of the sliding window, in the *current sliding window* is missing. We reverse the order of points so that points $l_{i+1}, l_{i+2}, .., l_{i+mid}$ are used to extrapolate the point l_i at time $t^i = t_{mid+1} - \frac{t_{mid+1} - t_{mid-1}}{2}$ and the procedure calls it l^B. In line nine, we use l^F and l^B geo-locations to calculate the midpoint, l^C, using Equation 5.1. [2]

$$l^C = l^B + \frac{l^F - l^B}{2} \tag{5.1}$$

We calculate the *error value* (ϵ_i) of the *current sliding window* (S_{ow}) by measuring the distance between l^C and l_i. Then we append ϵ_i to error signal (E). The error signal (E) is finally computed in line 13 by appending $[0, ..., 0]$ with $[ws/2]$ zeros, where the error values are obtained by calculating the haversine distance between l_i and l^C.

In lines 14 and 15, we update the error values assigned to zero at the start and end of the Error Signal (E) with adjacent value to smooth our Error Signal.

The l^B (blue extrapolated trajectory point), l^F (red extrapolated trajectory point), l_4 (green point) and l^C (orange point) are shown in Figure 5.1.

The haversine distance from the estimated trajectory point l^C to the real trajectory point l_4 is displayed by the orange line connecting l^C and l_4 in the example of Figure 5.1. This may indicate that the moving object behavior has changed at position l_4.

An example of an error signal generated by Algorithm 5.2 is shown in Figure 5.2. A raw trajectory (τ_{145}) with 145 trajectory points and eight segments is used in this example. Here are several trajectory points (e.g., around trajectory point 95, or around trajectory point 123) along the raw trajectory where the estimated trajectory points are far from the reported actual trajectory points of the moving object as shown in Figure 5.2.

The complexity of generating the error signal is $O(n)$, where n is the number of trajectory points because the line five of Algorithm 5.2 repeats $n - ws$ times.

[2]Here, we applied Euclidean distance because the difference between using haversine and Euclidean distance was not significant for our datasets. Using haversine increases the amount of calculation that is not in line with our objective. Moreover, the error in the calculating distance is in two opposite directions where may neutralize this error. The average error of calculating distance using the euclidean formula is 0.06%. If two points have 40° distance in their latitude on the longitude of 0°, this average error decreases to 0.002%. For example, the distance from (0,0) to (0,40), using haversine, is 4449 KM. If we use the euclidean distance, this number changes to 4410 KM. Therefore, the inaccuracy of euclidean distance is 39 KM in this example. However, there is no harm using haversine here if you can maintain the computation cost.

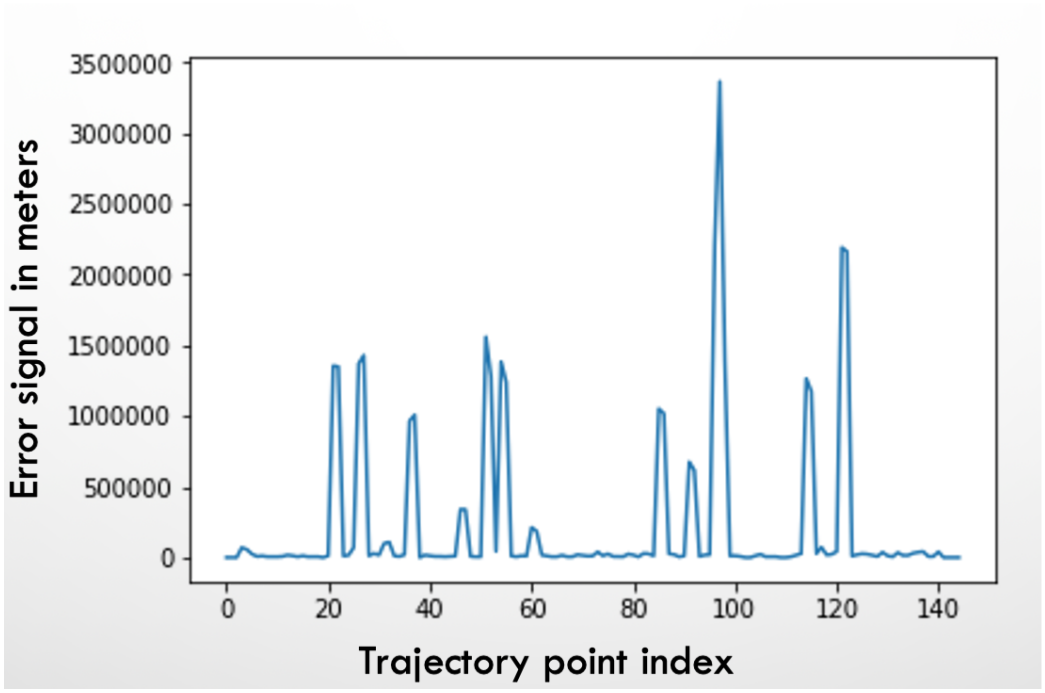


Figure 5.2: This figure shows an error signal generated using 145 trajectory points, which includes eight segments. There are few spikes (e.g., in index 95 and 123) representing a considerable change in error value. We consider them as boundaries of segments.

5.3 The Segmentation Procedure

The intuition behind our algorithm is that when a moving object changes from one behaviour to another, it can be captured directly from its location. To compute an estimated position — where the moving object is supposed to be if its behaviour does not change — we use interpolation methods, which are the marriage of forward and backward extrapolation methods. By evaluating the error signal, it is possible to estimate if the moving object changed its behaviour in a region and use this information to create segments.

The second component of the Sliding Window Segmentation (SWS) algorithm is detailed in Algorithm 5.3, which receives a single ϵ threshold value as input. The intuition behind our trajectory segmentation algorithm is that segments are created by cutting the raw trajectory at *partitioning positions*, where the error values from E are higher than the ϵ threshold value. These *partitioning positions* are created as a list of tuples with the indexes of where segments start and end.

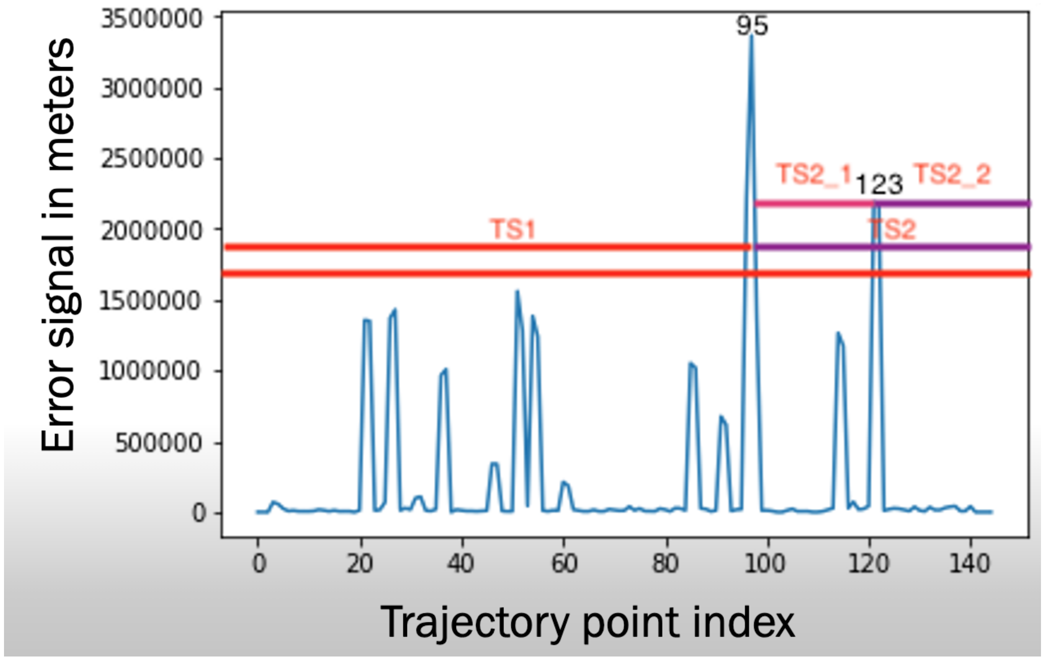


Figure 5.3: This figure shows how Algorithm 5.3 works. First, it detects index 95 and generates two trajectories: (i) $TS1$ and (ii) $TS2$. $TS1$'s error values are lower than our threshold; therefore, we append it to our segments set. We continue processing $TS2$ by detecting index 123. Then, we generate $TS2_1$ and $TS2_2$ and append them to our segments set.

The error signal E is the first processing step of Algorithm 5.3, and the output of the procedure is computed in Algorithm 5.2. In lines two, the algorithm initializes three variables ($first$, q, p). It set the $first$ variable with a zero value, which represents the starting index of the trajectory and creates the first tuple $(first, n)$ that represents the entire trajectory and adds it to the array q. The third variable (p) with all the partitioning positioning tuples is declared as an empty set. As long as the array q is not empty, lines four to 23 are executed. The algorithm fetches the first item from set q and assign it to tuple t. This tuple has the start and end index of the current segment under process. line four. Using indexes in tuple t, the algorithm creates a subset of error signal ($curr$) from index t_0 to t_1, line five. Then it finds its maximal error value m (line six). If this maximal error value is greater than the threshold, ϵ (a parameter that can be adjusted for each domain and changes the sensitivity of the algorithm), the index of m is retrieved, and two new tuples are appended to q if there is a single position with value m (lines 10 to 11). These new tuples are analyzed in the next iteration of the algorithm, which will look for other error values higher than the

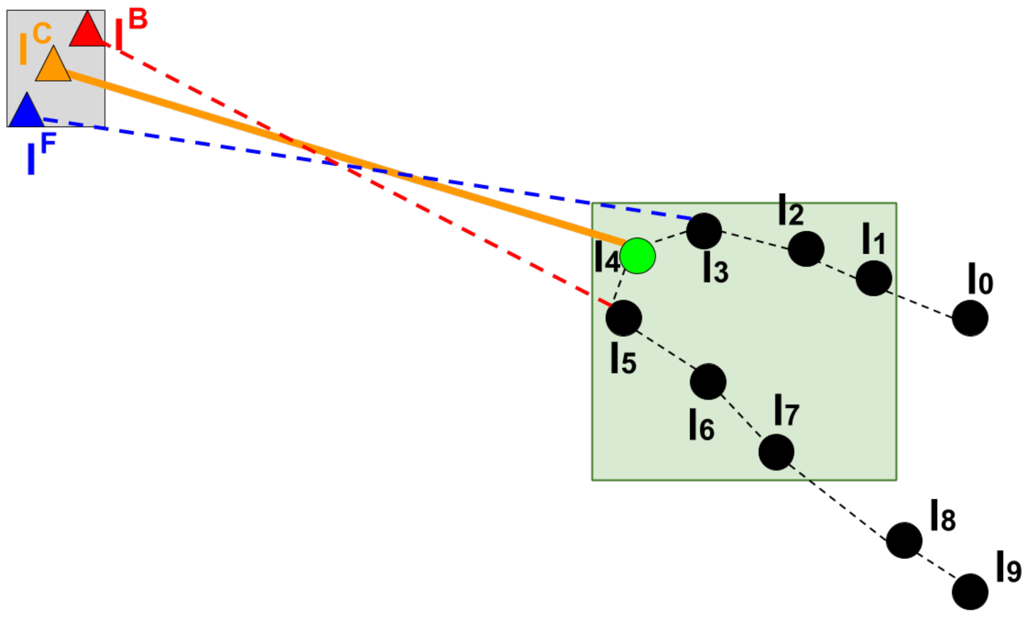


Figure 5.4: An example of error calculation when the moving object changes its behaviour at l_4. The speed changes from l_4 to l_5, l_5 to l_6, and the length of time that moving object spends at the point l_4 influences the error value.

ϵ threshold (ϵ threshold represents *partitioning positions*). If there is more than one partitioning position with a value equal to m (lines 13 to 20), sub-segments on this part will be added to the q for the subsequent iteration processing. Notice that when *length(idx)* is more than one, we have an interval in which the error signal is higher than ϵ. In this situation, we divide this interval into two or more segments by using a function called **find_sub_segments**. This function assigns each half of the interval to a segment after each slope sign changes. The minimum number of segments that can be generated here is two segments. This procedure will run until all the tuples with partitioning positions are created where error values are higher than the error threshold of ϵ. In the worst case, ϵ is less than all error values; this process runs n times. In the line 22, if $m \leq \epsilon$, tuple t is appended to the final list p. In line 25, the p is converted to a point-feature.

A practical example of executing Algorithm 5.3 is shown in Figure 5.3. In the first step, the full input trajectory is considered as one segment. By setting the threshold at 1,600,000, the algorithm finds TS_1 starting from zero to 95 as the first discovered segment that cannot be divided into more segments. Therefore, TS_1 appends to the final list, p. Although the error value of 1,600,000 meters is considered a very long distance for a moving object to travel, we remark that this is an abstract distance and can happen in a situation similar to the following. First, the moving object has a stop at the point it changes its behaviour. Then, the moving object behaviour changes. After that, it continues the movement in a new direction. Figure 5.4 shows this example that a moving object has a three-hour stop at point l_4, followed by a drastic change of its direction.

The algorithm repeats for the TS_2. In this step, the algorithm can find $TS2\_1$ as an unbreakable segment and appends it to p. Then the algorithm repeats for the rest of the trajectory, which is $TS2\_2$. Since this is not a breakable segment, the $TS2\_2$ is appended to the final list and algorithm finishes. Therefore, we segment T into $TS\_1, TS2\_1, TS2\_2$. Here our variable p is equal to $[(0, 95), (96, 123), (124, 145)]$.

Assuming our ground truth are the following.

$$SID = [(0, 21), (22, 27), (28, 51), (52, 55), (56, 94), (95, 114), (115, 123), (124, 145)]$$
$$Label = [Stop, Move, Stop, Move, Stop, Move, Stop, Move]$$

We can evaluate the performance of algorithm by calculating *purity*, *coverage*, and harmonic mean as explained in Section 4.2. For calculating C, we average the *coverage* of discovered segments and we average the *purity* of ground truth segments for P. Therefore,

$$\begin{aligned} C &= \frac{0.4 + 0.7 + 1}{3} && = 0.7 \\ P &= \frac{1 + 1 + 1 + 1 + 0.97 + 1 + 1 + 1}{8} && = 0.99 \\ H &= \frac{2 * 0.7 * 0.99}{0.7 + 0.99} && = 0.82 \end{aligned}$$

Algorithm 5.3 Segmentation Procedure

```
Require: $\epsilon$ {by default $Percentile_{95}(E)$}
Ensure: $segment\_id$
1: $E \leftarrow$ Generate Error Signal ($\tau_n$)
2: $q \leftarrow [(0, n)], p \leftarrow \emptyset$
3: while $q \neq \emptyset$ do
4:     $t \leftarrow q.pop()$ {get a tuple from q}
5:     $curr \leftarrow E[t[0] : t[1]]$ {get error signal of the current trajectory}
6:     $m \leftarrow max(curr)$ {find max error of the current trajectory}
7:     if $m \geq \epsilon$ then
8:         $idx \leftarrow index(curr == m)$ {return the index of $\epsilon$ where $curr == m$}
9:         if $length(idx) \leq 1$ then
10:            $q.append((t[0], t[0] + idx[0]))$
11:            $q.append((t[0] + idx[0] + 1, t[1]))$ {break a trajectory to two segments}
12:        else
13:            $mp, mq \leftarrow find\_sub\_segments(curr, \epsilon)$
14:            for all $i \in mp$ do
15:                $p.append(i)$
16:            end for
17:            for all $i \in mq$ do
18:                $q.append(i)$
19:            end for
20:        end if
21:    else
22:        $p.append(t)$ {t is a desired segment, appended to final set of segments}
23:    end if
24: end while
25: $segment\_id = generate\_segment\_id(p, \tau)$ {$generate\_segment$ is a function to con-
    vert the $p$ to a column of size $n$. for example, for $p = [(0, 2), (2, 4)]$ it returns
    $[1, 1, 2, 2]$}
26: return $segment\_id$
```

5.3.1 Parameter Selection

SWS algorithm (Algorithm 5.1) has four parameters: interpolation kernel (*kernel*), size of the sliding window (ws), ϵ threshold value, and a flag ($flag$). In this chapter we always set $flag = True$. This tells the algorithm to return the segment results.

The interpolation kernel can be chosen based on each domain. We have experimented with five interpolation kernels in our experiments and suggest the best interpolation for each dataset in Section 5.4.2. We suggest a parameter tuning to find the proper kernel for the domain if we have access to labelled data. We suspect random walk performs well for domains with many variations of high and low speed. We assume the kinematic extrapolation generates higher-quality segments for a dataset that includes only high-speed moving objects. For the movement in the water, such as vessel movement, we expect cubic interpolation performs well according to Zhang et al. [83]. For unknown situations, we assume a linear interpolation as a default kernel because of its simplicity and generality.

We conducted a few experiments on three datasets to find the best sliding window size. Our experiments were repeated for a window size of 5, 7, 9, 11, and 13 for cubic, random walk and linear regression kernels. The linear kernel has a limitation of window size three by definition. Also, the kinematic interpolation has a restriction of sliding window size seven by definition. The results of these experiments are reported in Section 5.4.3. We assume the default value of seven for the sliding window when we cannot tune this parameter.

Having a fixed ϵ threshold for a domain can be a difficult decision. We suggest a parameter tuning step for calculating the best ϵ for each dataset. In this parameter tuning approach, we calculate an array of candidate ϵ by calculating percentiles 90, 91, ..., 99 of the error signal and testing them on a labelled dataset to measure the best performance.

A design choice is to select percentile 95 of the error signal as the epsilon threshold for default value of this parameter. We suggest this configuration because percentile 95 of error signal produced reasonably good quality segments for most of our tests. We should consider the fact that the quality of data and the number of GPS jumps and gaps can affect the best ϵ value.

5.4 Experimental Evaluation

We define the structure of our experiments in next section. Then we experiment to compare the five defined kernels for SWS, including linear interpolation, kinematic, random walk, linear regression, and cubic kernels. The results of these experiments are presented in Section 5.4.2. Then we conduct another experiment to analyze the effect of window size using three kernels that can receive a variable sliding window: linear regression, random walk, and cubic kernels. The results of these experiments are discussed in Section 5.4.3. After that, we compare our SWS, with the best kernel and window size, against CB-SMoT, GRASP-UTS, SPD, and WKMeans, discussed in Section 5.4.4. We evaluate our method and compare it with other methods using four metrics:

1. Purity, coverage and harmonic mean, discussed in Section 5.4.5

2. Number of segments generated, addressed in Section 5.4.6

3. Amount of memory and CPU time used by segmentation algorithm, discussed in Section 5.4.7

4. V-measure which is a clustering metric, discussed in Section 5.4.8

5.4.1 Experiment Setup

We applied the same experimental protocol on all datasets and segmentation methods to conduct our experiments. This experimental protocol defines the structure of all of our experiments. Each experiment has four requirements: 1) Dataset, 2) Segmentation algorithm, 3) Tuning parameters, 4) Metrics. Dataset for each experiment is prepared in ten folds[3], $D = [\tau_1, ..., \tau_{10}]$, each of which, τ_i, includes a raw trajectory which is called *data* and two ground truth, $SegmentID$ and $label$, which are features for that trajectory. In lines 3 and 4 of experimental protocol, presented in page 72, we select one fold as $TuningSet$ and the rest of folds together as $TestSet$. Here, we make sure that combining the folds in $TestSet$ does not generate time collision[4].

Trajectory segmentation algorithms such as GRASP-UTS, SPD, CB-SMoT, WKMeans, and SWS do not use a training dataset. These algorithms tune their parameters using a small dataset, which is called $TuningSet$. A default value can be set for each parameter, and the algorithm will provide segments. However, the quality of segments can be increased by tuning the parameters.

[3]The fishing and Atlantic hurricanes datasets are obtained from GRASP-UTS research, and they come in ten folds. The evaluations in this thesis are based on these ten folds. We use the same folds to be consistent with their work.

[4]When two objects move simultaneously, their trajectory cannot be processed by an algorithm designed for a single moving object because when we sort based on time, the trajectory points of two moving object mixes. Therefore, each object's movement must not happen at the same time that other object moves. The fishing dataset is an example of a dataset with multiple vessels moving simultaneously. For processing such a dataset, we serialized multiple vessels' movement over time so that there is no time collision.

Regarding the question if SWS is a lazy algorithm because of not using training data, a lazy algorithm has three characteristics: 1) defer processing inputs until they receive a query, 2) they answer to the questions by combining their stored data 3) they discard the constructed answer and any intermediate results [2]. SWS does not defer the processing of its input to query time. Therefore, our proposed algorithm is not a lazy algorithm.

Segmentation Algorithm, *TS_algorithm*, is a function that has two methods: 1) tuning, 2) segment. The tuning method receives the *TuningSet* and the *Tuning Parameters*, line five of the experimental protocol. This method returns the best parameters, *BestParam*, that can generate the highest harmonic mean of purity and coverage. The *segment* method receives the *TestSet* and the best parameters, *BestParam*, and returns the generated segment ID for the *TestSet*, line 6 in experimental protocol. The report function, in line 7 of experimental protocol, receives the ground truth for *TestSet*, the generated segment identifier, *SegmentId*, and a metric, *Metric*, to measure the metric and returning the value of the metric. We have some milestones in the experimental protocol that we capture the amount of memory and CPU time used in each experiment, lines 1, 8, 10. Therefore, each experiment is a call of experimental protocol with the experiment parameters.

Experimental protocol Structure of our experiments

```
Require: $D = [\tau_1, ..., \tau_{10}]$ {The raw trajectory in ten folds}
Require: $TS\_algorithm$ {Segmentation algorithm under experiment}
Require: $TuningParameters$ {All parameters required by the algorithm}
Require: $Metric$ { Performance metric for trajectory segmentation}
Ensure: $Results$
 1: $Record(CPUTime, Memory)$
 2: for $t_i \in D$ do
 3:     $TuningSet \leftarrow t_i$
 4:     $TestSet \leftarrow D - t_i$
 5:     $BestParam \leftarrow TS\_algorithm.tuning(TuningSet, TuningParameters)$
 6:     $SegmentId \leftarrow TS\_algorithm.segment(TestSet.data, BestParam)$
 7:     $Results \leftarrow Report(TestSet.GroundTruth, SegmentId, Metric)$
 8:     $Record(CPUTime, Memory)$
 9: end for
10: $Record(CPUTime, Memory)$
11: return $Results$
```

SWS kernel	Median	Linear Regression	Linear	Random Walk	Cubic	Kinematic Kernel
Linear Regression	91.36	1.0	0.496	0.364	**0.005**	0.150
Linear	91.05	0.496	1.0	0.198	**0.002**	0.082
Random Walk	84.65	0.364	0.198	1.0	0.096	0.364
Cubic	**80.59**	**0.005**	**0.002**	0.096	1.0	0.705
Kinematic Kernel	81.94	0.150	0.082	0.364	0.705	1.0

* ws=window size

Table 5.1: The p-value results of the Wilcoxson test to compare different kernels for the SWS algorithm on the fishing dataset with window size seven for linear regression, random walk, cubic, and kinematic kernel and window size three for the linear kernel. The p-values with significant differences are highlighted in bold.

