Adaptive Sampling Methods for Vehicle Trajectory Data

Choudhury Nazib Wadud Siddique

A dissertation

Submitted in partial fulfillment of the requirement for the degree of

Doctor of Philosophy

University of Washington

2019

Committee:

Xuegang (Jeff) Ban, Chair

Yinhai Wang

Edward McCormack

Program Authorized to Offer Degree:

Civil and Environmental Engineering

University of Washington

Abstract

Adaptive Sampling Methods for Vehicle Trajectory Data

Choudhury Nazib Wadud Siddique

Chair of the Supervisory Committee:

Xuegang (Jeff) Ban, Associate Professor

Department of Civil and Environmental Engineering

The ubiquitous use of smartphones and the emergence of new technologies such as ridesourcing and connected/automated vehicles provide new opportunities for mobile sensors in traffic monitoring and data collection. To make GPS based smartphones an effective and practical source of transportation data, one needs to address the multifaceted challenges related to mobile sensing. Since the mobile sensing technology requires individual sensors (often from end-users) sending location information periodically to the data collector (e.g., a server), one of such challenges is the storage and data transmission cost incurred to individual sensors/users, as well as the battery life of mobile sensors.

This research aims to balance the data transmission cost and the needs to collect detailed mobile sensing data by developing a method to resample the smartphone-based GPS data at the user side. The work introduces the concept of Vehicle Flow State (VFS) to explain the implicit nature of probe vehicle's motion. Then the work proposes a methodology which first estimates the vehicle flow state (VFS) of the sensor/vehicle from its trajectory data and then uses the estimated VFS to adjust the sampling rate of the trajectory accordingly.

The primary contributions of this work are as follows. First, this work develops the concept of vehicle flow state (VFS) and developed an HMM-based method to identify the VFS of an individual vehicle. Second, two self-adaptive sampling strategies for vehicle trajectory data are presented based on the identified VFS, which reduces the overall data size and transmission cost. Finally, this work presents comprehensive testing and validation of the proposed methods with real-world trajectory data. The methods and algorithms provided in this work will be of significant value to the server-side and the user/client side of a smartphone-based vehicle trajectory data collection system. The reduced data using proposed methods show a promising result in traffic modeling applications (such as queue length estimation) and the end user's privacy protection.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Dr. Jeff Ban for his continuous support during my graduate study. On the academic level, Dr. Ban taught me the fundamentals of conducting scientific research in the field of transportation engineering. Under his supervision, I learned how to define a research problem, find a solution to it, and publish the results. On a personal level, Dr. Ban inspired me by his hardworking and passionate attitude.

Besides my advisor, I would like to thank the rest of my doctoral committee: Prof. Yinhai Wang, Prof. Ed McCormack, and Prof. Qing Shen, for their insightful comments and encouragement, but also for the hard questions which incented me to widen my research from various perspectives.

I thank my fellow colleagues in iUTS lab: Jingxing Wang, Rong Fan, Wan Li, Qiangqiang Guo, and Yiran Zhang for their insight, support, and friendship. In particular, Wan Li has been a willing and capable partner for many of my collaborations both in UW and RPI. I would also like to thank my friends in the RPI, in particular, Asif Jahangir Choudhury and Aritra Bose for their friendship, and support during my time at RPI.

Finally, and most importantly, I would like to thank my wife, Palbasha. This dissertation would not have been possible without her warm love, continued patience, and endless support.

Chapter 1. INTRODUCTION

1.1. BACKGROUND AND MOTIVATION

The rapid advent of GPS-enabled mobile devices especially smartphones has created a new stream of location data. Approximately 77% of global mobile device connections in 2025 will be smartphones, a number that translates into 7 billion smartphone users worldwide (Statista, 2018). By 2025, there will be 116 million connected cars in the U.S., and according to one estimate by Hitachi (Quartz, 2015), each of those connected cars will upload 25 gigabytes of data to the cloud per hour. That is 219 terabytes each year, meaning roughly 25 billion terabytes of total connected car data each year. The rise of smartphones and the emergence of new technologies such as ridesourcing and connected/automated vehicles have led to a wide deployment of mobile sensors in transportation. The ability to provide extensive spatial and temporal coverage has made those mobile sensors an important source of traffic data (Wan et al., 2016). Among the first real-world applications of crowdsourced cell phone location data for traffic information, Google Maps began offering real-time traffic information in early 2007 (Crackberry, 2007). The availability and use of commercial probe vehicle data in traffic information systems and analysis increased in the following years. In 2013 HERE North America (formerly NAVTEQ/Nokia) began providing probe vehicle data for the entire national highway system under contract with the Federal Highway Administration (FHWA). This dataset, titled the National Performance Management Research Data Set (NPMRDS), was obtained through mobile phones, dedicated GPS, and embedded fleet systems and provided by the FHWA free of charge to metropolitan planning organizations (MPOs) and departments of transportation (DOTs) throughout the USA for performance monitoring and planning activities (FHWA Office of Operations 2013). Today, GPS based location data is widely used for almost all types of applications in transportation from traffic state estimation, performance

evaluation, to travel/mobility patterns and travel demand analysis, and decision making (Ban and Gruteser, 2012; Zheng, 2015).

To make mobile sensors especially GPS-equipped smartphones an effective and practical solution to transportation applications, one needs to address the multifaceted challenges related to mobile sensing. Since the mobile sensing technology usually requires individual sensors (often from end-users) sending location information periodically to the data collector (e.g., a server or cloud), one of such challenges is the storage and data transmission cost incurred to individual sensors/users (Meratnia and Rolf, 2004), as well as the battery life of mobile sensors (Wang et al., 2009). Figure 2.1 illustrates the schematics of GPS based location data collection process and related costs on client and server sides. The cost collecting such "crowdsourcing" location data depends on the sampling rate, i.e., how many data records to collect every second from individual sensors. Using GPS-enabled mobile sensors as an example, those sensors have the potential to collect one or multiple data records every second. However, if relatively high sampling rate (e.g., every second) is used, the large amount of data generated on the sensor side can incur expensive data transmission cost (Meratnia and Rolf, 2004). Over time, it may lead to a series of difficulties in storing, transmitting, and data analysis, as the size of trajectory data increases and the scale of data grows huge and complex. First of all, the sheer volumes of data can quickly overwhelm available data storage on the device (sensor) side, which will make it difficult to store the data. For instance, if data is collected at 2-second intervals, 1 GB of storage capacity is required to store just over 800 objects for a single day. Therefore, the storage of data will result in an enormous cost. The cost of transmitting a large amount of trajectory data, which may be expensive and problematic, is the second major problem. For example, tracking a fleet of 4,000 vehicles for a single day would incur a cost of $5,000 to $7,000, or approximately $1,800,000 to $2,500,000

annually, if data are collected every second (Muckell et al., 2014). This is an important issue even for nowadays when the concept and methods of "big data" have been proposed and used in many areas. As shown in Figure 1.1, although the data collectors (i.e., the server or the cloud side) usually have very powerful tools and systems that can deal with datasets with almost any size, *individual users* (*sensors*) typically do not have the interest to spend considerable resources (for storage and data transmission) related to mobile data collection.