5.4.2 Evaluation of SWS Kernel Method

In this experiment, we aim to find the best kernels of the SWS algorithm for each dataset. The performance of SWS using cubic, linear, kinematic, random walk, and linear regression kernels on the fishing dataset is compared in Figure 5.5. As we noted that cubic interpolation is expected to produce higher-quality segments for the movement in the water, according to Zhang et al. in [83], we expected high performing results for the cubic kernel on the fishing dataset. However, this kernel did not produce the best *harmonic mean*. The reason is that the fishing dataset includes two activities: 1) Fishing 2) Non-Fishing. The Non-Fishing part is a movement or stop in the water; however, the fishing activity is a movement controlled by some external variables such as the movement of fish. The linear regression kernel gained the highest median *harmonic mean*, 91.3%. Linear interpolation gained the second-best performance by 91.0% median of *harmonic mean*. Random walk kernel produced reasonable results for *harmonic mean*. The random walk kernel can simulate a kinematic interpolation when the speed of the moving object is high, by decreasing the variation of direction. This helps random walk to provide high-quality results in cases that the dataset includes slow movements and fast movements.

The results of Wilcoxon tests[5]comparing different kernels on the fishing dataset is

[5]We are interested in comparing the results of our experiments for each trajectory segmentation algorithm when using the same datasets. Each measurement is repeated for ten folds. Thus we have

shown in Table 5.1. The median *harmonic mean* in SWS with linear regression kernel and a window size of seven was 91.36% ($IQR^6 = 2.18$), whereas the median in the cubic kernel with a window size of seven was 80.59% ($IQR = 7.53$). The Wilcoxon test showed that the difference between SWS with linear regression kernel and cubic kernel was significant ($p = 0.005$, $statistic = -2.79$). Moreover, the Wilcoxon test indicated that the difference between SWS with linear kernel and cubic kernel was significant ($p = 0.002$, $statistic = -3.02$). The Wilcoxon test showed no significant difference between linear regression kernel and linear kernel ($p = 0.496$, $statistic = 0.68$).

The experiment results, shown in Figure 5.5 and Table 5.1, suggest that **linear interpolation** and **linear regression kernels** provided more robust results with less variance on **the fishing dataset**. Although we expected to see the best performance on the fishing dataset using the cubic kernel for in water movements [83], but the cubic kernel did not surpass the linear regression. We think this is because the behaviours of moving objects in this dataset are a mixture of fishing and non-fishing (anchoring or voyaging), not purely in water movement.

The results of Wilcoxon tests to compare SWS kernels on the Atlantic hurricanes dataset is displayed in Table 5.2. The median *harmonic mean* in SWS with the cubic kernel and window size of seven was 87.90% ($IQR = 0.91$), whereas the median in the kinematic core with a window size of seven was 82.83% ($IQR = 1.36$). The Wilcoxon test showed that the difference between SWS with cubic kernel and linear regression kernel was significant ($p = 0.004$, $statistic = 2.87$). This test indicates that the kinematic kernel generates statistically significant lower *harmonic mean* results in comparison to all other methods on the Atlantic hurricanes dataset.

The performance of SWS using cubic, linear, kinematic, random walk, and linear regression kernels on the Atlantic hurricanes dataset is compared in Figure 5.6.

ten values for each measurement. To compare two distributions of these measurements, we can use a parametric test such as T-Test or a nonparametric test such as Mann-Withney's or Wilcoxon's test. Nonparametric tests are distribution-free tests because they don't assume that the data follow a specific distribution. We choose a nonparametric test because we do not have a piece of evidence that our measured values follow a specific distribution. Additionally, we must identify whether our samples are independent or paired. When each value in one set of measurements is uniquely matched to a data point in another set of measures, we have paired sample data. In our study, each measured value can be uniquely paired with another measured value, because we reported the measurement for the same tuning set and test set. Consequently, We select a nonparametric paired sample test to compare our results, which is the Wilcoxon rank-sum test.

[6]interquartile range: the difference between 75th and 25th percentiles. $IQR = Q_3 - Q_1$

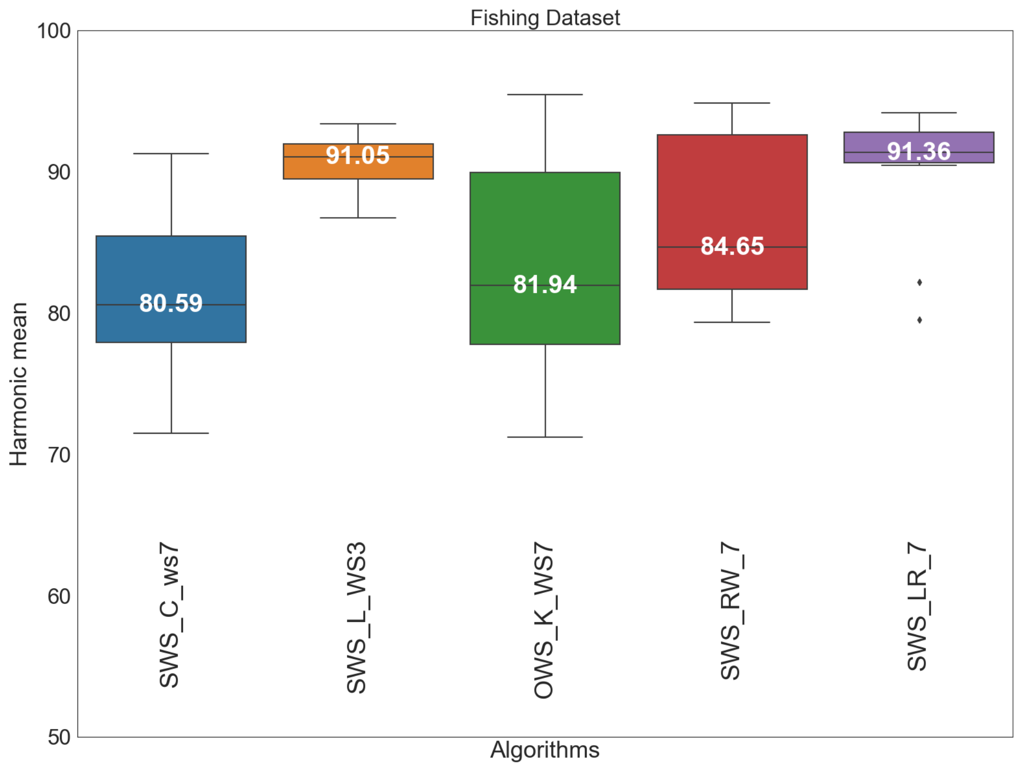


Figure 5.5: This figure compares different kernels of the SWS Algorithm on the fishing dataset. The results suggest that **linear regression (SWS_LR_7)** and **linear interpolation (SWS_L_WS3)** kernels are performing better than the random walk(SWS_RW_7), cubic (SWS_C_7) and kinematic (SWS_K_7) kernels on the fishing dataset.

The experiment results, shown in Figure 5.6 and Table 5.2, suggest that kinematic interpolation gained the lowest performance among all experimented kernels. The experiments suggest that the cubic kernel gained the highest *harmonic mean* (87.8%) in comparison to linear interpolation (86.6%), random walk (86.3%), linear regression (86.4%), and kinematic kernel (82.8%).

The performance gain using cubic kernel is because the level of hurricanes does not follow the kinematic rules of moving objects. We observed that the cubic interpolation could be a better representation of hurricanes' movement than kinematic interpolation. We think this is because the behaviours of hurricanes follow more curve patterns. Also, the sampling rate in this dataset is every 8 hours on average. As future work, we think changing the sampling rate can improve the performance of the trajectory segmentation task.

SWS kernels	Median	Linear regression	Linear	Random walk	**Cubic**	Kinematic Kernel
Linear regression	86.42	1.000	0.820	0.545	**0.004**	**0.001**
Linear	86.67	0.820	1.000	0.762	0.015	0.082
Random walk	86.39	0.545	0.762	1.000	**0.001**	**0.000**
Cubic	87.90	**0.004**	**0.015**	**0.001**	1.000	**0.000**
Kinematic kernel	82.83	**0.001**	0.082	**0.000**	**0.000**	1.000

Table 5.2: The p-value results of the Wilcoxson test to compare different kernels for the SWS algorithm on the Atlantic hurricanes dataset with window size seven for linear regression, random walk, cubic, and kinematic kernel and window size three for the linear kernel. The p-values with significant differences are highlighted in bold.

SWS kernels	Median	Linear Regression	Linear	Random Walk	Cubic	**Kinematic Kernel**
Linear Regression	91.20	1.000	0.939	1.000	0.879	**0.028**
Linear	89.69	0.939	1.000	0.762	0.596	**0.028**
Random Walk	89.56	1.000	0.762	1.000	0.596	**0.049**
Cubic	91.98	0.879	0.596	0.596	1.000	**0.049**
Kinematic Kernel	**93.86**	**0.028**	**0.028**	**0.049**	**0.049**	1.000

Table 5.3: The p-value results of the Wilcoxson test to compare different kernels for the SWS algorithm on the Geolife dataset with window size seven for linear regression, random walk, cubic, and kinematic kernel and window size three for the linear kernel. The p-values with significant differences are highlighted in bold.

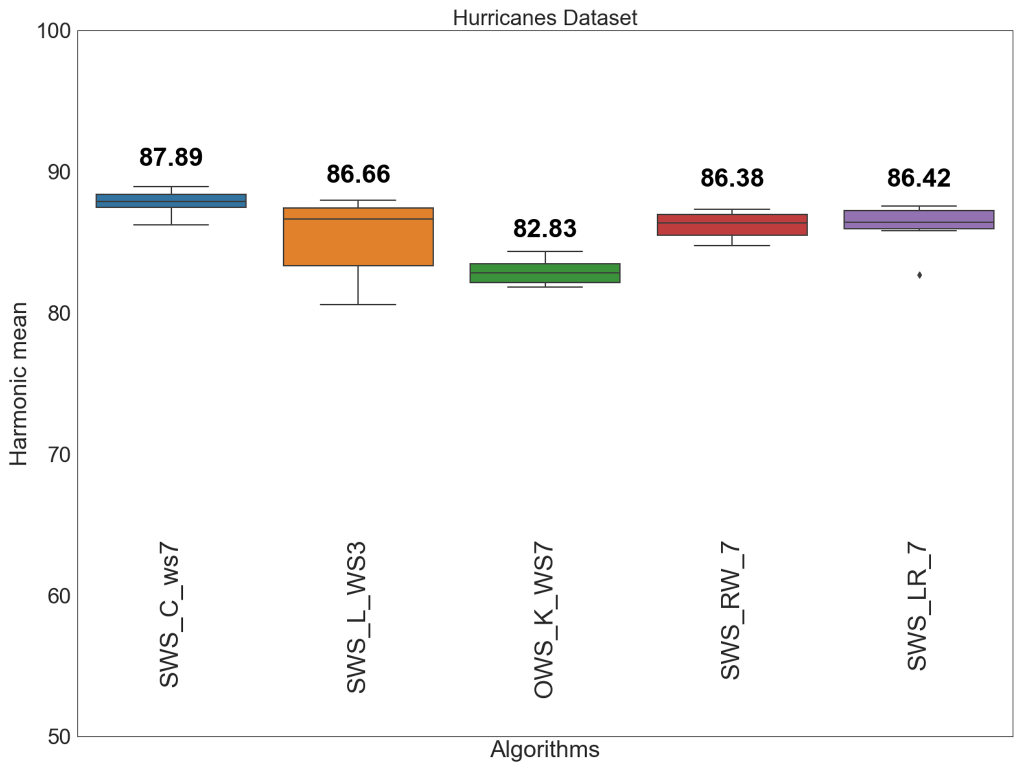


Figure 5.6: This figure compares different kernels of SWS algorithm on the Atlantic hurricanes dataset. The results suggest that cubic kernel (SWS_C_ws7) performs better than random walk (SWS_RW_7), linear interpolation (SWS_L_ws3), linear regression (SWS_LR_7), and kinematic (SWS_K_WS7) kernels.

The results of Wilcoxon tests on the Geollife dataset to compare different kernels is presented in Table 5.3. The median *harmonic mean* in SWS with a kinematic kernel with window size seven was 93.86% ($IQR = 2.17$), whereas the median in random walk kernel and a window size of seven was 89.56% ($IQR = 4.18$). The Wilcoxon tests showed that the difference between SWS with kinematic kernel and random walk kernel was significant ($p = 0.049$, $statistic = -1.96$). These tests indicate that there is no statistically significant difference between the use of random walk kernel compared to linear regression, as well as linear and cubic kernels on the Geolife dataset.

The performance of SWS using cubic, linear, kinematic, random walk, and linear regression kernels on the Geolife dataset is compared in Figure 5.7. The experiment results, shown in Figure 5.7 and Table 5.3, indicate that kinematic kernel gained the highest performance by the *harmonic mean* median of 93.8%. Kinematic kernel, along with the cubic kernel (91.9%) provides the lowest variation in the result of this experiment. Linear (89.6%), random walk (89.5%) and Linear regression (91.1%) had more volatile results. We think the kinematic kernel's superiority is because of the properties of this dataset, which is labelled by transportation modes. In this dataset, most of the moving objects follow the kinematic extrapolation.

The kinematic kernel generated the highest *harmonic mean* of purity and coverage on the Geolife dataset because the moving objects in the Geolife dataset follow the kinematic rule of movement, and this makes a considerable difference between this kernel and other kernels.

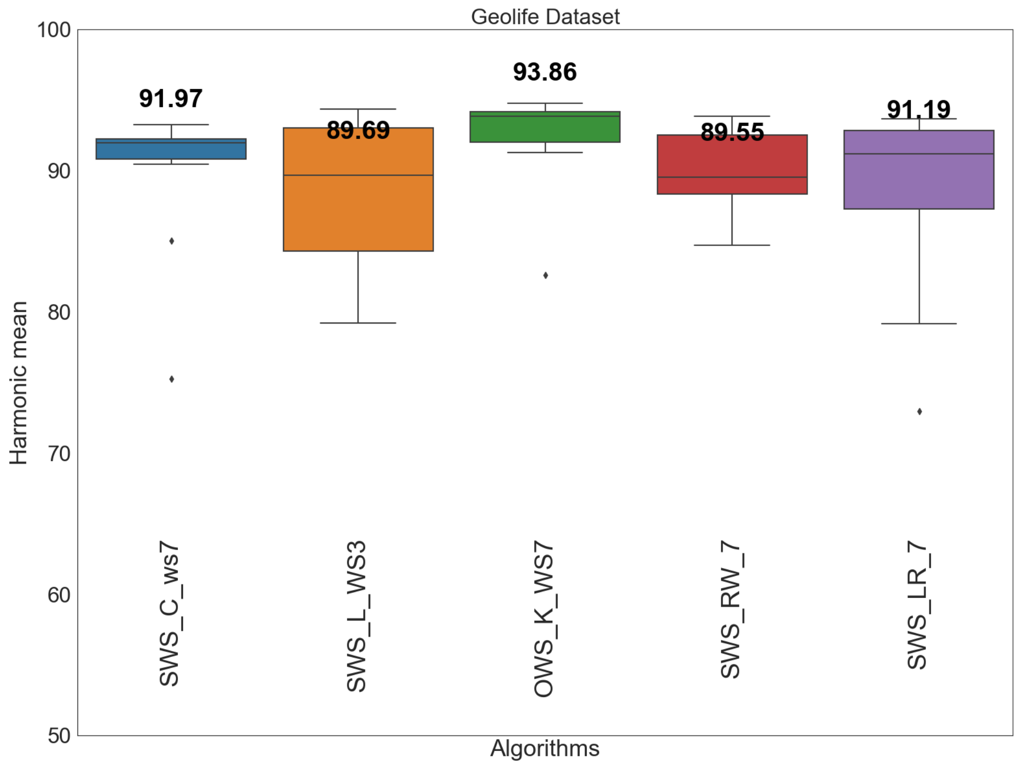


Figure 5.7: This figure compares different kernels of the SWS algorithm on the Geolife dataset. The results suggest that kinematic kernel (SWS_K_WS7) performs better than a random walk (SWS_RW_7), linear interpolation (SWS_L_WS3), linear regression (SWS_LR_7), or cubic kernel (SWS_C_ws7).

5.4.3 Evaluation of SWS Window Size

Among our five defined kernels, three of them have the capability of having sliding windows with different sizes. The kinematic core has to apply only window sizes of seven (three points on each side to calculate acceleration and one mid point), according to definition in Section 2.2.3. Linear interpolation has to use sliding windows of size minimum three, according to definition in Section 2.2.1. More points in linear interpolation does not add value. We compared cubic, random walk and linear regression with sizes of 5, 7, 9, 11, and 13 on the fishing dataset, Atlantic hurricanes dataset and Geolife dataset. Figure 5.8 shows the results of this experiment on the fishing dataset. We observed by increasing the window size from 5 to 13 in the cubic kernel, the performance of the algorithm on the fishing dataset, the median of *harmonic mean*, increases from 77.7% to 85.9%.

The random walk kernel results indicated that window size five gained the best performance, median of *harmonic mean*, by 93.6% on the fishing dataset. Increasing the window size for this kernel did not improve the performance.

The performance of SWS with Linear regression kernel increased from 91.3% to 93.4% when we increased the window size from 7 to 13 on the fishing dataset.

Although we identify some performance-boosting by changing the size of the sliding window, there is no linear correlation between the sliding window size and the performance of SWS on the fishing dataset, $R_{C,ws} = 0.23, R_{LR,ws} = 0.13, R_{RW,-0.26}$ where R is the Pearson correlation.

Fine-tuning plays an important role in getting the best results since there is no linear correlation between window size and the performance of trajectory segmentation. Despite no linear correlation, the results of distance correlations [30] suggest some non-linear correlation for cubic and random walk kernels, $dcor_{C,ws} = 0.76, dcor_{LR,w} = 0.39, dcor_{RW,ws} = 0.86$ on the fishing dataset. This evidence supports that a fine-tuning on window size can identify the best window size for this dataset. The best window size may vary since the rate of sampling in each dataset can vary.

The results of our experiment for different window sizes on the Atlantic hurricanes dataset is displayed in Figure 5.9. These results suggest that by increasing the window size, the performance of this algorithm decreases. We think this decrease is because the window size of five covers 41.55 hours of a hurricane on average, and window size 13 covers 108.03 hours of a hurricane. Changes in the behaviour of hurricanes do not stay for that long. Therefore, the performance of the algorithm decreases.

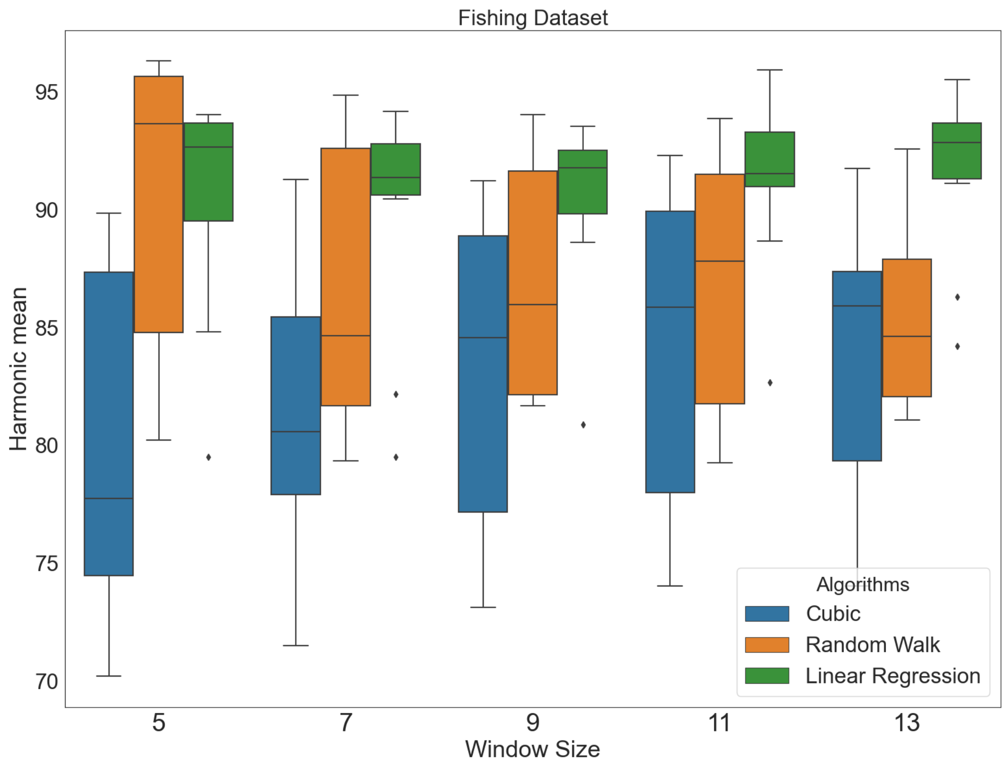


Figure 5.8: This figure compares different sliding window sizes of SWS Algorithm on the fishing dataset. The results suggest that the performance of linear regression kernel increases by increasing the window size.

Pearson correlation test indicates that there is a negative linear correlation between performance and window size using random walk kernel on the Atlantic hurricanes dataset, $R_{RW,ws} = -0.71$. There is no linear correlation between the window size and the performance of SWS on the Atlantic hurricanes dataset using cubic and linear regression kernels, $R_{C,ws} = -0.24, R_{LR,ws} = 0.12$.

Similar to the fishing dataset, sliding window size adjustment on the Atlantic hurricanes dataset is highly related to the attributes of the dataset, such as the sampling rate, and percentages of gaps. We should prudently select a window size so that we expect all the trajectory points inside the sliding window are roughly related.

Our experiments on different window sizes on the Geolife dataset show that all kernels benefit from increasing the window size to some extent. By increasing the sliding window size from 5 to 13, our cubic kernel performance improved from 91.6% to 93.1%, and its standard deviation decreased from 5.8 to 3.9 on the Geolife dataset. This decrease in standard deviation suggests more robustness for the method. Increasing the

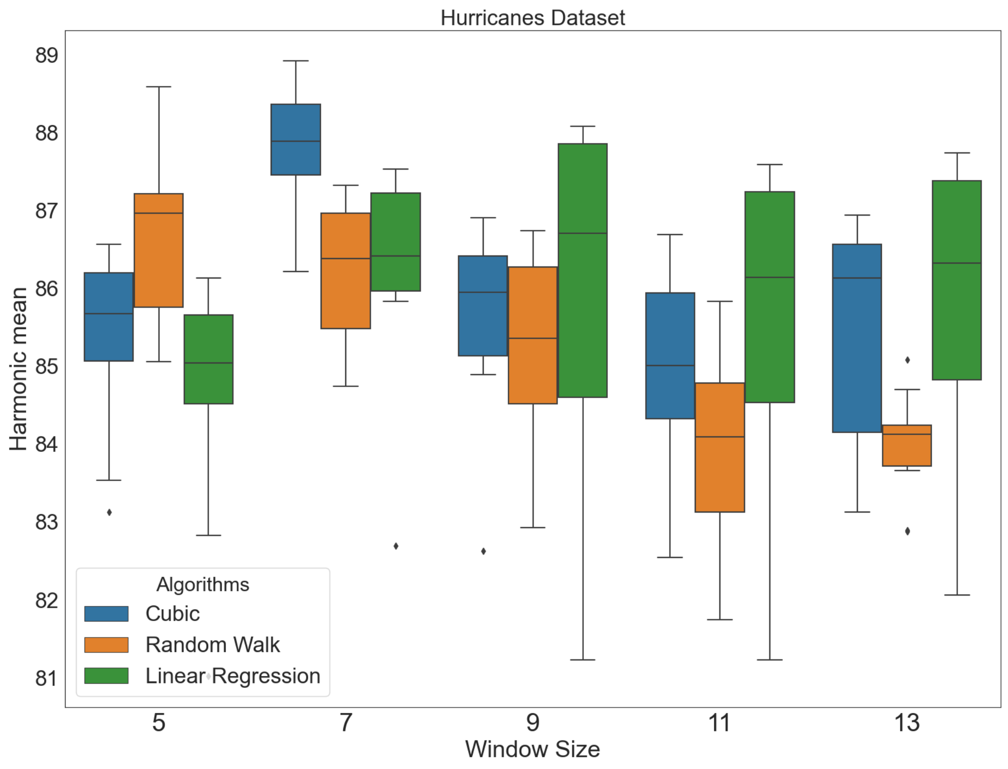


Figure 5.9: This figure compares different sliding window sizes of SWS algorithm on the Atlantic hurricanes dataset. The results suggest that the performance of SWS decreases by increasing the window size.

window size from 5 to 13 using the random walk kernel, improved the performance from 89.8% to 90.9% and decreased standard deviation from 4.8 to 3. Moreover, increasing the window size from 5 to 11 for linear regression kernel leads to boosting the performance from 92.7% to 93.5%. By increasing the window size to 13 in linear regression, we observed a drop in performance from 93.5% to 91.5%. Pearson correlation test indicated that there is no strong linear correlation between performance of SWS and window size on the Geolife dataset, $R_{RW,ws} = 0.16, R_{C,ws} = 0.16, R_{LR,ws} = 0.10$.

Every 22.73 minutes on average, we capture a new trajectory point on the Geolife dataset, while this rate is 105.72 minutes on the fishing dataset and 498.72 minutes on the Atlantic hurricanes dataset. Increasing window size on the Geolife dataset showed some improvements because by increasing the sliding window size, we add more relevant knowledge to our trajectory segmentation algorithm. If we expand the sliding window size more than a threshold, the performance of the trajectory segmentation task decreases because of participating more irrelevant points in segmentation.

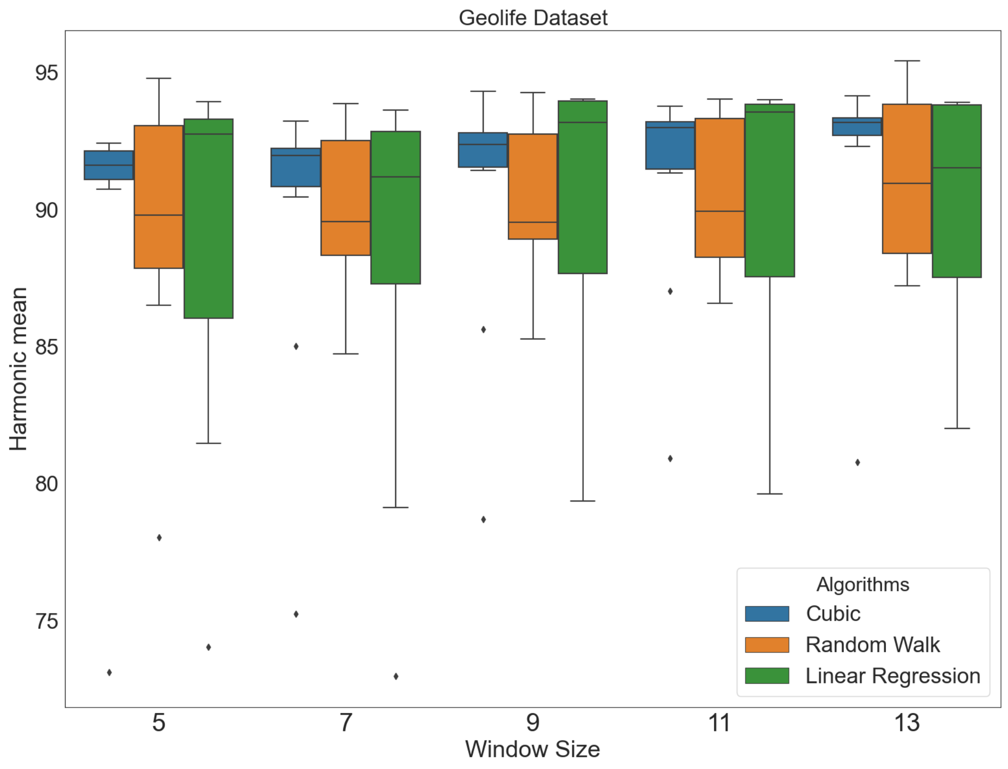


Figure 5.10: This figure compares different sliding window sizes of the SWS algorithm on the Geolife dataset. The results suggest that all kernels benefit from increasing the window size to some extend.

To evaluate the effect of the sampling rate on the performance of the trajectory segmentation task, we conducted the following experiment. Using the results of linear regression, cubic and random walk for five window sizes of 5, 7, 9, 11, and 13, kinematic and linear kernels (17 median harmonic mean, shown in gray lines in Figure 5.11), we calculated the average performance of SWS for each dataset, the fishing dataset 86.8%, the Atlantic hurricanes dataset 85.3%, the Geolife dataset 90.0%, shown in blue lines in Figure 5.11. Figure 5.11 shows the performance of the trajectory segmentation algorithm on each dataset and the average period of capturing data in each dataset. Atlantic hurricanes dataset has the most prolonged period of capturing data, on average, about 8 hours with the highest variance among experimented datasets shown in red areas in Figure 5.11. The blue line shows the calculated average performance in terms of *harmonic mean* of purity and coverage. The red colour area gives an idea of the variation of the period of capturing for each dataset, $0.001 * std$.

We applied Pearson product-moment correlation between the performance of SWS (median *harmonic mean*) on each dataset and the average period of capturing data on each dataset.

The results, $R_{Performance,PeriodOfCapturing} = -0.84$ (A strong negative correlation), indicated that an increase in the period of capturing results in a decrease in trajectory segmentation and a reduction in the period of obtaining new trajectory points increases the performance of SWS. This is because by increasing the time between capturing a new point, the moving object behaviour can be more unrelated to the previous point.

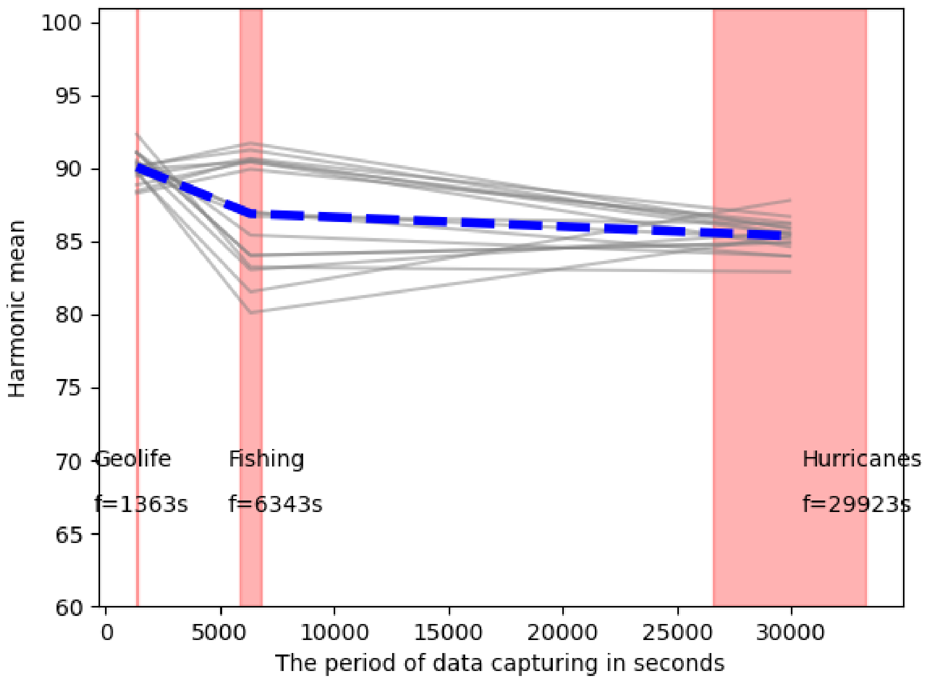


Figure 5.11: When comparing the performance of SWS on three datasets with a different period of capturing, the performance of SWS increases by decreasing the sampling rate in each dataset. The gray lines show the results of experiments on each dataset with different parameters. The blue line shows the average harmonic means on each dataset. The red area represents the variation of period of capturing in each dataset.

Comparing the results of experiments on window size and the period of capturing showed that by decreasing the period of capturing new trajectory points, the increase of sliding window size has a more positive effect on *harmonic mean* measure.

5.4.4 Comparison With Other Algorithms

In this section, we compare the SWS algorithm with four other trajectory segmentation algorithms. We selected GRASP-UTS [69], CB-SMoT [58], SPD [85], and WKMeans [44] for the following reasons. First, SWS is working on a single sensor trajectory. Therefore, we did not include multi moving object trajectory segmentation algorithms such as TRACLUS. Based on extractable knowledge for trajectory segmentation algorithms, we select SPD and CB-SMoT from the category of algorithms searching for interesting points, detailed in Table 3.2 in page 33. GRASP-UTS were chosen from the group of Homogeneous Segmentation and WKMeans from the class of algorithms searching for similar pattern. Another reason that we include GRASP-UTS was that the type of features applied in this algorithm is *Semantic*, details in Section 3.2.2. WKMeans requires the number of segments. These two algorithms receive more information from their environment to work, and we expect using more information provides more robust results and makes competition for SWS harder. We choose SPD because it is designed for a transportation mode domain. We assume SPD beats other algorithms on the Geolife dataset. CB-SMoT also requires the calculation of speed as a point feature. This algorithm benefits from the sequential order of trajectories, and we expect it to produce high-quality segments quickly and efficiently. GRASP-UTS was excluded from the experiment on the Geolife dataset because it did not generate segments after waiting three days while other algorithms produced their segments in the magnitude of seconds or minutes. First, we explain the details of parameter tuning for each algorithm. Then we use statistical analysis to compare their performance in terms of *harmonic mean*.

SWS has three parameters to adjust: 1) kernel, 2) window size, and 3) ϵ-threshold. The $flag$ parameter (forth parameter of SWS) is set to $True$ in this chapter. We selected the best kernel for SWS among Linear (L), Kinematic (K), Random Walk (RW), Linear Regression (LR), and Cubic (C) kernels, detailed in Section 5.4.2. Linear regression gained the best performance for the fishing dataset and the Geolife dataset. The best core for the Atlantic hurricanes dataset was the cubic kernel. We fixed this parameter for testing all folds. We did not tune this parameter for each fold and applied the same setting for all folds, τ_i in experimental protocol.

For adjusting the size of the sliding window parameter, we explored window sizes of 5, 7, 9, 11, and 13 for linear regression, cubic and random walk kernels, addressed in Section 5.4.3. For the linear interpolation kernel, we used only a sliding window size of three, and for the kinematic kernel, we used a window size of seven. The best window size for the fishing dataset was window size of 13, and the best window size for the Atlantic hurricanes dataset was window size of seven. The best window size for the Geolife dataset was 11. We fixed this parameter (window size) for testing all folds. We adjusted the ϵ threshold using the tuning set by percentiles of 90, ..., 99 of the tuning set error signal. This parameter was tuned for testing each fold. Table B.1 shows the parameters applied on the fishing dataset for each fold. This table shows that the only variable parameter of SWS, ϵ-threshold, is in a minimal range of [97,99].

In order to adjust parameters for the CB-SMoT, we tuned the best values for $AreaParam$ of $0.1, 0.3, 0.5, 0.7, 0.9$, and $MinTimeParam$ of $60, 360$, and 3600 seconds. We defined $TimeToleranceParam$ to zero, and we set $MergeToleranceParam$ to Zero. Since we tuned $AreaParam$ we set $MaxDistParam$ to None. We gave the algorithm freedom of changing all the parameters for each tuning fold, and we did not limit the algorithm to fix any of these parameters. This is to make sure we compare our method with the best possible results of CB-SMoT. The parameters of CB-SMoT were selected from a wide range of $[0.01, 0.9]$ for $Area$ and $[360, 3600]$ for min_time. Discovering the best parameters for this algorithm in an unknown setting can be challenging because of the high variance of parameters on the same dataset.

GRASP-UTS was tuned for $PartitioningFactor$, $MaxIterations$, $MinTime$, and JCS. We provided distance and speed as two extra features for this algorithm. We provided wind speed as a piece of extra information for testing the Atlantic hurricanes dataset using GRASP-UTS. This algorithm execution time was very long that it did not produce results for the Geolife dataset after working for three days. This long processing time is not acceptable for a pre-processing task. Similar to CB-SMoT, the best parameters of GRASP-UTS on the same dataset (fishing) came from wide ranges of $[0.3, 0.7]$ for $alpha$, $[6, 360]$ for min_time, $[0.3, 0.7]$ for JCS, and $[10, 30]$ for $MaxIterations$. This can imply the possibility of over-fitting or limitation in generalization.

The SPD algorithm was tuned for $ThetaDistancePara$ and $ThetaTimeParam$.

$ThetaDistancePara$ was adjusted using 100, 200, 500, 1000, 6000, and the $ThetaTime$ $Param$ was tuned using 60, 300, 600, and 2000. We gave the algorithm the freedom to choose the best parameter for each fold. The best parameters of SPD selected for each fold showing a high variance in tuning the parameters, $[60, 2000]$ for $ThetaTime$ $Param$ and $[100, 6000]$ for $ThetaDistanceParam$.

The WKMeans algorithm for parameter *delta* tested for 0, 0.2, 0.4, 0.6, 0.8, and 1 and we provided the number of segments equals the number of folds in test times the number of segments in tuning set. We assume each fold on average has the same number of segments. This is the only way to prepare a fair comparison environment while other algorithms do not know the number of segments in the test set. Tables B.5, B.4, B.3, and B.2, presented in page 139 - 141, show the parameters applied for each fold on the fishing dataset for all WKMeans, SPD, GRASP-UTS, and CB-SMoT.

We report the best results for each algorithm on three datasets using four metrics: harmonic mean, number of segments, memory and CPU time, and v-measure. We present the results of the harmonic mean of purity and coverage, purity, and coverage in Section 5.4.5. In Section 5.4.6, we report the results of the number of discovered segments. We discuss the performance of algorithms in terms of memory and CPU time in Section 5.4.7. Finally, we provide results of experiments for v-measure on all three datasets in Section 5.4.8.

5.4.5 Purity, Coverage, and Harmonic Mean

In this section, we report the performance of trajectory segmentation algorithms using a measure called *harmonic mean*, which is the harmonic mean of purity and coverage, detailed in section 4.2. We report two components of purity and coverage for debugging purposes and finding instances of over-segmentation and under-segmentation. The results of a Wilcoxon test on the fishing dataset, the Atlantic hurricanes dataset and the Geolife dataset are presented in Tables 5.4, 5.5, 5.6. In the following we review these results.

The results of Wilcoxon tests to compare *harmonic mean* of purity and coverage for trajectory segmentation algorithms on the fishing dataset is presented in Table 5.4. The median harmonic mean for the SWS algorithm on the fishing dataset was

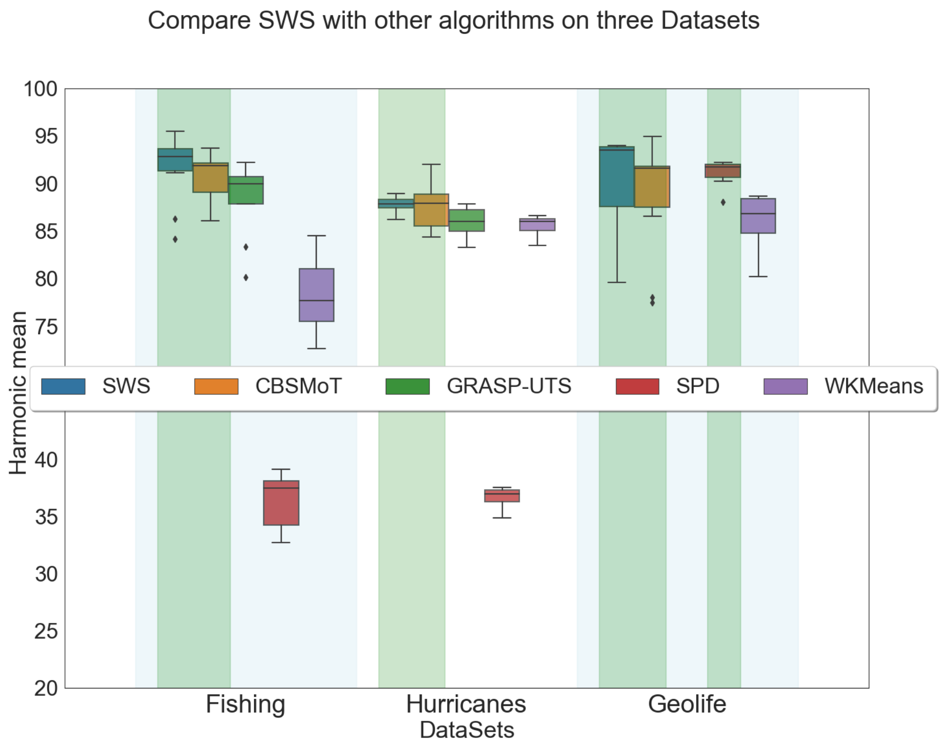


Figure 5.12: Comparing the performance of SWS, CBSMoT, GRASP-UTS, SPD, and WKMeans on three datasets. The green background shows the best algorithms that there is no statistically significance difference between them.

92.86 % (IQR = 2.37), whereas the median for the SPD algorithm was 37.50 % (IQR = 3.83). The Wilcoxon test showed that the difference between SWS and GRASP-UTS was significant ($p = 0.019$, $statistic = -2.34$). Furthermore, SWS gained significant higher harmonic mean in comparison to WKMeans ($p = 0.000$, $statistic = -3.70$). There was no significant difference between CB-SMoT and SWS according to Wilcoxon test results ($p = 0.226$, $statistic = -1.20$); however, the results of SWS had lower variations ($IQR = 2.37$) in comparison to CB-SMoT ($IQR = 3.10$).

The results of Wilcoxon tests to compare harmonic mean of purity and coverage for trajectory segmentation algorithms on the Atlantic hurricanes dataset are presented in Table 5.5. The median harmonic mean for SWS algorithm on the Atlantic hurricanes dataset was 87.92% (IQR = 0.91), whereas the SPD algorithm gained the lowest median for harmonic mean, 37.00% (IQR = 1.06). The Wilcoxon test showed that the difference between SWS and GRASP-UTS was significant ($p = 0.003$,

Trajectory Segmentation Algorithms	Median	CBSMoT	GRASP-UTS	SPD	SWS	WKMeans
CBSMoT	91.90	1.000	0.173	**0.000**	0.226	**0.000**
GRASP-UTS	89.96	0.173	1.000	**0.000**	**0.019**	**0.000**
SPD	37.50	**0.000**	**0.000**	1.000	**0.000**	**0.000**
SWS	**92.86**	0.226	**0.019**	**0.000**	1.000	**0.000**
WKMeans	77.74	**0.000**	**0.000**	**0.000**	**0.000**	1.000

Table 5.4: The Wilcoxson test's p-value results to compare the harmonic mean of different segmentation algorithms on the fishing dataset. The p-values with significant differences are highlighted in bold.

Trajectory Segmentation Algorithms	Median	CBSMoT	GRASP-UTS	SPD	SWS	WKMeans
CBSMoT	**87.92**	1.000	**0.049**	**0.000**	1.000	0.058
GRASP-UTS	86.01	**0.049**	1.000	**0.000**	**0.003**	0.449
SPD	37.00	**0.000**	**0.000**	1.000	**0.000**	**0.000**
SWS	**87.90**	1.000	**0.003**	**0.000**	1.000	**0.000**
WKMeans	86.02	0.058	0.449	**0.000**	**0.000**	1.000

Table 5.5: The Wilcoxson test's p-value results to compare the harmonic mean of different segmentation algorithms on the Atlantic hurricanes dataset. The p-values with significant differences are highlighted in bold.

Trajectory Segmentation Algorithms	Median	CBSMoT	SPD	SWS	WKMeans
CBSMoT	**91.58**	1.000	0.596	0.289	0.058
SPD	**91.74**	0.596	1.000	0.405	**0.000**
SWS	**93.55**	0.289	0.405	1.000	**0.041**
WKMeans	86.84	0.058	**0.000**	**0.041**	1.000

Table 5.6: The Wilcoxson test's p-value results to compare the harmonic mean of different segmentation algorithms on the Geolife dataset. The p-values with significant differences are highlighted in bold.

$statistic = -2.94$). Moreover, SWS gained significant higher harmonic mean in comparison to WKMeans ($p = 0.000$, $statistic = -3.55$). There was no significant difference between CB-SMoT and SWS according to Wilcoxon test results ($p = 1.000$, $statistic = 0.00$); however, the results of SWS had lower variations ($IQR = 0.91$) in comparison to CB-SMoT ($IQR = 3.34$).

The results of Wilcoxon tests to compare harmonic mean of purity and coverage for trajectory segmentation algorithms on the Geolife dataset is shown in Table 5.6. The median harmonic mean for SWS algorithm on the Geolife dataset was 93.55% ($IQR = 6.29$), whereas the median for SPD algorithm boosted to 91.74% ($IQR = 1.34$). GRASP-UTS did not generate segmentation results after working for three days, while other algorithms provided their findings in the range of seconds or minutes. The Wilcoxon test showed that the difference between SWS and WKMeans was significant ($p = 0.041$, $statistic = -2.04$). There was no significant difference between CB-SMoT, SPD and SWS, according to Wilcoxon test results.

The results of these algorithms for all three experimented datasets is presented in Figure 5.12. While SPD gained a reasonably good performance on the Geolife dataset, there was no statistically significant difference between SPD, SWS and CB-SMoT on the Geolife dataset; SPD did not perform well on the fishing dataset and Atlantic hurricanes dataset. We present the results for purity and coverage to discuss the reason for its failure. Considering that WKMeans had a privilege to be aware of an estimated number of segments, it did not produce outstanding results for harmonic mean. The results of GRASP-UTS for the fishing dataset and Atlantic hurricanes datasets were statistically significantly lower than SWS and CB-SMoT for all experimented datasets. The experiment results of a Wilcoxon between SWS and CB-SMoT indicated that there was no statistically significant difference between the performance of them on all datasets. We show the results of algorithms with no statistical difference using a green background in Figure 5.12.

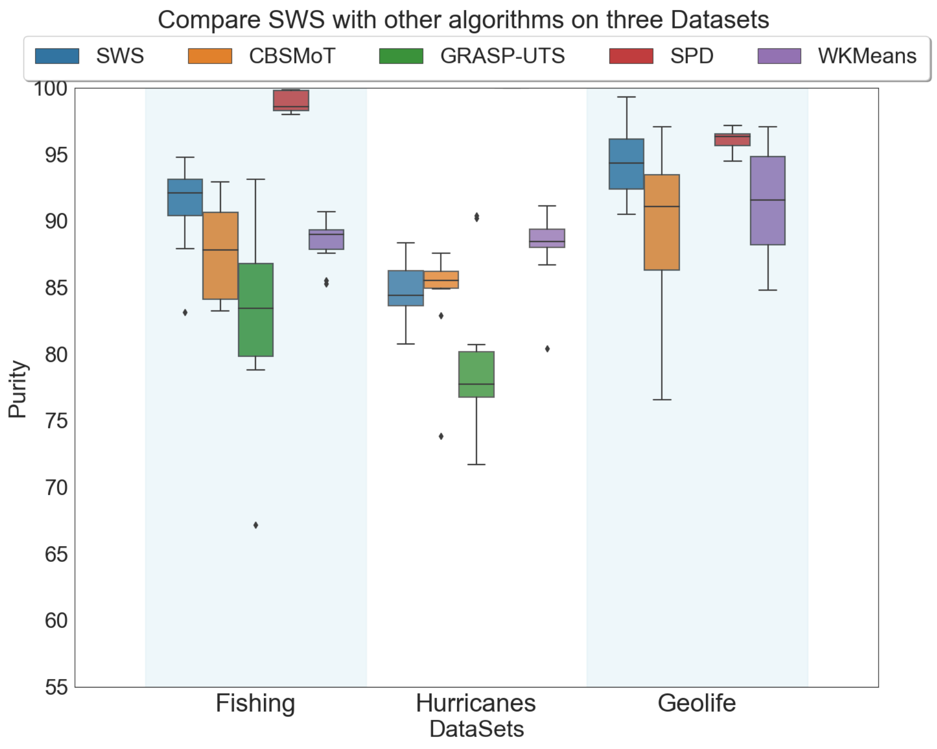


Figure 5.13: Comparing the performance of SWS, CBSMoT, GRASP-UTS, SPD, and W-KMeans on three datasets in terms of purity.

We provide the results of comparing purity and coverage for all experimented algorithms on all three datasets in Figure 5.13 and 5.14. The results of SPD for purity indicated that this algorithm suffered from under-segmentation on the fishing dataset and Atlantic hurricanes dataset, even though we perform an exhaustive search for adjusting its best parameters.

We say an algorithm performs **Under-Segmentation** when it is producing a considerably lesser number of segments than expected. The worst-case for under-segmentation is when the algorithm does not discover any segment; in that, we have a coverage of 100%. GRASP-UTS on hurricanes dataset and WSII (detailed in Chapter 6) on fishing dataset are examples of under-segmentation. We say an algorithm over-segments when algorithm produces considerably too many short segments. The worst-case for over-segmentation is when the algorithm generates one segment for each trajectory point in which the purity is 100% which we observed in the case of SPD on fishing and hurricanes dataset.

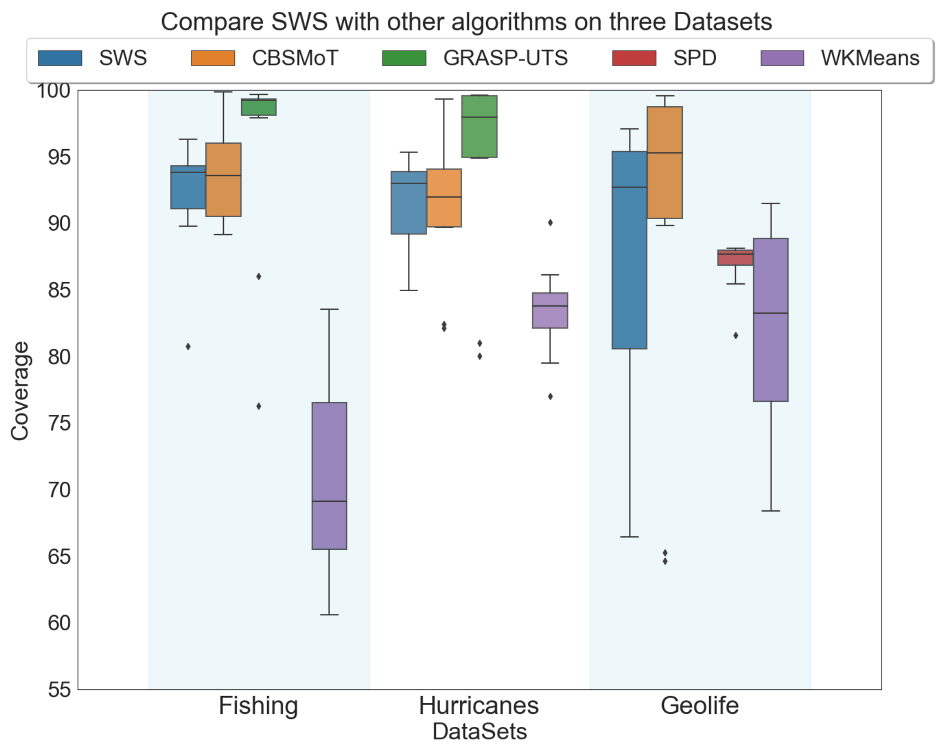


Figure 5.14: Comparing the performance of SWS, CBSMoT, GRASP-UTS, SPD, and W-KMeans on three datasets.

To provide a more precise understanding of over-segmentation and under-segmentation, we experimented with measuring these two parameters in an over-segmented case and an under-segmented example for all three datasets. When we provided the worst possible under-segmented results for the fishing dataset on each fold, the average purity was 21.2%, and the average coverage was 100%. The same experiment on the Atlantic hurricanes dataset produced the average purity of 17.8%, and the average coverage was 100%. For the Geolife dataset, the average purity under these circumstances was 23.2%, and the average coverage was 100%.

When we provide the worst possible over-segmented results for the fishing dataset, the average purity was 100%, and the average coverage was 9.2%. For the Atlantic hurricanes dataset, the average purity was 100%, and the average coverage was 21.7%. And for the Geolife dataset, the average purity was 100%, and the average coverage was 4.9%. This experiment also shows that purity and coverage are more reliable when we have a data set with a higher frequency of capturing new data such as the Geolife dataset.

5.4.6 Number of Segments

One of the metrics that frequently used to evaluate trajectory segmentation algorithms is the number of segments. To find the number of discovered segments, we post-process our data to find the most extended segment generated for each segment identified in our ground truth. The number of discovered segments and the *Base*, which is the number of segments in our ground truth (shown in blue), is compared in Figure 5.15.

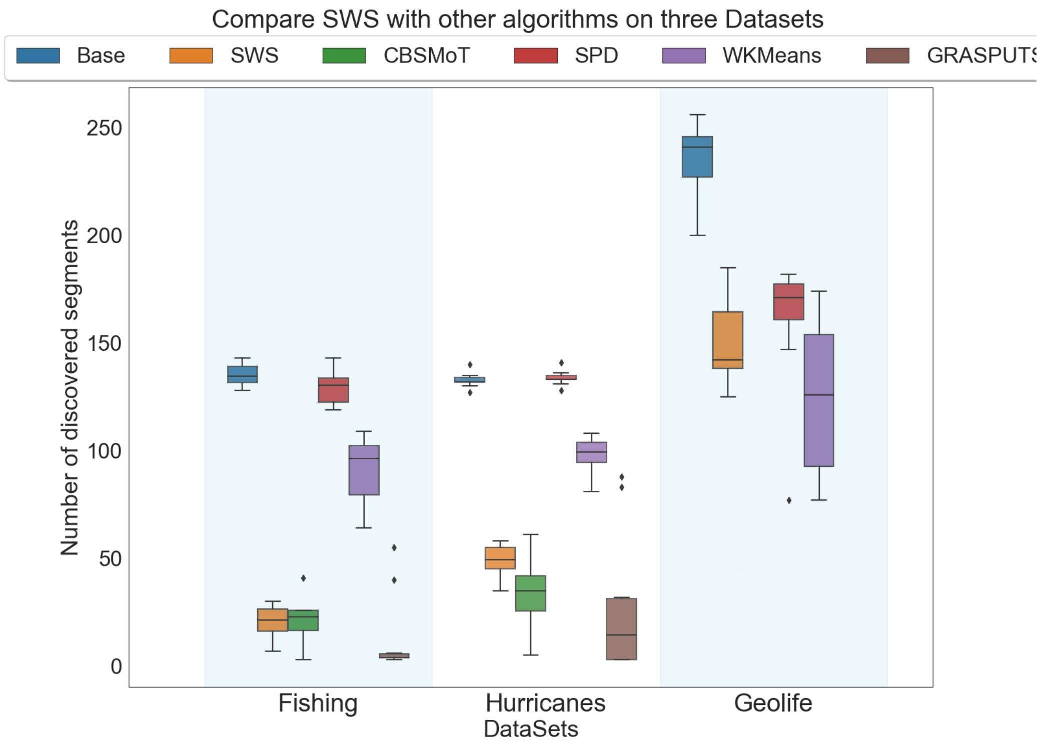


Figure 5.15: Comparing the number of discovered segments of SWS, CBSMoT, GRASP-UTS, SPD, and WKMeans on three datasets. SWS discovered more segments than other algorithms and over-segmentation in SPD discovered more low quality segments on the fishing and Atlantic hurricanes datasets.

The average number of segments on the fishing dataset, *Base*, is 135 segments. SPD discovered 129 segments on average, which is the result of over-segmentation, and most of the discovered segments contain a few trajectory points. Among other methods, GRASP-UTS found 12 segments on average, which was lower than CB-SMoT (20), WKMeans (91), and SWS (20) on the fishing dataset. The number of discovered segments of WKMeans is not a surprise, near to the *Base*, because an estimate of the parameter k, number of segments, is provided to this trajectory segmentation algorithm.

The ground truth on the Atlantic hurricanes dataset indicates that there are 132 segments on average for this dataset, *Base*. SPD discovered in 133 segments on average as a result of over-segmentation. SWS found the highest number of segments (49) on average in comparison to GRASP-UTS (27), CB-SMoT (34), WKMeans (98) on the Atlantic hurricanes dataset.

In the Geolife dataset, the number of segments in ground truth was 235 on average. SPD was able to discover 160 segments on average this time without over-segmentation. After that, SWS discovered the second-highest number of segments, 149, in comparison to WKMeans (125), CB-SMoT (65).

The results of this experiment show that SWS was a successful algorithm on all three datasets, while algorithms such as SPD was just gained success in a specific domain. Comparing SWS and CB-SMoT shows that SWS able to discover more segments while they both at the same level in terms of harmonic mean, discussed in Section 5.4.5.

5.4.7 Memory and CPU time

In this section, we compare trajectory segmentation algorithms from the perspective of memory consumption and processing demands. We measure the amount of memory used by each algorithm and the CPU time each algorithm used in milestones such as the start of execution, after loading data, after each fold tuning, after each fold testing, and end of the experiment. We used the same code template to execute all tests.

The amount of memory used by each algorithm during the execution is presented in Figure 5.16. The GRASP-UTS graph shows many ups and downs that are the result of swap memory and consumed way more than other algorithms memory and CPU time. As we can see, SWS with a linear interpolation kernel (SWS_L) was

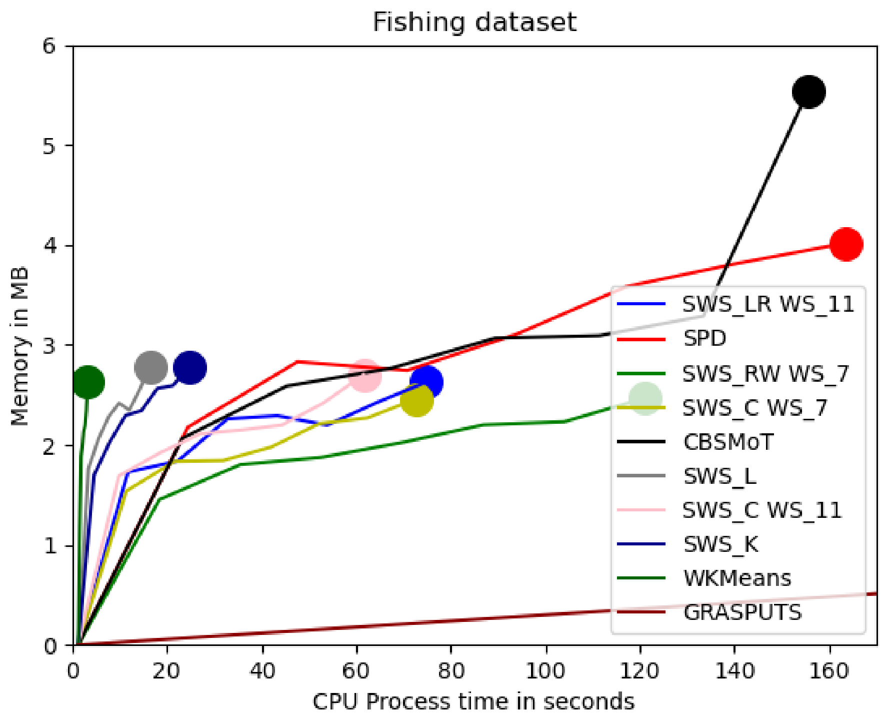


Figure 5.16: Comparing the performance of SWS, CBSMoT, GRASP-UTS, SPD, and W-KMeans on fishing datasets. GRASP-UTS produced segments after 16 hours and used near 60MB memory. WKMeans generated segments faster than all algorithms. SWS with linear kernel (SWS_L) gained the second rank after WKMeans.

the fastest algorithm after WKMeans. The GRASP-UTS was the slowest trajectory segmentation algorithm among experimented algorithms. Comparing CB-SMoT and SWS with a linear kernel (SWS_L), the SWS algorithm was executed using almost half of the amount of memory and much less of the CPU time. Because we provide just geo-location to WKMeans, and the algorithm was aware of the number of segments, the complexity of the time dimension and discovering the number of segments was removed from this algorithm. This made WKMeans executes faster than all other experimented algorithm.

5.4.8 V-measure

The segmentation task has some similarities to clustering tasks. From one perspective, we are making clusters of trajectory points and call them segments. However, we have a limitation that two consecutive segments could not belong to the same cluster and elements of same cluster can appear in more than one segment. We decided to compare trajectory segmentation algorithms from the perspective of clustering. One of the frequently used metrics for clustering is V-measure, which is the harmonic mean of homogeneity and completeness. V-measure is one of the metrics frequently used for evaluating the quality of a clustering tasks [63]. We measure this metric for all of our experiments to compare segmentation algorithms from this view point. SWS was able to obtain the best V-measure among all algorithms on the Geolife dataset and the Atlantic hurricanes dataset, shown in Figure 5.17. However, it did not gain the highest V-measure for the fishing dataset, but it gained a competitive V-measure.

A Wilcoxon test indicated a significant difference between SWS and CB-SMoT on the fishing dataset (p=0.049, s=−1.96). A Wilcoxon test showed that SWS results of v-measure on the Atlantic hurricanes dataset are significantly higher than all other experimented algorithms ($mdn = 92.39$, $IQR = 9.60$). GRASP-UTS generated unfortunate results with high variations for v-measure on the fishing dataset and the Atlantic hurricanes dataset. CB-SMoT performed poorly on the Geolife dataset in terms of v-measure. SPD still did not gain a good result of V-measure while it achieved the best results similar to SWS on the Geolife dataset ($p_{value} = 0.096$, $statistic = 1.66$). SPD was designed for the Geolife dataset by researchers in the mobility domain, whereas SWS is more generic and achieves competitive results in all datasets.

5.5 Discussion

We compared our proposed method, SWS, against four well-known approaches in trajectory segmentation, including CB-SMoT, GRASP-UTS, SPD, and WKMeans on three datasets from three different domains (the fishing dataset, the Atlantic hurricanes dataset, and the Geolife dataset). We compared these trajectory segmentation algorithms from four perspectives: 1) *harmonic mean* of purity and coverage, 2) number of discovered segments, 3) memory consumption and computing desire and 4) V-measure. Comparing from the *harmonic mean* perspective showed that SWS and CB-SMoT gained the highest performance. The results of comparing algorithms concerning the number of discovered segments showed that SWS found more segments

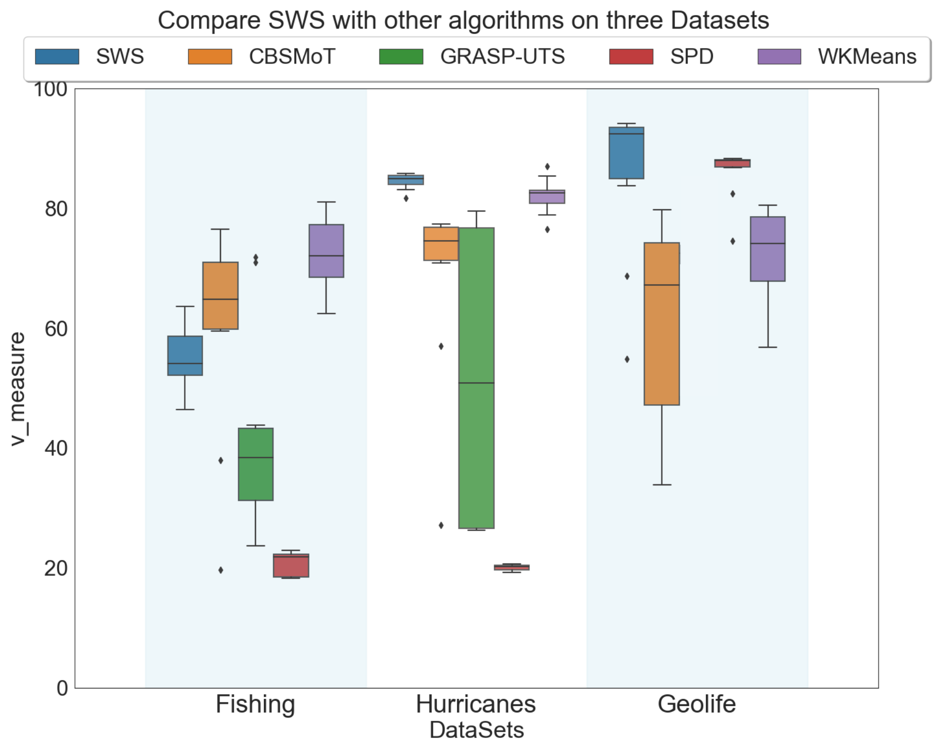


Figure 5.17: Comparing the performance of SWS, CBSMoT, GRASP-UTS, SPD, and WKMeans on three datasets in terms of V-Measure.

on average compared to that of the CB-SMoT. The memory and computing requirements for SWS remained low, and GRASP-UTS consumed the highest amount of memory and CPU time to generate its results. From the clustering perspective, CB-SMoT execution on the fishing dataset gained second high v-measure after WKMeans and SWS went into the third rank; however, SWS gained the most elevated status among all algorithms on the Atlantic hurricanes dataset and the Geolife dataset in terms of v-measure.

Debugging the segmentation results for SWS revealed some of the weaknesses of this algorithm that we discuss here. Figure 5.18 shows one of the frequent errors that SWS made during the segmentation task. This graph applied SWS algorithm with a random walk with a window size of seven, which is tuned on fold one of the fishing dataset. The figure on the right side shows that this trajectory is one segment. Our algorithm discovered three segments in this trajectory, which are indicated by black, yellow, and magenta. Our interpretation is that there might be more than one behaviour change in the course of fishing. SWS aims to discover the behavioural changes in a trajectory, and it seems that it identified these behaviour changes. We register this issue as one of the weaknesses of SWS that resulted in low v-measure.

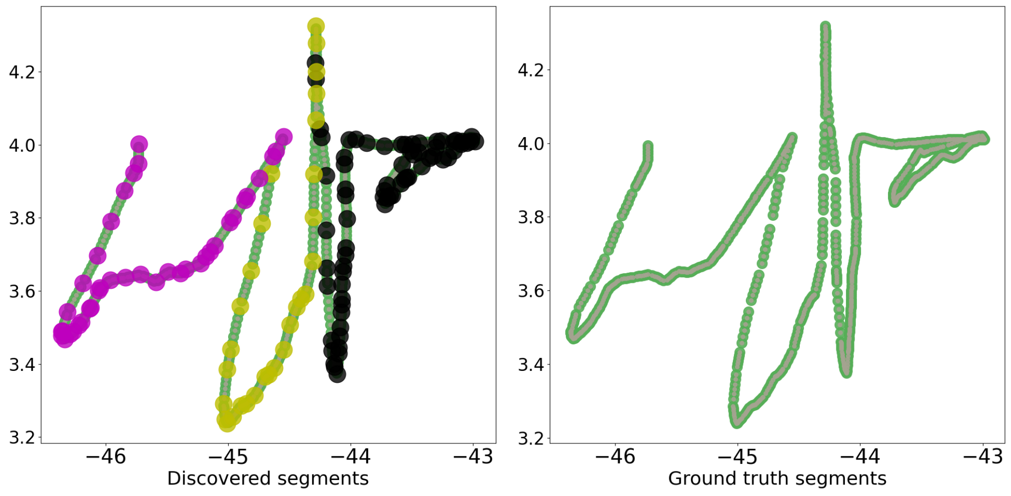


Figure 5.18: Trajectory on the right shows a fishing activity labeled by subject matter expert. The left image shows the output of SWS segmentation that identified three segment. This graph shows a limitation of SWS when a course of movement includes more than one behavior change.

The second weakness of SWS is when the rate of capturing new samples are low in a dataset. The performance of SWS increases when the period of data capturing in average decreases, detailed in Section 5.4.3 and presented in Figure 5.11.

5.6 Summary and Concluding Remarks

In this chapter, we proposed our Sliding Window Segmentation (SWS) algorithm to segment trajectories. This algorithm works based on the idea that any deviation from the normal behaviour of moving objects is a sign of changing moving object's behaviour. The regular routine is modelled using different kernels for different domains. For each trajectory point, we generate error value that shows the deviation of that point from the interpolated point. Then using these values, we create an error signal and process this signal to discover the partitioning positions . Our experiments show that SWS and CB-SMoT, with no statistically significant difference, provide the best performance on all three datasets in terms of *harmonic mean*. SPD provides similar performance only on the Geolife dataset while it was not performing well on the fishing dataset and the Atlantic hurricanes dataset. The performance of WKMeans and GRASP-UTS was statistically significantly lower than other methods. Our experiments showed that there is a linear correlation between the frequency of capturing new trajectory points in a dataset and the performance of SWS. Furthermore, by decreasing the period of capturing new trajectory points, the performance of SWS increases, detailed in Section 5.4.3. Moreover, purity and coverage metrics become more reliable by increasing the frequency of capturing new points in a dataset, detailed in Section 5.4.5.

SWS was able to discover more segments than CB-SMoT on the Geolife dataset. Also, SWS provides the result of segmentation faster than CB-SMoT, GRASP-UTS, and SPD with less memory usage.

We discussed two weaknesses of SWS in Section 5.5. First, SWS may discover shorter segments because of the nature of the movement. For example, in the course of fishing shown in Figure 5.18, SWS generated three segments where the ground truth registered one segment. The second weakness was when SWS is trying to segment a dataset with more extended periods of capturing new points. Lower frequency of capturing new point includes more unrelated trajectory points to the sliding window. Increasing this gap gives the moving object possibility of hiding the behaviour change from the trajectory segmentation sights.

Chapter 6

Wise Sliding Window Segmentation

A limited number of trajectory datasets contain labelled data, annotated segments by a subject matter expert. This kind of data is processed by a subject matter expert and segmented using some tools such as VISTA [68], which facilitates access to some labelled trajectories segmented by a subject matter expert. A supervised trajectory segmentation algorithm is beneficial when we have access to such labelled data. The labelled data helps the trajectory segmentation algorithm to be adjusted to a domain or customized to the subject matter preferences. The objective of a supervised trajectory segmentation algorithm may be to facilitate trajectory segmentation in a trajectory tagging platform, which can save a considerable amount of time for a subject matter expert by providing accurate suggestions on partitioning position. When we have a high rate of capturing, this task becomes more tedious, and deciding on a partitioning position becomes harder. We did not find any supervised method for trajectory segmentation in the literature. Therefore, we explored this approach to propose a supervised trajectory segmentation algorithm. Wise Sliding Window Segmentation (WSII, it reads W-S-Two) is a supervised trajectory segmentation algorithm that inherits the segmentation idea using the deviation from a norm as an indicator to segment data.

In the following section, we explain the details of WSII approach. Then, we report the results of our experiments on three datasets in Section 6.2. After that, we discuss our results in Section 6.3. Finally, we summarize our proposed supervised approach in Section 6.4.

Since we have a single object trajectory in this chapter, we do not use a superscript as an object identifier. The superscript notation here will be used to discriminate between labeled and unlabeled data.

Algorithm 6.1 Wise Sliding Window Segmentation (WSII)

```
Require: $\tau_n^l$ - the labeled raw trajectory
    $\tau_m^u$ - the unlabeled raw trajectory
    $ws$ - Size of sliding window (an odd number)
    $kernel$ - interpolation kernel
    $B, B_{param}$ - Binary classifier and its parameters
    $\mu$ - Majority vote threshold
Ensure: SegmentID
 1: $E^l \leftarrow GenerateErrorSignal(\tau^l, ws, kernel)$
 2: $E^u \leftarrow GenerateErrorSignal(\tau^u, ws, kernel)$
 3: $D_x^l, D_y^l \leftarrow CreateTrainingData(\tau^l.TSID, E^l)$
 4: $B.Train(D_x^l, D_y^l)$
 5: $U_x^l \leftarrow CreateBlindSamples(E^u)$
 6: $U_{y'}^l \leftarrow B.Predict(U_x^l)$
 7: $SegmentID \leftarrow MajorityVote(U_{y'}^l, \mu)$
 8: return $SegmentID$
```

6.1 Wise Sliding Window Segmentation (WSII)

In this algorithm, we have two trajectory data: 1) τ_n^l, which is a trajectory data annotated by a subject matter expert. 2) τ_n^u, which is a trajectory data that we do not have the trajectory segmentation ground truth, and we are willing to segment it. The algorithm inherits the core functions (Generating error signal) from SWS; Therefore, ws and $kernel$ are the window size and the kernel of SWS, as explained in Chapter 5. We keep the window size of the proposed algorithm similar to the window size of the core algorithm in this research. This algorithm benefits from training a binary classifier, B that can be tuned as needed, Algorithm 6.1 line 4. Furthermore, the algorithm uses a majority vote decision-making process in which the percentage of required affirmative votes is defined by μ, Algorithm 6.1 line 7. For example, $\mu = 0.50$ means that the algorithm needs more than fifty percent of the votes to make its decision. Algorithm 6.1 shows the steps of WSII in detail.

An overview of the Wise Sliding Window Segmentation (WSII) method is presented in Figure 6.1, which has five core procedures: 1) Calling SWS to generate Error Signal, 2) Create Training Data, 3) Create blind samples, 4) Binary Classification Model, and 5) Majority Vote. First, the WSII creates the error signal from the labelled trajectory dataset (τ_n^l), and unlabeled dataset (τ_n^u) by calling SWS using

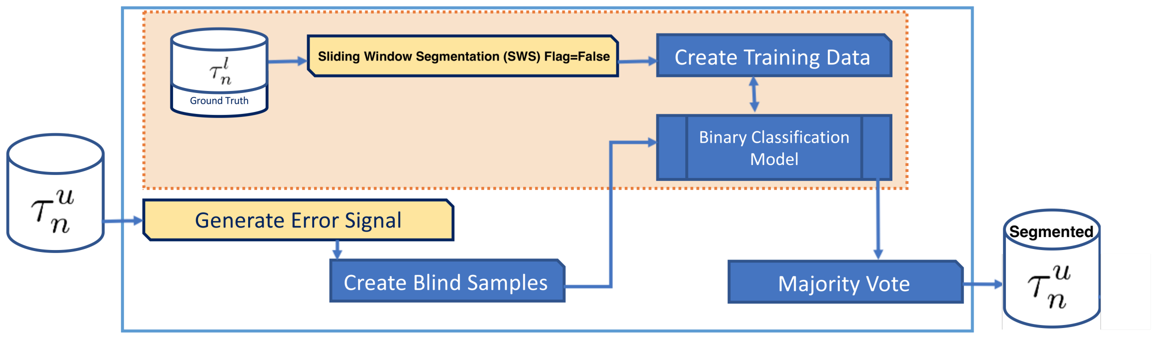


Figure 6.1: An overview of the Wise Sliding Window Segmentation (WSII). WSII benefits from using a binary classifier and a majority vote mechanism to find potential partitioning positions.

parameter $flag = False$ (two yellow solid bordered boxes in Figure 6.1), which is detailed in Section 6.1.1. The second step is to generate the training data using the error signal, by sliding a window over its values and adding the presence or absence of a partitioning position. This part is detailed in Section 6.1.2. The third step is to train a binary classifier to recognize the partitioning positions over the sequence of error signals. This part is detailed in Section 6.1.4. Finally, WSII creates samples from unlabeled trajectories, predicts the presence of partitioning position in each sample, and utilizes a majority vote technique to decide on the location of partitioning position, as detailed in Section 6.1.5.

6.1.1 Generating the Error Signal

The first step of our algorithm is a call to the SWS algorithm using parameter $flag = False$ (this parameter tells the SWS to return the error signal). It is detailed in Chapter 5, which creates a sliding window over a trajectory to compute a single error for trajectory points. The error is generated by calculating the deviation of the interpolated midpoint of the window from the actual midpoint. This process is repeated by moving a sliding window by one point forward, so receiving a new trajectory point, it adds the next point to the window and removes the oldest point from the sliding window. An example of this process is shown in Figure 5.1 in page 62, and the procedure is elaborated in Algorithm 5.2 in page 63.

6.1.2 Creating Training Data

The second core procedure of WSII is to create a training dataset using the sequential error values extracted in the previous step. First, we create an array of size q of error values that will belong to the first training sample (these q error values are the attributes of our classifier). We expect q error values in the middle of a segment generate a monotone pattern; whereas, they may generate non-monotone pattern when they include a partitioning position. Selecting q depends on the sampling rate in the dataset and should include enough trajectory points to capture a behaviour change. We use the ground truth information (i.e., if in this particular region there was a change in the behaviour) to annotate the label of this sample, Algorithm 6.2 line five. If this window includes a partitioning position, it is labelled as 1 and 0 otherwise. This annotation plays the role of target value for our classifier. Therefore, $XTrain$ plays the role of attributes for the binary classifier and $YTrain$ plays the role of target value in Algorithm 6.2.

Algorithm 6.2 Create Training Data

```
Require: $\tau_n^l.TSID$ { The SID of the raw trajectory}
Require: $E^l$ {Error signal for $\tau_n^l$}
Require: $q$ {Number of attributes}
1: $XTrain \leftarrow []$
2: $YTrain \leftarrow []$
3: for $i \leftarrow 0$ to $n - q$ do
4:     $XTrain[i] \leftarrow E^l[i, i + q]$
5:     $YTrain[i] \leftarrow Sign(len(set(TSID[i, i + q])) - 1)$
6: end for
7: return $XTrain, YTrain$
```

By receiving every new trajectory point, we remove one point from the start of our window and add the next to the end of the window. Then we create our next sample by applying the same step of labelling 1 when a partitioning position is present in the sliding window, and 0 if it is not. We repeat this procedure until all the error signals are evaluated.

To understand how the labeling process works, we show an example in Figure 6.2. In this example, the training data are created for the sliding window built with seven (e_1 to e_7) trajectory points over eleven slides (i.e., w_1 to w_{11}). As can be seen in

Figure 6.2, from w_1 to w_3, there was no big change in the error signal (ranging from 120 to 340 meters). In w_4, the value of 560 characterizes a high jump in the estimated error. It reflects a real change in the moving object's behaviour, resulting in a positive example (i.e., there is a partitioning position) in the training data. Examples from w_4 to w_{10} are labelled as positive due to the presence of a partitioning position in the sliding window. From w_{11}, the samples are again labelled as negative examples due to the absence of a partitioning position.

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	**label**
...	...	...	...	...	...	...	...	...
w_1	120	160	150	140	180	130	200	0
w_2	160	150	140	180	130	200	210	0
w_3	150	140	180	130	200	210	340	0
w_4	140	180	130	200	210	340	**560**	1
w_5	180	130	200	210	340	**560**	320	1
w_6	130	200	210	340	**560**	320	210	1
$w_{...}$	...	...	...	...	...	...	...	...
w_{10}	**560**	320	210	120	320	273	200	1
w_{11}	320	210	120	320	273	200	130	0

Figure 6.2: Example of a training set generated by WSII.

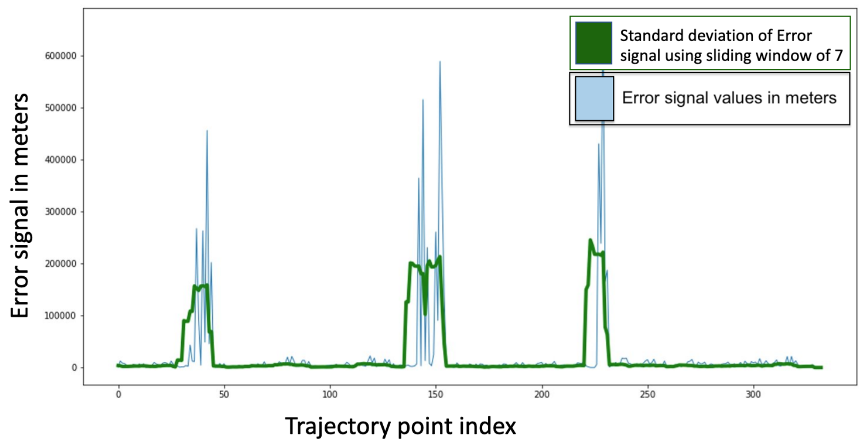


Figure 6.3: The blue line shows the error signal for a sample trajectory with three segments. Using a sliding window of 7, we calculate the standard deviation of the error signal of the sliding window and we plot them in green. This graph shows that maximum fluctuations in error value happen when we calculate error signal for the boundaries of a segment.

6.1.3 Create Blind Samples

Algorithm 6.3 shows how we generate sample data for the unlabeled data, τ_m^u. This procedure creates samples for the binary classifier to predict the presence of a partitioning position. Note that we select the same number of attribute, q, as we used for our training step. Each sample passes to the binary classifier, and a prediction assigns to the sample that indicates the partitioning position is present in the sample or not.

Algorithm 6.3 Create Blind Samples

```
Require: $E^u$ { Error signal for $\tau_m^u$}
Require: $q$ {Number of attributes}
Ensure: $XTrain$
1: $XTrain \leftarrow []$
2: for $i \leftarrow 0$ to $m - q$ do
3:     $XTrain[i] \leftarrow E^u[i, i + q]$
4: end for
5: return $XTrain$
```

6.1.4 Binary Classification Model

A binary classifier is used by WSII to categorize each error signal sample into either a partitioning position or not. The labelled trajectory data, created in the previous step, is used to generate training samples ($XTrain$, attributes of the classification) for this binary classifier so that it can classify signal samples into a class ($YTrain$, target variable) where a sliding window has a partitioning position (e.g., value 1) or a class when it does not have a partitioning position (e.g., value 0).

A sample error signal is presented in Figure 6.3 that it has the minimum fluctuations far from a partitioning position, and the maximum variations while transitioning from one segment to a new one.

Therefore, detecting the area that includes partitioning positions is an indicator that the behaviour has changed. We apply the binary classifier to identify these areas over a trajectory with the highest likelihood of containing partitioning positions. In this work, we used a random forest classifier [9] as a base binary classifier to benefit from its bagging power while processing extended window sizes faster by limiting the number of features. However, we explored some other classification models that can

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	b_{cls}	m_{cls}
$w_{...}$	...	...	...	...	...	...	...	...	...
w_m	100	150	230	160	170	320	400	0	...
w_{m+1}	150	230	160	170	320	400	390	1	...
w_{m+2}	230	160	170	320	400	390	320	0	...
w_{m+3}	160	170	320	400	390	320	380	0	0
w_{m+4}	170	320	400	390	320	380	330	0	1
w_{m+5}	320	400	390	320	380	330	490	1	0
w_{m+6}	400	390	320	380	330	490	450	1	0
w_{m+7}	390	320	380	330	490	450	230	1	...
w_{m+8}	320	380	330	490	450	230	160	0	...
w_{m+9}	380	330	490	450	230	160	190	0	...
$w_{...}$	...	...	...	...	...	...	...	...	...

Figure 6.4: Example of the majority vote mechanism.

be used in this step. After forecasting these transitioning areas, we use a majority vote mechanism to decide precisely where to place a partitioning position, explained in the next section.

6.1.5 Majority Vote

At this step, we use the same sliding window of size q to decide if a partitioning position occurred. The steps of this procedure are explained in Algorithm 6.4. Since we are using a sliding window, each trajectory point can be part of q sliding windows, and we classify each window using the binary classifier. This means that we have q predicted values by the binary classifier, each of them is generated using one of the sliding windows that contains the trajectory point, Algorithm 6.4 line 5. Using a majority vote mechanism for these q outputs leads us to the final decision: the trajectory point is a partitioning position if more than $\mu\%$ of the sampled signals are labelled as a partitioning position.

Leveraging this feature and applying the voting technique, we can have a more robust evaluation to support if a point is a partitioning position or not. q results support the decision on identification of a trajectory point as a partitioning position, each of which contributes $1/q$ to the final decision. This means a misclassification of the binary classifier weights $1/q$. Although increasing q can make the algorithm more robust to noise, it will make it fail to identify segments with a length smaller than q and only possible when we have a good sampling rate. Furthermore, the algorithm is robust against noisy points for longer trajectories, which may happen in trajectory

Algorithm 6.4 Majority Vote

```
Require: $U_y^l$
Require: $\mu, q$
Ensure: SegmentID
1: $mid \leftarrow int(q/2)$
2: $start \leftarrow 0, ng \leftarrow -1, pg \leftarrow 1, SegmentID \leftarrow []$
3: $l \leftarrow len(U_y^l)$
4: for $i \leftarrow mid + 1$ to $l - mid$ do
5:    if $\sum_{n \leftarrow i-mid}^{i+mid} U_y^l[n] > [ws * \mu]$ then
6:       if $i - start \leq q$ then
7:          $SegmentID \leftarrow SegmentID + [-ng] * (i - start)$ {$A * n$, where $A$ is an array and $n$ is an integer, and it concatenates $n$ copies of array $A$. For example, $[0] * 3 = [0, 0, 0]$}
8:          $ng \leftarrow ng - 1$
9:          $start \leftarrow i$
10:      else
11:         $SegmentID \leftarrow SegmentID + [-pg] * (i - start + 1)$
12:         $pg \leftarrow pg + 1$
13:         $start \leftarrow i + 1$
14:      end if
15:   end if
16: end for
17: $SegmentID \leftarrow SegmentID + [pg] * (l - start)$
18: return $SegmentID$
```

data due to collection device errors.

The inability to discover segments shorter than the sliding window is one of the WSII weaknesses.

An example of the majority vote mechanism's advantages is shown in Figure 6.4, where a window with $q = 7$ was used. In Figure 6.4, the column b_{cls} was forecasted by the binary classifier for w_m to w_{m+9}. It is possible to see in Figure 6.4 that w_{m+3} is decided by evaluating the b_{cls} column values from w_m to w_{m+6} (0,1,0,0,0,1,1). The decision regarding a majority vote for w_{m+3} is equal to 0 since $|\#0| = 4$ and $|\#1| = 3$. For deciding the final value of w_{m+4} the lines from w_{m+1} to w_{m+7} are used. The evaluation of the set (1,0,0,0,1,1,1) through a majority vote ($|\#0| = 3$ and $|\#1| = 4$) results in the decision of 1 (i.e., a partitioning position occurred). As previously stated, such a strategy makes WSII robust against spatial jumps due to GPS error in the data collection process.

6.2 Experimental Evaluation

In this section, we evaluate our proposed supervised method and compare it to state-of-the-art approaches. We focus on three experiments: 1) majority vote selection, 2) binary classifier selection, 3) comparison against other trajectory segmentation approaches. In Section 6.2.1, we evaluate the majority vote parameter tuning. Then we present the experimental results on the selection of the binary classifier in Section 6.2.2. After that, we compare the performance of WSII with other trajectory segmentation algorithms in Section 6.2.3.

6.2.1 Majority Vote Experiment

The majority vote's μ value is one of the parameters of the WSII algorithm. This parameter identifies the number of affirmative votes for a trajectory point so that the algorithm can call it partitioning position. In this experiment, we applied random forest as the binary classifier and measured the effect of the majority vote parameter, μ, on three datasets: the fishing dataset, the Atlantic hurricanes dataset, and the Geolife dataset. In this experiment, we avoid cleaning the dataset from segments shorter than the window size of the WSII to capture and highlight any weakness of WSII.

In Chapter 4, we reviewed some features of our datasets. One of these features was the number of short segments in each dataset, discussed in Section 4.1.

Tables 4.1, 4.2, and 4.3 show that $283/1990 \approx 14.22\%$ of the Atlantic hurricanes dataset are segments shorter than window size seven, $158/5190 \approx 3.04\%$ of the fishing dataset are segments shorter than sliding window size seven, and $44/32046 \approx 0.13\%$ of the Geolife dataset is shorter than window size seven. When we increased the majority vote parameter, μ, from 0.6 to 0.9, the average *harmonic mean* of purity and coverage increased from 67.31% to 68.85% on the fishing dataset. On the Atlantic hurricanes dataset, the average *harmonic mean* increased from 71.37% to 82.94% by increasing the majority vote parameter, μ, from 0.6 to 0.9. The average *harmonic mean* of purity and coverage experienced a raise from 91.76% to 92.41% by increasing the majority vote parameter, μ, from 0.6 to 0.9 on the Geolife dataset.

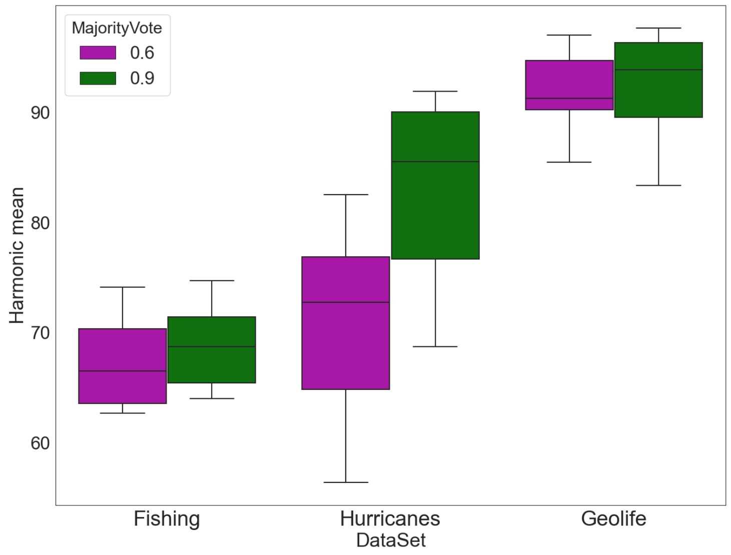


Figure 6.5: Segmentation using WSII with different Majority Vote parameters using random forest as binary classifier. The results show that the increase in majority vote value increases the harmonic mean; however, this increase is not significant for all datasets.

Despite the improvements in harmonic mean on all datasets, a Wilcoxon test indicated that just the gain on the Atlantic hurricanes dataset is statistically significant ($p = 0.019$, $s = 2.34$). The improvements on the Geolife dataset and the fishing dataset were not statistically significant. The results of this experiment are presented in Figure 6.5. The results show that by increasing μ, the WSII would not significantly improve in terms of harmonic mean on the Geolife dataset and the fishing dataset. Therefore, we suggest using $\mu = 0.6$ as a default value for this parameter. Depending the attributes of a dataset (amount of noise or sampling rate), we might find adjustment of this parameter beneficial.

6.2.2 Binary Classifier Experiment

The binary classifier is one of the principal components of WSII. There are several choices for a binary classifier such as Decision Tree (DT), Random Forest (RF), Neural Networks (NN)[1], and Naive Bayes (NB). To evaluate the performance of WSII using

[1]Feed-forward Neural Networks with one hidden layer with seven nodes and *Relu* activation functions, an Adam optimizer and $alpha = 0.001$

these binary classifiers, we apply these four binary classifiers, which are representatives of four categories of classifiers. Since we are using a majority vote decision-making mechanism after applying the binary classifier, a weak classifier still can produce satisfactory results. Considering the complexity of some classifiers such as neural networks, we prefer to use a simple classifier such as decision tree or Naive Bayes as a default parameter for our algorithm.

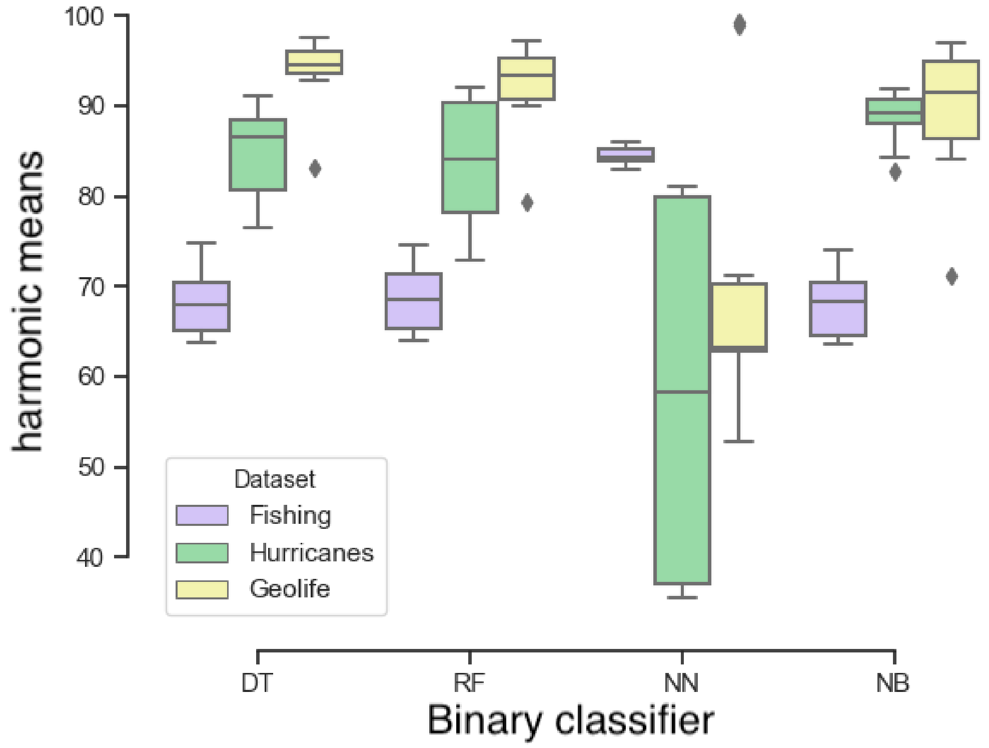


Figure 6.6: WSII experiment for different binary classifiers.

Similar to the previous experiment, we decided to skip the cleaning step for WSII. Therefore, we did not remove the segments shorter than the sliding window of size seven.

The results of Wilcoxon tests on three datasets using four classifiers with a random walk kernel is presented in Table 6.1. The median *harmonic mean* in WSII with neural networks as the core binary classifier was 84.22% ($IQR = 1.37$), the highest on the fishing dataset. On the Atlantic hurricanes dataset, the Naive Bayes binary classifier gained the most top *harmonic mean* of 89.30% ($IQR = 2.80$), and the highest

Datasets		Median	DT	RF	NN	NB
Fishing Dataset	DT	68.2	1.000	0.596	**0.000**	0.820
	RF	68.6	0.596	1.000	**0.000**	0.596
	NN	**84.2**	**0.000**	**0.000**	1.000	**0.000**
	NB	68.2	0.820	0.596	**0.000**	1.000
Atlantic Hurricanes Dataset	DT	86.6	1.000	0.879	**0.005**	0.082
	RF	84.1	0.879	1.000	**0.012**	0.289
	NN	58.3	**0.005**	**0.012**	1.000	**0.000**
	NB	**89.3**	0.084	0.289	**0.000**	1.000
Geolife Dataset	**DT**	**94.5**	1.000	0.325	**0.023**	0.226
	RF	93.5	0.325	1.000	**0.023**	0.496
	NN	63.2	**0.023**	**0.023**	1.000	**0.025**
	NB	91.6	0.226	0.496	**0.025**	1.000

DT: Decision Tree
RF: Random Forest
NN: Neural Network
NB: Naive Bayes

Table 6.1: The Wilcoxson test's p-value results to compare the harmonic mean of WSII with different binary classifiers on the three datasets. The p-values with significant differences are highlighted in bold.

harmonic mean on the Geolife dataset was 94.53% ($IQR = 2.48$). The Wilcoxon test indicated that there is a significant difference in using neural networks on the fishing dataset compared to all other classifiers. Moreover, it suggested that the use of Naive Bayes as a core binary classifier is not significantly different than using the random forest.

Considering the performance of Naive Bayes comparing to random forest, the use of Naive Bayes as a core binary classifier may bring considerable performance improvement.

Furthermore, the Wilcoxon test showed that the difference between using a Decision tree and Random Forest on the Geolife dataset was not significant ($p = 0.325$, $statistic = -0.98$).

On the fishing dataset, the neural network as the binary classifier of WSII gained the highest *harmonic mean* (84.22%) in comparison to Decision Tree (68.27%), Random Forest (68.62%), and Naive Bayes (68.27%). On the Atlantic hurricanes dataset, Naive Bayes classifier produced the highest *harmonic mean* (89.30%) in comparison to Decision Tree (86.66%), Random Forest (84.19%), and Neural Networks (58.32%).

The best *harmonic mean* is produced by Decision Tree (94.53) on the Geolife dataset in comparison to Naive Bayes (91.61%), Random Forest (93.50%), and Neural Networks (63.25%). Figure 6.6 shows the results of this experiment.

The results of this experiment suggest that Decision Tree and Random Forest classifiers perform better than other classifiers on the Geolife dataset. For Atlantic hurricane dataset, the Naive Bayes gained the highest performance among all binary classifiers. For the fishing dataset, Neural Networks provided the highest *harmonic mean* among all experimented binary classifiers.

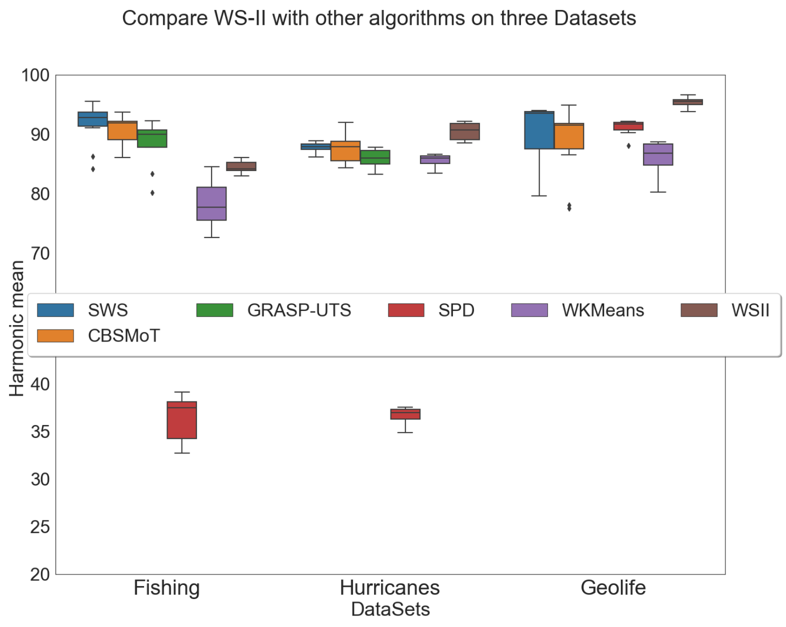


Figure 6.7: Comparing the results of *harmonic mean* of WSII against five other trajectory segmentation algorithms on three different datasets. WSII in brown outperformed SWS in blue on the Atlantic hurricanes dataset and the Geolife dataset.

6.2.3 Comparison With Other Segmentation Approaches

We compared WSII results on *harmonic mean* with SWS [23], GRASP-UTS [69], SPD [85], CB-SMoT [58] and WKMeans [44] on the fishing dataset, the Atlantic hurricanes dataset, and the Geolife dataset. The GRASP-UTS was excluded from experiments on the Geolife dataset because it did not generate the segments after waiting for three days, while other algorithms produced their results in the range of a few seconds or minutes. In these experiments, we decided to skip cleaning the short segments from

Trajectory Segmentation Algorithms	Median	CBSMoT	GRASP UTS	SPD	SWS	WKMeans	WSII
CBSMoT	91.90	1.000	0.173	**0.000**	0.226	**0.000**	**0.000**
GRASP-UTS	89.96	0.173	1.000	**0.000**	**0.019**	**0.000**	**0.019**
SPD	37.50	**0.000**	**0.000**	1.000	**0.000**	**0.000**	**0.000**
SWS	**92.86**	0.226	**0.019**	**0.000**	1.000	**0.000**	**0.000**
WKMeans	77.74	**0.000**	**0.000**	**0.000**	**0.000**	1.000	**0.000**
WSII	84.22	**0.000**	**0.019**	**0.000**	**0.000**	**0.000**	1.000

Table 6.2: The p-value results of the Wilcoxson test to compare the harmonic mean of WSII with different binary classifiers on the fishing dataset. The p-values with significant differences are highlighted in bold.

the datasets to highlight one of the weaknesses of WSII.

Learning from previous experiments in Sections 5.4.2 and 6.2.2, we set up this experiment with the following parameters to achieve better results in each domain. Note that the experiments followed the experimental protocol (see Section 5.4.1). First, we choose a Neural Network's core binary classifier and linear interpolation kernel for the fishing dataset. On the Atlantic hurricanes dataset, we use a Naive Bayes classifier with a linear interpolation kernel. For the Geolife dataset, we applied a random forest binary classifier with a kinematic interpolation kernel.

The results of Wilcoxon tests on the fishing dataset for six trajectory segmentation algorithm is presented in Table 6.2. Contrary to our expectation, WSII did not perform well on this dataset and was not able to outperform CB-SMoT, GRASP-UTS, and SWS.

We found that the number of short segments, the sub-segments in the course of fishing, the presence of more than one moving object in each fold, and the frequency of capturing new trajectory points can be four major factors for this failure.

The results of Wilcoxon tests on the Atlantic hurricanes dataset for six trajectory segmentation algorithm is shown in Table 6.3. A Wilcoxon test indicated that WSII with linear interpolation kernel and Naive Bayes binary classifier could significantly outperform all other experimented trajectory segmentation algorithm.

The results of Wilcoxon tests for WSII with kinematic interpolation kernel and random forest binary classifier on the Geolife dataset against five trajectory segmentation algorithms is displayed in Table 6.4. The median *harmonic mean* for WSII

Trajectory Segmentation Algorithms	Median	CBSMoT	GRASP UTS	SPD	SWS	WKMeans	WSII
CBSMoT	87.92	1.000	**0.049**	**0.000**	1.000	0.058	**0.008**
GRASP-UTS	86.01	**0.049**	1.000	**0.000**	**0.003**	0.449	**0.000**
SPD	37.00	**0.000**	**0.000**	1.000	**0.000**	**0.000**	**0.000**
SWS	87.90	1.000	**0.003**	**0.000**	1.000	**0.000**	**0.000**
WKMeans	86.02	0.058	0.449	**0.000**	**0.000**	1.000	**0.000**
WSII	**90.71**	**0.008**	**0.000**	**0.000**	**0.000**	**0.000**	1.000

Table 6.3: The Wilcoxson test's p-value results to compare the harmonic mean of WSII with different binary classifiers on the Atlantic hurricanes dataset. The p-values with significant differences are highlighted in bold.

algorithm on the Geolife dataset was 95.50 ($IQR = 0.85$), whereas the median for SWS algorithm was 93.55 ($IQR = 6.29$). The Wilcoxon test showed that the difference between WSII and SWS was significant ($p = 0.000$, $statistic = -3.47$). Furthermore, WSII gained significant higher harmonic mean in comparison to CB-SMoT ($p = 0.000$, $statistic = -3.55$). There was significant difference between WSII and SPD according to Wilcoxon test results ($p = 0.000$, $statistic = -3.77$). Moreover, the results of WSII had lower variations ($IQR = 0.85$) in comparison to other experimented algorithms.

A Wilcoxon test indicated that WSII with kinematic interpolation kernel and random forest binary classifier could significantly outperform all other experimented trajectory segmentation algorithms on the Geolife dataset.

Since one of our datasets (the Geolife dataset) has a low number of short segments, the increase in the performance of WSII on this dataset shows the importance of having segments longer than the sliding window. WSII cannot produce high-quality segments when the size of a segment is less than the sliding window size. To achieve the best performance of WSII, we need to set the frequency of capturing so that the shortest segment size be double the size of the sliding window. This is because the head and tail of error signal, $[ws/2]$ of the head and $[ws/2]+1$ of the rear of the error signal are not the best representation of interpolation method, Algorithm 5.2 lines 14 and 15.

The results of this comparison using harmonic mean of purity and coverage is presented in Figure 6.7. The results indicate that WSII provides higher-quality segments

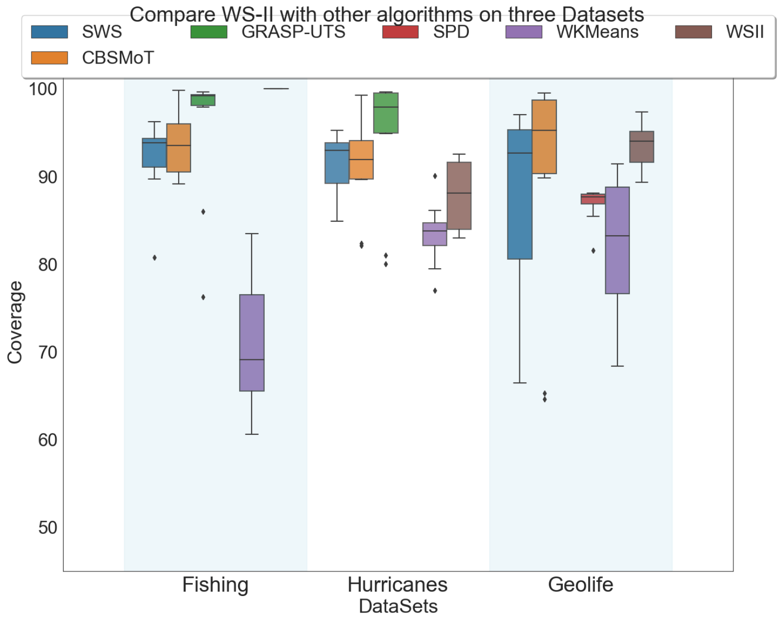


Figure 6.8: Comparing the results of coverage of WSII against five other trajectory segmentation algorithms on three different datasets.

on the Atlantic hurricanes dataset and the Geolife dataset. WSII did not gain the best performance on the fishing dataset. We attribute this to the weakness of WSII to handle short segments and the dataset properties. Figures 6.8 and 6.9 show the results of measuring purity and coverage. The WSII performs unsuccessfully in terms of coverage on the fishing dataset. We measured the number of discovered segments using WSII, shown in Figure 6.10.

WSII and SWS can be compared in the sense that both of them receive one fold

Trajectory Segmentation Algorithms	Median	CBSMoT	SPD	SWS	WKMeans	WSII
CBSMoT	91.58	1.000	0.596	0.289	0.058	**0.000**
SPD	91.74	0.596	1.000	**0.405**	**0.000**	**0.000**
SWS	93.55	0.289	**0.405**	1.000	**0.041**	**0.000**
WKMeans	86.84	0.058	**0.000**	**0.041**	1.000	**0.000**
WSII	**95.50**	**0.000**	**0.000**	**0.000**	**0.000**	1.000

Table 6.4: The Wilcoxson test's p-value results to compare the harmonic mean of WSII with different binary classifiers on the Geolife dataset. The p-values with significant differences are highlighted in bold.

of labeled data and the rest of the data without labels for the test. In the former the labeled fold is used to train the binary classifier and in the latter the labeled fold is used for tuning the epsilon parameter. WSII performed better than SWS on the Geolife dataset and the Atlantic hurricanes dataset; whereas, it produced unfortunate results on the fishing dataset. The properties of the dataset and the nature of fishing were significant factors of this weakness.

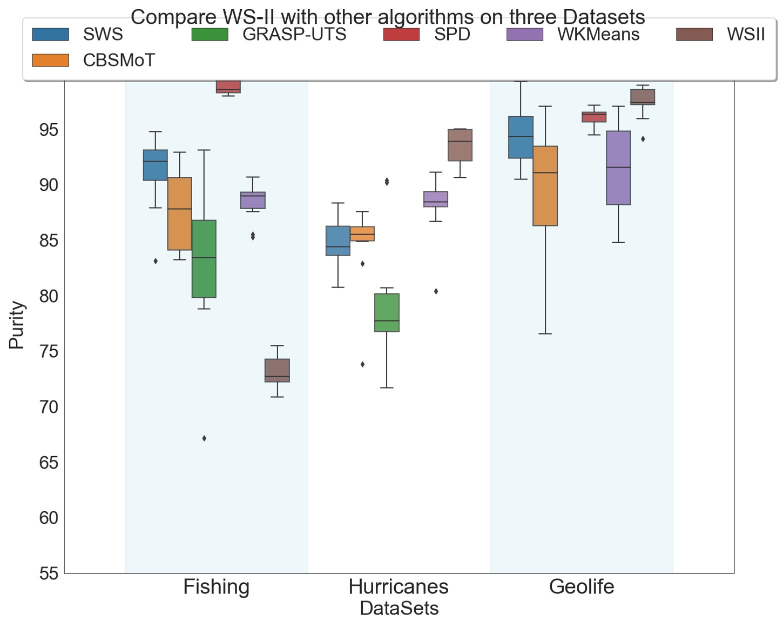


Figure 6.9: Comparing the results of purity for WSII against five other trajectory segmentation algorithms on three different datasets.

6.3 Discussion

WSII is a supervised trajectory segmentation algorithm. This algorithm relies on the labelled data provided for its training. One application of a supervised trajectory segmentation algorithm is in trajectory tagging applications such as VISTA [68]. In these applications, a visual representation of a trajectory is provided to a subject matter expert. The task is that the user finds the partitioning positions by using visual tools available on the platform. Visualization of many trajectory points and functionality of zooming in and out on the map to find the best segments is a tedious task for subject matter experts. A supervised trajectory segmentation can gradually

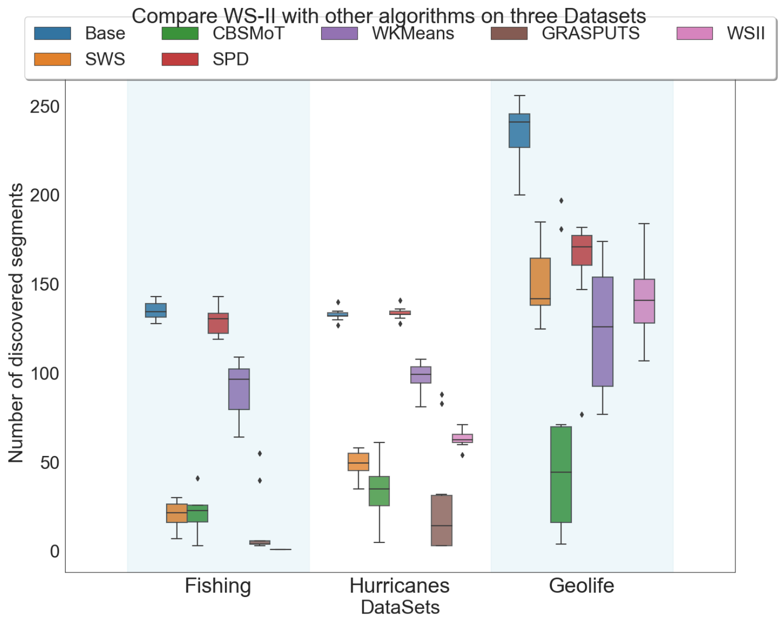


Figure 6.10: Comparing the number of discovered segments with ground truth using six trajectory segmentation algorithms on three datasets.

learn from the subject matter as they provide more labels to predict and suggest the next best partitioning positions. The complexity of this task is high when dealing with trajectories with a higher frequency of capturing new points. WSII is designed to assist subject matter experts in this situation.

The period of capturing a new trajectory point affects the performance of the WSII, which comes from SWS. We observed that this algorithm did not perform well on the fishing dataset. We think the following could be the reasons that limit WSII. First, there is a lot of short segments in the fishing dataset. WSII is working based on the sliding window and majority vote. Therefore, it has limitations on discovering short trajectory segments. Second, the properties of a dataset, such as the frequency of capturing a new trajectory point and its variation, can affect the quality of WSII. Third, the fishing dataset includes multiple moving objects in each fold. This makes WSII confused in the tuning phase. This kind of datasets can be processed in a way that their time overlap issue is resolved before using WSII.

6.4 Summary and Concluding Remarks

In this chapter, we proposed a supervised solution for a trajectory segmentation task called WSII. In this approach, the algorithm requires labelled data to train the core binary classifier. Experiments showed that this approach could segment trajectories to high-quality segments. However, this algorithm has two weaknesses: 1) WSII cannot process segments shorter than the sliding window size. 2) WSII requires a high frequency of capturing data. Despite its weaknesses, WSII is the first supervised trajectory segmentation algorithm that can be adjusted to the domain by providing labelled data.

6.5 Guideline

We provide a guideline for an end-user to choose a trajectory segmentation algorithm based on the available data and the application domain. We remark that the data used in this research is sampled at irregular intervals, and any sampling at a regular rate can improve the quality of segmentation. However, sampling at a regular rate is not common in mobility data since we have jump noise and poor GPS signal coverage since moving objects cannot communicate with GPS satellites.

The first factor in evaluating for selection of a trajectory segmentation algorithm is the type of trajectory data. If the trajectory data is in the form of multi-object trajectory data and there is more than one moving object in the dataset, there are two possible options. First, utilizing an algorithm designed for such data considers the collective behaviour of moving objects such as TRACLUS. Second, converting the multi-object trajectory data to a single object trajectory dataset so that algorithms designed for single moving objects such as SPD, CBSMoT, GRASP-UTS, SWS, WSII can be utilized. This conversion must assure the user that there is no time conflict in moving two objects. SPD, CBSMoT, GRASP-UTS, SWS, WSII, WKMeans are designed for single moving object trajectory datasets. When our trajectory data is sparse such as datasets for point of interest, using SPD and GRASP-UTS can produce high-quality segments. In these cases, algorithms such as SWS, WSII, CBSMoT and WKMeans do not perform well because they rely on sliding window speed and capturing trajectory points.

When the trajectory data comes with some semantic knowledge and labels, algorithms such as GRASP-UTS can benefit from the extra information. In cases where a subject matter expert is interactively working with a trajectory dataset such as VISTA [68], WSII can benefit from learning patterns of subject matter expert segmentation and facilitate trajectory tagging tasks. The current version of the VISTA platform benefits from the visualization of point features. This platform can suggest to users where the partitioning position could be after collecting some labelled data from interaction with a subject matter expert. In cases that trajectory data does not have any semantic knowledge, SWS, CBSMoT, or SPD exhibit reasonable performance. SPD and CBSMoT require parameter tuning, and the parameter tuning for them can be challenging, depending on the available tuning data. CBSMoT and SPD are more domain-specific, and using them in a new domain requires more experiments. Whereas, SWS can perform well in most situations independent from the domain because of its adaptability to a proper kernel domain, this algorithm uses limited memory, and it can be used in an edge environment such as a raspberry pi.

Selecting SWS as a general approach for unsupervised segmentation requires some consideration that we discuss here. For a general-purpose domain, SWS can produce reasonably good segments using linear interpolation with epsilon equal to percentile 95. we suggest using this epsilon because this value provided reasonably good segments on average in all of our experiments on all datasets. Selecting a suitable kernel for each domain can increase the quality of the produced segments. Our experiments suggest that a linear regression kernel with window size seven is the right choice for the fishing dataset. Moreover, the experiments did not support the use of the cubic kernel for this domain. Experiments on the Atlantic hurricanes dataset showed that the cubic kernel is the right choice for this application. In the transportation domain, the best kernel is the kinematic kernel with a window size of seven. The experiments strongly supported the use of the kinematic kernel in this application. In the end, considering the weakness of SWS in working with a sparse dataset and being sensitive to the quality of capturing is vital. Short segments cannot be processed well by these SWS and WSII algorithms because of the use of a sliding window. Decreasing the size of the sliding window depends on the applied interpolation kernel. The minimum window size is three for linear interpolation and seven for kinematic interpolation. In cases where a segment does not have enough trajectory points, we suggest increasing the frequency of capturing new trajectory points to overcome this issue. We recommend avoiding the application of the approach presented here in a given application that includes segments shorter than the window size.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this work, we studied the trajectory segmentation task — a pre-processing task for trajectory mining — in which a trajectory is split into smaller parts. Applying trajectory segmentation algorithms helps researchers to extract a graph representation for trajectory data. This representation facilitates the discovery of hidden patterns in trajectories. It answers queries on: graph topology features, edges features, vertices features and similarity features, including but not limited to the following questions. What are the trending stop points over the years? What are the predictions on an edge or vertices feature? What are the abnormal movement behaviours? What are the top n most connected vertices over the years, and what are the changes from one year to the next?

Moreover, such segmentation algorithms can be beneficial for researchers and subject matter experts interested in annotating trajectories. An example is the use of trajectory segmentation algorithms in a platform like VISTA [68].

We reviewed available trajectory segmentation algorithms and discussed their properties from four perspectives, including learning, features, performance, and other aspects. We identified two gaps in the literature of trajectory segmentation. First, an unsupervised single moving object trajectory segmentation algorithm that 1) uses only geolocation of moving object and does not rely on any particular knowledge, and 2) consumes low memory and CPU time, during the discovery of segments. We followed the idea that such an algorithm must discover segments when a moving object changes its behaviour. Such an algorithm can be domain-independent because it just utilizes the geolocation of moving objects. To cover this gap, we proposed Sliding Window Segmentation (SWS) as a single moving object unsupervised trajectory segmentation algorithm that discovers segments by identifying behaviour changes in the course of movement, detailed in Chapter 5. SWS generates a geolocation error signal as an intermediate step to find segments. This error signal represents possible partitioning positions where a moving object changes its behaviour and is used to split trajectory data into segments. The proposed model is flexible concerning different domains by adjusting an interpolation method, which is the kernel of SWS. In the course of searching for this trajectory segmentation algorithm, we developed two trajectory mining libraries in Python (TrajSeg[1], TrajLib[2]) to segment trajectories and generate features for each segment. We proposed some applications of trajectory segmentation in extracting a graph representation for trajectory data [11]. We compared the quality of discovered segments by SWS against four other segmentation algorithms (GRASP-UTS, CBSMoT, SPD, WKMeans) on three datasets from separate domains (the fishing dataset, the Atlantic hurricanes dataset, and the Geolife dataset).

The experimental results showed that SWS significantly increased the overall performance of trajectory segmentation on all datasets. Although the results of *harmonic mean* show that SWS and CBSMoT generating the same quality segments, the number of discovered segments, amount of memory and CPU consumption and the v-measure suggested that SWS identified more high-quality segments with less memory and CPU usage. Our experiments showed that the best interpolation kernels of SWS are as follows:

- For the fishing dataset, the best kernel is the linear regression, with no significant difference than the linear kernel.
- For the Atlantic hurricanes dataset the best kernel is the cubic kernel.
- For the Geolife dataset, the best kernel is the kinematic kernel.

Second, we could not find any supervised trajectory segmentation algorithm to facilitate the trajectory annotation task. To cover this gap, we proposed a supervised trajectory segmentation algorithm called Wise Sliding Window Segmentation (WSII), detailed in Chapter 6. WSII applies the idea of discovering segments using change detection in a trajectory, a binary classifier, and a majority vote decision-making process. In the course of research for the trajectory segmentation algorithm,

[1]`https://github.com/metemaad/TrajSeg`
[2]`https://github.com/metemaad/TrajLib`

we captured a rich trajectory dataset from Automatic Identification System (AIS) messages in partnership with AISHUB. We applied a subset of this dataset (published under the common creative licence to facilitate research in this domain) for debugging the WSII algorithm. During this research, we presented a platform for trajectory annotation called VISTA (see Appendix C.1 for more information) that can benefit from a supervised trajectory segmentation algorithm. We compared the results of segmentation using WSII with five other segmentation algorithms (SWS, GRASP-UTS, CBSMoT, SPD, WKMeans) on three datasets from different domains (the fishing dataset, the Atlantic hurricanes dataset, and the Geolife dataset).

The experimental results showed the WSII boosted the performance of the trajectory segmentation task on the Atlantic hurricanes dataset and the Geolife dataset. A Wilcoxon test indicated that the performance boost of WSII on the Atlantic hurricanes dataset and the Geolife dataset in comparison to the fishing dataset are significant. WSII was not able to increase the performance of the trajectory segmentation task on the fishing dataset. We consider contributing factors such as dataset properties and the presence of short segments as the reason for this limitation. WSII comes in handy for helping subject matter experts to segment trajectories with high frequency. WSII requires a capturing rate in which we receive at least twice the window size for each segment. Moreover, it can not process trajectory segments shorter than its window size.

7.2 Limitations

We evaluated both proposed algorithms (SWS and WSII) to find their weaknesses and the situation they can become most useful by conducting various experiments. Before we review the weaknesses of our proposed algorithms, we discuss the situation in which the algorithms are designed for them. Our proposed method is designed for a single moving object to segment its trajectory of movement. Our algorithm is intended to be fast and to consume as little memory as possible to run on a moving object platform (for example, a raspberry pi) with a limited power source and processing power; the experiment in Section 5.4.7 indicated that SWS requires less memory and CPU time than CBSMoT, SPD, and GRASP-UTS. In general, a moving object does not need to share the full trajectory of movement information after applying segmentation. It can benefit by sharing some summary statistics about each segment to increase its privacy, which can be a future work. In this situation, our moving object has real-time access to variables such as time and geolocation. SWS is a single moving object unsupervised trajectory segmentation algorithm that is proposed in Chapter 5. This algorithm has two key weaknesses. First, the algorithm requires a sampling rate in which the behaviour changes in moving objects be identifiable. A low sampling rate could result in missing evidence of behaviour change, which is the only knowledge that is provided to our method. Second, SWS identifies behaviour changes which may not follow a specific semantic. We observed this limitation for SWS in segmenting the fishing dataset. This limitation is because the ground truth are segmented in the level of Fishing and Non-fishing, where there are some behaviour changes in the course of fishing activity, detailed in Section 5.6 and Figure 5.18.

Our second proposed method (WSII) is a single moving object supervised trajectory segmentation aim for facilitating the trajectory annotation task. This algorithm inherits the weaknesses of SWS because they are using the same idea in their kernel. WSII requires a reasonable sampling rate to be functional because it has two sliding windows and a majority vote process. Having short segments in ground truth is an indication of not enough captured trajectory points. WSII cannot identify segments in such a situation. Moreover, this algorithm generates an error signal in its kernel. Therefore, the sampling rate is critical for this algorithm.

7.3 Future work

As future work, trajectory segmentation research can continue in three following categories: 1) algorithmic and implementation enhancements 2) evaluation methods, 3) applications.

Here, we discuss five future work on algorithmic and implementation enhancements of the presented approaches. First, future research ideas focus on improving the quality of SWS and WSII by removing their weaknesses. We may be able to learn from the amount of available data to train interpolation models for each domain. For example, using AIS data, we may be able to train a deep neural network model to extrapolate the next trajectory point. Such a model can be applied as the SWS kernel. We can improve WSII and SWS by studying a post-processing mechanism that merges similar trajectories based on the semantic definitions. For example, the post-processing can combine three behaviours identified by SWS and recognize them as one course of behaviour.

Second, trajectory segmentation algorithms need to be implemented in a stream and online environment. These methods can benefit from parallel processing (executing on NVIDIA Jetson Nano or Google Coral, or similar IoT devices with the power of parallel processing) and become much faster.

Third, trajectory segmentation has a focus on segmenting trajectories without any overlap. In a real-world situation, we may see some overlaps, such as the trajectory of a user travelling in a subway. The subway stops at a station, but the movement of the passenger might ahead or with delay.

Fourth, trajectory segmentation with considering the overlap between segments has not been studied exhaustively. We just observed one research that indirectly pointed to this subject. In some types of movement, the moving object gradually changes its behaviour from one segment to another segment that makes it hard to find an exact cutting point. For example, while a person starts walking in a train before departure. We think some solutions using fuzzy logic membership can open opportunities to have a segmentation while segments can overlap.

Fifth, when we introduced the kinematic movement, we did not consider the weight of the moving object. Including the weight of a moving object in a kinematic interpolation can produce more accurate results in cases that moving object follows the rules of physics. This can be enriched by adding more parameters such as fraction force of the type of surface that moving object moves there.

Our work can be extended by doing research on the evaluation methods for trajectory segmentation tasks. The metrics applied to evaluate the quality of segments can improve. A trajectory segmentation that has mistakes in the head or rear of a trajectory may be punished less than a trajectory segmentation algorithm with mistakes in the middle of a trajectory, resulting in breaking a trajectory to two.

The last category of our future work is the applications of trajectory segmentation to various tasks. The following are seven applications of trajectory segmentation that can extend our work.

First, the study of trajectory annotation can benefit from embedding WSII in a platform similar to VISTA. This can create a new generation of trajectory tagging platforms that can learn from their users. Studying the possibility of transfer learning and active learning in such platforms can be another branch of this study.

Second, we provide many questions, Section 7.1, on trajectory mining that can be answered after segmenting a trajectory. These are studies based on the topology of the graph representation extracted from a trajectory.

Third, our proposed trajectory segmentation can be applied to text mining to segment a text written by an author (texttiling) [34]. We can use word similarity functions as an interpolation method in our proposed work, such as cosine similarity between the vectorized version of each word extracted from a word embedding such as GloVe. The generated error value can be processed in the same way as we proposed in SWS. When the writer switch from one subject to the next topic, the error signal may show this difference because of dissimilarity between two sections. Therefore, the text can be segmented.

Forth, calculating the error signal from a kinematic interpolation method can be applied in an application that helps regulators to find the best place to place road signages such as speed limit postage. The idea behind this application is when the moving object error signal increases from the kinematic interpolation, the moving object might deviate from the road. Therefore, the moving object requires to change its behaviour. Accordingly, the speed limit signage can place a little bit before each identified segment. In a railway application, these places can be identified to place some security sensors to control the train's speed to automatically decrease the speed of the train by mechanical movements in the rails.

Fifth, recently, the insurance industry has started to leverage the use of telematics and mobility data. An example of that is the use of sharp braking by a driver in the calculation of car insurance premiums. Another indicator can be driven from our proposed method, which is the average area under the error signal for each driver. When this area is higher for a driver, this means that driver drives in a way that is far from normal kinematic. Hence the more possibility of going off the road or cause loss. Therefore, research on this topic can devise new measures for evaluating the quality of driving for each driver.

Sixth, trajectory of hand movements while using a wearable device like a watch can be a novel security method. Trajectory segmentation can be applied to find segments in each movement to trigger an event such as unlocking a car, signing a transaction, or performing an artistic movement. A car security key can be a pattern of hand movement that the user performs to unlock a car. Finding such a pattern is possible when we can segment a movement and compare segments with a recorded signature. This can be used to trigger any event using wearable devices. For example, a firework artist can use these patterns to magically commence an event such as changing the projector colour or starting a ventilation system to produce smoke and make a foggy scene.

Seventh, a trajectory segmentation algorithm application could be applied to the process of heart rate variability signals, captured during a therapeutic session such as massage therapy, physiotherapy, or osteopathy. In this application, the normal pattern of heart rate variability of a patient can be captured before starting the therapeutic session. This information can be used as an interpolation in our proposed algorithms. Therefore, the signal captured during the session can be segmented, and the distance of each segment be calculated from the normal. In this way, a health practitioner can identify which segments in the session are more healing for a patient.

Bibliography

[1] Charu C Aggarwal and S Yu Philip. A general survey of privacy-preserving data mining models and algorithms. In *Privacy-preserving data mining*, pages 11–52. Springer, 2008.

[2] David W Aha. *Lazy learning.* Springer Science & Business Media, 2013.

[3] Luis Otavio Alvares, Vania Bogorny, Bart Kuijpers, Jose Antonio Fernandes de Macedo, Bart Moelans, and Alejandro Vaisman. A model for enriching trajectories with semantic geographical information. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, page 22. ACM, 2007.

[4] Luis Otavio Alvares, Vania Bogorny, Bart Kuijpers, Jose Antonio Fernandes de Macedo, Bart Moelans, and Alejandro Vaisman. A model for enriching trajectories with semantic geographical information. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, page 22. ACM, 2007.

[5] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod record*, volume 28, pages 49–60. ACM, 1999.

[6] Fateha Khanam Bappee, Amílcar Soares Júnior, and Stan Matwin. Predicting crime using spatial features. In Ebrahim Bagheri and Jackie C.K. Cheung, editors, *Advances in Artificial Intelligence*, pages 367–373, Cham, 2018. Springer International Publishing.

[7] Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[8] Azedine Boulmakoul, Lamia Karim, and Ahmed Lbath. Moving object trajectories meta-model and spatio-temporal queries. *arXiv preprint arXiv:1205.1796*, 2012.

[9] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[10] Ricardo JGB Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(1):5, 2015.

[11] Emanuele Carlini, Vinicius Monteiro de Lira, Amilcar Soares, Mohammad Etemad, Bruno Brandoli Machado, and Stan Matwin. Uncovering vessel movement patterns from ais data with graph evolution analysis. In *Proceedings of the Workshops of the EDBT/ICDT 2020 Joint Conference, EDBT/ICDT 2020,*, 2020.

[12] Pablo Samuel Castro, Daqing Zhang, Chao Chen, Shijian Li, and Gang Pan. From taxi gps traces to social and community dynamics: A survey. *ACM Computing Surveys (CSUR)*, 46(2):17, 2013.

[13] Nitin R Chopde and Mangesh K Nichat. Landmark based shortest path detection by using a* and haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering*, 1(2):298–302, 2013.

[14] Sina Dabiri and Kevin Heaslip. Inferring transportation modes from gps trajectories using a convolutional neural network. *Transportation research part C: emerging technologies*, 86:360–371, 2018.

[15] Maria Luisa Damiani, Fatima Hachem, Hamza Issa, Nathan Ranc, Paul Moorcroft, and Francesca Cagnacci. Cluster-based trajectory segmentation with local noise. *Data Mining and Knowledge Discovery*, pages 1–39, 2018.

[16] Erico N de Souza, Kristina Boerder, Stan Matwin, and Boris Worm. Improving fishing pattern detection from satellite ais using data mining and machine learning. *PloS one*, 11(7):e0158248, 2016.

[17] Renata Dividino, Amilcar Soares, Stan Matwin, Anthony W Isenor, Sean Webb, and Matthew Brousseau. Semantic integration of real-time heterogeneous data streams for ocean-related decision making. In *Big Data and Artificial Intelligence for Military Decision Making*. STO, 2018.

[18] Yuki Endo, Hiroyuki Toda, Kyosuke Nishida, and Akihisa Kawanobe. Deep feature extraction from trajectories for transportation mode estimation. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 54–66. Springer, 2016.

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

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

[21] Mohammad Etemad, Zahra Etemad, Amilcar Soares, Vania Bogorny, Stan Matwin, and Luis Torgo. Wise sliding window segmentation: A classification-aided approach for trajectory segmentation. In *Advances in Artificial Intelligence: 33st Canadian Conference on Artificial Intelligence, Canadian AI 2020, Ottawa, ON, Canada, May , 2020, Proceedings 33*. Springer, 2020.

[22] Mohammad Etemad, Amílcar Soares Júnior, Arazoo Hoseyni, Jordan Rose, and Stan Matwin. A trajectory segmentation algorithm based on interpolation-based change detection strategies. In *Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference, EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019.*, 2019.

[23] Mohammad Etemad, Amilcar Soares, Elham Etemad, Jordan Rose, and Stan Matwin. Sws: An unsupervised trajectory segmentation algorithm based on change detection with interpolation kernels. *GeoInformatica*, 2020.

[24] Mohammad Etemad, Amílcar Soares, Stan Matwin, and Luís Torgo. On feature selection and evaluation of transportation mode prediction strategies. In *Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference, EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019.*, 2019.

[25] Mohammad Etemad, Amílcar Soares Júnior, and Stan Matwin. Predicting transportation modes of gps trajectories using feature engineering and noise removal. In *Advances in Artificial Intelligence: 31st Canadian Conference on Artificial Intelligence, Canadian AI 2018, Toronto, ON, Canada, May 8–11, 2018, Proceedings 31*, pages 259–264. Springer, 2018.

[26] Mohammad Etemad, Nader Zare, Amilcar Soares, Bruno Brandoli Machado, and Stan Matwin. Using deep reinforcement learning methods for autonomous vessels in 2d environments. In *Advances in Artificial Intelligence: 33rd Canadian Conference on Artificial Intelligence, Canadian AI 2020, Ottawa, ON, Canada, May 13–15, 2020, Proceedings*, page 220. Springer Nature, 2020.

[27] Shanshan Feng, Gao Cong, Bo An, and Yeow Meng Chee. Poi2vec: Geographical latent representation for predicting future visitors. In *AAAI*, pages 102–108, 2017.

[28] Miao Gao, Guoyou Shi, and Shuang Li. Online prediction of ship behavior with automatic identification system sensor data using bidirectional long short-term memory recurrent neural network. *Sensors*, 18(12):4211, 2018.

[29] Yikai Gong, Richard O Sinnott, and Paul Rimba. Rt-dbscan: Real-time parallel clustering of spatio-temporal data using spark-streaming. In *International Conference on Computational Science*, pages 524–539. Springer, 2018.

[30] Arthur Gretton, Kenji Fukumizu, and Bharath K Sriperumbudur. Discussion of: Brownian distance covariance. *The annals of applied statistics*, 3(4):1285–1294, 2009.

[31] Antonin Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[32] Frank R Hampel. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, 1974.

[33] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[34] Marti A Hearst. Text tiling: Segmenting text into multi-paragraph subtopic passages. *Computational linguistics*, 23(1):33–64, 1997.

[35] Samuel J Howarth and Jack P Callaghan. Quantitative assessment of the accuracy for three interpolation techniques in kinematic analysis of human movement. *Computer methods in biomechanics and biomedical engineering*, 13(6):847–855, 2010.

[36] Myeong-Hun Jeong, Seung-Bae Jeon, Tae-Young Lee, Min Kyo Youm, and Dong-Ha Lee. Vessel trajectory reconstruction based on functional data analysis using automatic identification system data. *Applied Sciences*, 10(3):881, 2020.

[37] Jungwook Jun, Randall Guensler, and Jennifer Ogle. Smoothing methods to minimize impact of global positioning system random error on travel distance, speed, and acceleration profile estimates. *Transportation Research Record: Journal of the Transportation Research Board*, 1(1972):141–150, 2006.

[38] Amílcar Soares Júnior, Chiara Renso, and Stan Matwin. Analytic: An active learning system for trajectory classification. *IEEE computer graphics and applications*, 37(5):28–39, 2017.

[39] Amilcar Soares Júnior, Valeria Cesario Times, Chiara Renso, Stan Matwin, and Lucídio AF Cabral. A semi-supervised approach for the semantic segmentation of trajectories. In *2018 19th IEEE International Conference on Mobile Data Management (MDM)*, pages 145–154. IEEE, 2018.

[40] M Kuhn, Horst Tomaschewski, and Hermann Ney. Fast nonlinear time alignment for isolated word recognition. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'81.*, volume 6, pages 736–740. IEEE, 1981.

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

[42] Luis A Leiva and Enrique Vidal. Revisiting the k-means algorithm for fast trajectory segmentation. In *ACM SIGGRAPH 2011 Posters*, page 86. ACM, 2011.

[43] Luis A. Leiva and Enrique Vidal. Warped k-means: An algorithm to cluster sequentially-distributed data. *Information Sciences*, 237(10):196–210, 2013. In Press.

[44] Luis A Leiva and Enrique Vidal. Warped k-means: An algorithm to cluster sequentially-distributed data. *Information Sciences*, 237:196–210, 2013.

[45] Qingyang Li, Nengchao Wang, and Dayi Yi. Numerical analysis. 1986. *Huazhong University of Science and Technology Publishing, China.*

[46] Lin Liao, Dieter Fox, and Henry Kautz. Location-based activity recognition. In *Advances in Neural Information Processing Systems*, pages 787–794, 2006.

[47] Hao Liu, Satoshi Oyama, Masahito Kurihara, and Haruhiko Sato. Landmark fn-dbscan: An efficient density-based clustering algorithm with fuzzy neighborhood. *Journal of Advanced Computational Intelligence Vol*, 17(1), 2013.

[48] Jed A Long. Kinematic interpolation of movement data. *International Journal of Geographical Information Science*, 30(5):854–868, 2016.

[49] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[50] MarineTraffic. Marinetraffic – a day in numbers. `https://www.marinetraffic.com/blog/a-day-in-numbers/`. Accessed: 2020-03-25.

[51] Ronaldo dos Santos Mello, Vania Bogorny, Luis Otavio Alvares, Luiz Henrique Zambom Santana, Carlos Andres Ferrero, Angelo Augusto Frozza, Geomar Andre Schreiner, and Chiara Renso. Master: A multiple aspect view on trajectories. *Transactions in GIS*, 2019.

[52] Anna Monreale, Roberto Trasarti, Dino Pedreschi, Chiara Renso, and Vania Bogorny. C-safety: a framework for the anonymization of semantic trajectories. *Trans. Data Privacy*, 4(2):73–101, 2011.

[53] Francisco Moreno, Anderson Castano, and Francisco Javier de Cos Juez. Spet algorithm: Stop and proximity episodes in trajectories. *Applied Mathematics & Information Sciences*, 9(2):549, 2015.

[54] Norman Morrison. *Introduction to sequential smoothing and prediction*. McGraw Hill, 2014.

[55] Movebank. Movebank website. `https://www.movebank.org/cms/movebank-main`,note=accessed:2020-03-25.

[56] Panagiotis Nikitopoulos, Aris-Iakovos Paraskevopoulos, Christos Doulkeridis, Nikos Pelekis, and Yannis Theodoridis. Hot spot analysis over big trajectory data. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 761–770. IEEE, 2018.

[57] Malay K Pakhira. A linear time-complexity k-means algorithm using cluster shifting. In *Computational Intelligence and Communication Networks (CICN), 2014 International Conference on*, pages 1047–1051. IEEE, 2014.

[58] Andrey Tietbohl Palma, Vania Bogorny, Bart Kuijpers, and Luis Otavio Alvares. A clustering-based approach for discovering interesting places in trajectories. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 863–868. ACM, 2008.

[59] Christine Parent, Stefano Spaccapietra, Chiara Renso, Gennady Andrienko, Natalia Andrienko, Vania Bogorny, Maria Luisa Damiani, Aris Gkoulalas-Divanis, Jose Macedo, Nikos Pelekis, et al. Semantic trajectories modeling and analysis. *ACM Computing Surveys (CSUR)*, 45(4):1–32, 2013.

[60] Lokukaluge P Perera, Carlos Guedes Soares, et al. Ocean vessel trajectory estimation and prediction based on extended kalman filter. In *The Second International Conference on Adaptive and Self-Adaptive Systems and Applications*, pages 14–20. Citeseer, 2010.

[61] Jose Antonio MR Rocha, Valéria C Times, Gabriel Oliveira, Luis O Alvares, and Vania Bogorny. Db-smot: A direction-based spatio-temporal clustering method. In *Intelligent systems (IS), 2010 5th IEEE international conference*, pages 114–119. IEEE, 2010.

[62] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007.

[63] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 410–420, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[64] P Samarati and L Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression (tech. rep. sri-csl-98-04). *CS Lab, SRI International*, 1998.

[65] R. W. Schafer. What is a savitzky-golay filter? [lecture notes]. *IEEE Signal Processing Magazine*, 28(4):111–117, July 2011.

[66] Amilcar Soares, Renata Dividino, Fernando Abreu, Matthew Brousseau, Anthony W Isenor, Sean Webb, and Stan Matwin. Crisis: Integrating ais and ocean data streams using semantic web standards for event detection. In *International Conference on Military Communications and Information Systems ICMCIS2019*. IEEE, 2019.

[67] Amílcar Soares, Renata Dividino, Fernando Abreu, Matthew Brousseau, Anthony W Isenor, Sean Webb, and Stan Matwin. Crisis: integrating ais and ocean data streams using semantic web standards for event detection. In *2019 International Conference on Military Communications and Information Systems (ICMCIS)*, pages 1–7. IEEE, 2019.

[68] Amílcar Soares, Jordan Rose, Mohammad Etemad, Chiara Renso, and Stan Matwin. VISTA: A visual analytics platform for semantic annotation of trajectories. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, pages 570–573, 2019.

[69] Amílcar Soares Júnior, Bruno Neiva Moreno, Valéria Cesário Times, Stan Matwin, and Lucídio dos Anjos Formiga Cabral. Grasp-uts: an algorithm for unsupervised trajectory segmentation. *International Journal of Geographical Information Science*, 29(1):46–68, 2015.

[70] Stefano Spaccapietra and Christine Parent. Adding meaning to your steps (keynote paper). In *International Conference on Conceptual Modeling*, pages 13–31. Springer, 2011.

[71] Stephen C Stubberud and Kathleen A Kramer. Kinematic prediction for intercept using a neural kalman filter. In *Proceedings of the IFAC World Congress*, 2005.

[72] Georgios Technitis, Walied Othman, Kamran Safi, and Robert Weibel. From a to b, randomly: a point-to-point random trajectory generator for animal movement. *International Journal of Geographical Information Science*, 29(6):912–934, 2015.

[73] Evimaria Terzi and Panayiotis Tsaparas. Efficient algorithms for sequence segmentation. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 316–327. SIAM, 2006.

[74] Tammy M Thompson, Sebastian Rausch, Rebecca K Saari, and Noelle E Selin. A systems approach to evaluating the air quality co-benefits of us carbon policies. *Nature Climate Change*, 4(10):917, 2014.

[75] Waldo R Tobler. A computer movie simulating urban growth in the detroit region. *Economic geography*, 46(sup1):234–240, 1970.

[76] Le Hung Tran, Quoc Viet Hung Nguyen, Ngoc Hoan Do, and Zhixian Yan. Robust and hierarchical stop discovery in sparse and diverse trajectories. Technical report, inst, 2011.

[77] Glen Van Brummelen. *Heavenly mathematics: The forgotten art of spherical trigonometry*. Princeton University Press, 2012.

[78] Iraklis Varlamis, Konstantinos Tserpes, Mohammad Etemad, Amílcar Soares Júnior, and Stan Matwin. A network abstraction of multi-vessel trajectory data for detecting anomalies. In *Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference, EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019.*, 2019.

[79] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[80] Min Wang, Genlin Ji, Bin Zhao, and Mengmeng Tang. A parallel clustering algorithm based on grid index for spatio-temporal trajectories. In *2015 Third International Conference on Advanced Cloud and Big Data*, pages 319–326. IEEE, 2015.

[81] Zhixian Yan, Nikos Giatrakos, Vangelis Katsikaros, Nikos Pelekis, and Yannis Theodoridis. Setrastream: semantic-aware trajectory construction over streaming movement data. In *International Symposium on Spatial and Temporal Databases*, pages 367–385. Springer, 2011.

[82] Hyunjin Yoon and Cyrus Shahabi. Robust time-referenced segmentation of moving object trajectories. In *2008 Eighth IEEE International Conference on Data Mining*, pages 1121–1126. IEEE, 2008.

[83] Daiyong Zhang, Jia Li, Qing Wu, Xinglong Liu, Xiumin Chu, and Wei He. Enhance the ais data availability by screening and interpolation. In *Transportation Information and Safety (ICTIS), 2017 4th International Conference on*, pages 981–986. IEEE, 2017.

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

[85] Yu Zheng, Lizhu Zhang, Zhengxin Ma, Xing Xie, and Wei-Ying Ma. Recommending friends and locations based on individual location history. *ACM Transactions on the Web (TWEB)*, 5(1):5, 2011.

[86] Max Zimmermann, Thomas Kirste, and Myra Spiliopoulou. Finding stops in error-prone trajectories of moving objects with time-based clustering. In *Intelligent interactive assistance and mobile multimedia computing*, pages 275–286. Springer, 2009.

[87] Dimitrios Zissis, Elias K Xidias, and Dimitrios Lekkas. Real-time vessel behavior prediction. *Evolving Systems*, 7(1):29–40, 2016.

Appendix A

Appendix A

A high-level view of our platform which is designed to capture and consume AIS data is shown in Figure A.1. This application includes the source code that runs on stations to capture data, the communication platform to share data with the AISHUB community, Processing and storage of data, and the applications consuming this data. In the next sections we explain each component briefly.

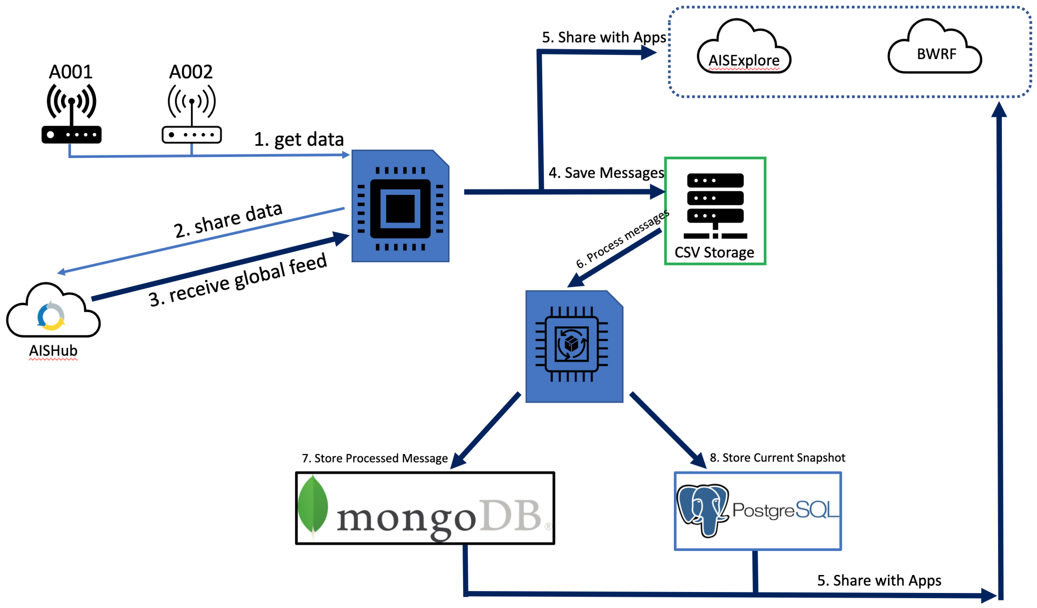


Figure A.1: A high-level view of our application to capture and use the AIS data.

A.1 Stations

A station is a terrestrial AIS antenna installed in a location that can capture AIS messages transmitted by vessels. Each station includes three main components: 1) antenna 2) processor 3) station firmware. We used a Shakespeare Galaxy Antenna for each station, which is a 243.8 x 7.6 x 7.6 cm omnidirectional antenna that works in 3 MHz within 2.0:1 VSWR. Each station's processor contains three hardware modules: a Raspberry Pi 4 Model B, dAISy HAT and a Raspberry Pi UPS HAT. The hardware is connected to the Internet. The station firmware is responsible for receiving ais messages and submit them to our central server. The firmware can catch received messages in memory when there is a problem in the network connection. The link between the station and the central server is secured by private and public key encryption that updated using elliptic curve cryptography. Each client submits a heartbeat signal periodically to the central server along with its statistics such as free memory, CPU usage and identified logs. The source code for a station firmware is available at https://git.cs.dal.ca/big-data-institute/client.

A.2 Central Server

This component of system is responsible for the following tasks: 1) communicating with stations and receive the submitted messages. 2) communicating with AISHUB to share the feed of collected AIS messages with the community. 3) receiving the shared feed of AIS messages from the AISHUB community. 4) sharing the real-time feed of Automatic Identification System (AIS) messages with applications such as AISExplore, or Ballast Water Reporting Form (BWRF). 5) Storing the raw AIS messages in CSV files for further processing 6) processing the heartbeat signals and providing an interface for monitoring all the system components.

A.3 Data Processor

This module is responsible for reading CSV files and process the AIS messages. Here we decode all AIS messages and perform more processing. The first step is to decode the raw AIS message and extract all the fields in each message. We process message five separately and store them in a P-SQL database for fast processing. We store all the decoded information in a MongoDB instance, which maintains spatial and temporal indexing for AIS messages. We applied a GPU processing with the help of level DB to store the latest location of each vessel around the globe for realtime visualization of AIS messages. A link to our processed data is provided to the application center, including AISExplore, with rest APIs function.

A.4 Applications

The platform is designed to provide AIS data to some applications using rest APIs functions. AISExplore is an application that is designed to consume this data.

A.4.1 AISExplore

This application consumes AIS messages for visualization and answering to users' AIS related queries. AISExplore receives the latest location of each vessel using rest APIs and provides an interface that the user can interact with the system.

Appendix B

Appendix B

B.1 Parameter Selection Tables

	Dataset	**Kernel**	**Window size**	**E-threshold**	**Selected Percentile**
Fold 1	Fishing	Linear Regression	11	11953.61	99
Fold 2	Fishing	Linear Regression	11	12604.23	99
Fold3	Fishing	Linear Regression	11	922983.78	98.5
Fold 4	Fishing	Linear Regression	11	483383.78	99
Fold 5	Fishing	Linear Regression	11	75376.38	90
Fold 6	Fishing	Linear Regression	11	9457.08	97.5
Fold 7	Fishing	Linear Regression	11	10626.88	98.5
Fold 8	Fishing	Linear Regression	11	15672.23	99
Fold 9	Fishing	Linear Regression	11	12648.73	98.5
Fold 10	Fishing	Linear Regression	11	10721.34	97

Table B.1: Parameters applied in experiments for on the fishing dataset for SWS

	Dataset	**area**	**min_time**	**time_tolerance**	**merge_tolerance**
Fold 1	Fishing	0.1	360	0	0
Fold 2	Fishing	0.1	360	0	0
Fold 3	Fishing	0.9	360	0	0
Fold 4	Fishing	0.9	360	0	0
Fold 5	Fishing	0.1	3600	0	0
Fold 6	Fishing	0.1	360	0	0
Fold 7	Fishing	0.01	360	0	0
Fold 8	Fishing	0.01	360	0	0
Fold 9	Fishing	0.1	360	0	0
Fold 10	Fishing	0.3	3600	0	0

Table B.2: Parameters applied for CBSMoT

	Dataset	**alpha**	**partitioning factor**	**max iterations**	**min time**	**jcs**
Fold 1	Fishing	0.3	0	10	6	0.7
Fold 2	Fishing	0.7	0	20	6	0.3
Fold 3	Fishing	0.7	0	10	60	0.7
Fold 4	Fishing	0.3	0	10	6	0.3
Fold 5	Fishing	0.7	0	20	60	0.7
Fold 6	Fishing	0.3	0	10	6	0.7
Fold 7	Fishing	0.3	0	20	60	0.7
Fold 8	Fishing	0.5	0	30	6	0.7
Fold 9	Fishing	0.3	0	10	6	0.7
Fold 10	Fishing	0.7	0	20	360	0.7

Table B.3: Parameters applied for GRASPUTS on the fishing dataset.

B.2 Comparing Haversine and Euclidean distance

Figure B.1 shows the difference between using haversine and euclidean formula to calculate the distance between two points on the earth. The x-axis increases from 0 to 180, which is representing the longitude of a point. We measured the distance of these points with latitude between 0 to 90 and longitude between 0 to 180 using haversine and euclidean formula. Then we calculate the average difference between these two numbers for each longitude. The red line shows this difference, and the y-axis unit is kilometre. The graph shows that if the distance between two points is less than 25 degrees, the error of the euclidean formula is minute.

	Dataset	ThetaTimeParam	ThetaDistancePara
Fold 1	Fishing	2000	6000
Fold 2	Fishing	60	6000
Fold 3	Fishing	2000	6000
Fold 4	Fishing	2000	6000
Fold 5	Fishing	2000	100
Fold 6	Fishing	2000	6000
Fold 7	Fishing	2000	6000
Fold 8	Fishing	60	6000
Fold 9	Fishing	60	6000
Fold 10	Fishing	2000	6000

Table B.4: Parameters applied for SPD on the fishing dataset.

	Dataset	Number of segments	delta
Fold 1	Fishing	171	1
Fold 2	Fishing	162	1
Fold 3	Fishing	198	1
Fold 4	Fishing	180	1
Fold 5	Fishing	81	1
Fold 6	Fishing	144	1
Fold 7	Fishing	126	1
Fold 8	Fishing	144	1
Fold 9	Fishing	72	1
Fold 10	Fishing	90	1

Table B.5: Parameters applied for WKMeans on the fishing dataset.

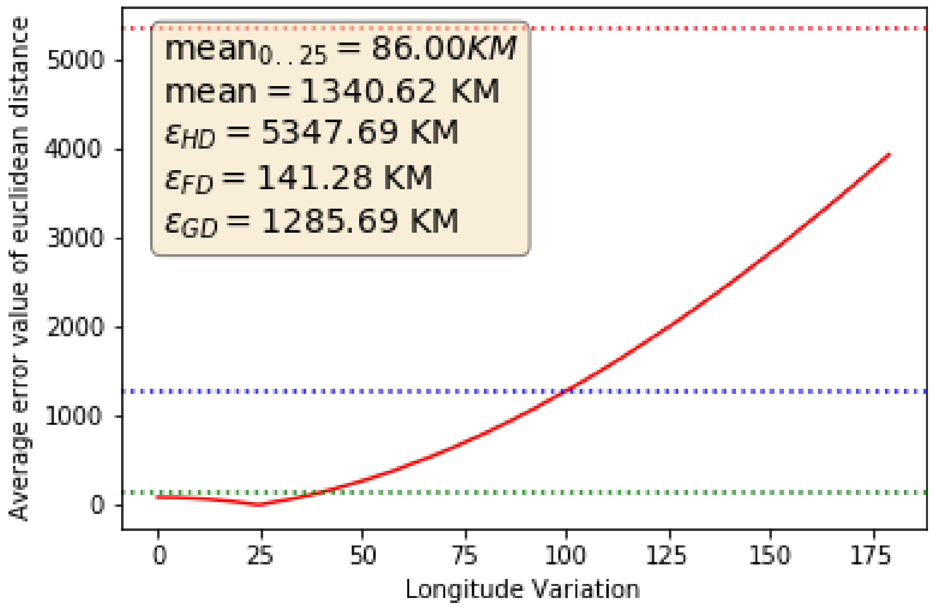


Figure B.1: Comparing the difference between haversine distance and Euclidean distance.

Appendix C

Appendix C

C.1 VISTA

VISTA is a visual analytics platform for semantic annotation of trajectories presented in [68]. Figure C.1 shows the architecture and workflow of this platform. This platform has three layers: 1) Data Collection, 2) Data Processing and, 3) Data Visualization. A raw trajectory and its related semantic information (Point Of Interests (POIs), Region of Interest(ROIs)) are submitted to the system in the Data Collection layer. In the data processing layer, VISTA provides some tools to generate point features and semantic features. During the data processing, VISTA benefits from our public trajectory mining library called TrajLib. After generating semantic and point features, the system provides a trajectory tagging session for annotator (a subject matter expert). Annotator, with the help of visualization tools provided by the platform, decides on where the partitioning position must be. Figure C.2 shows a screenshot of the annotator dashboard in the VISTA platform while a user in a tagging session for a vessel trajectory.

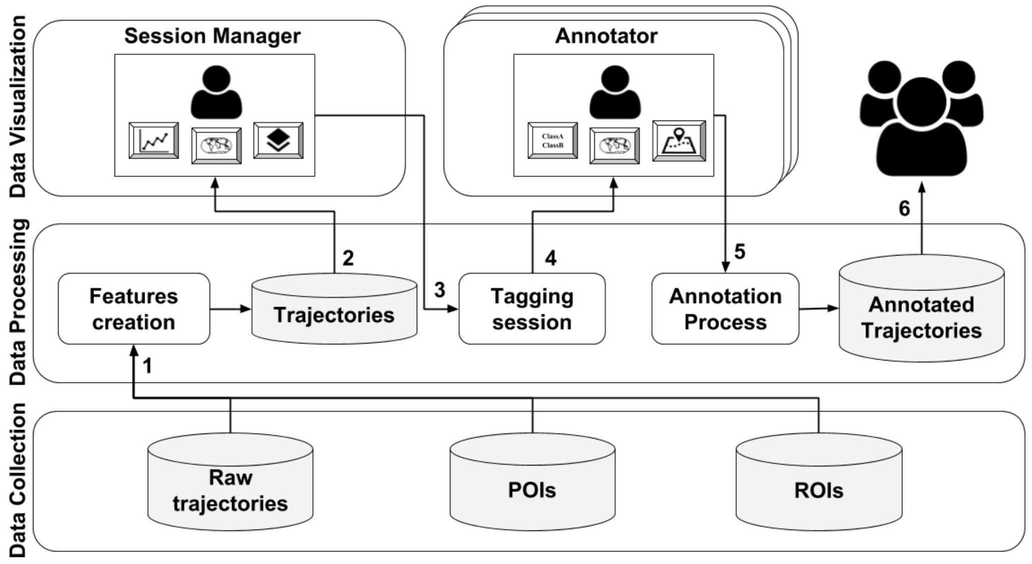
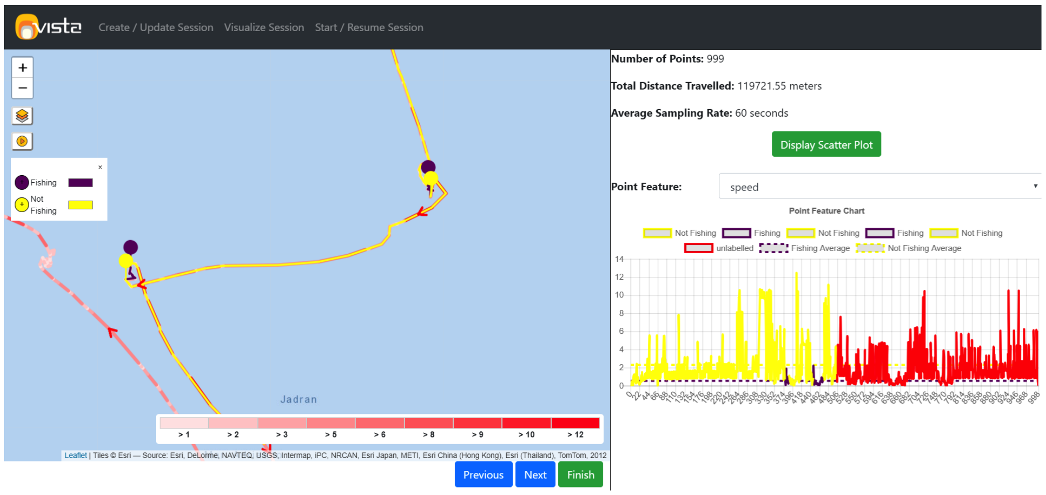


Figure C.1: The architecture and workflow of VISTA platform.



Number of Points: 999

Total Distance Travelled: 119721.55 meters

Average Sampling Rate: 60 seconds

Point Feature: speed

Figure C.2: A screenshot of the annotator user dashboard with the vessels trajectories dataset.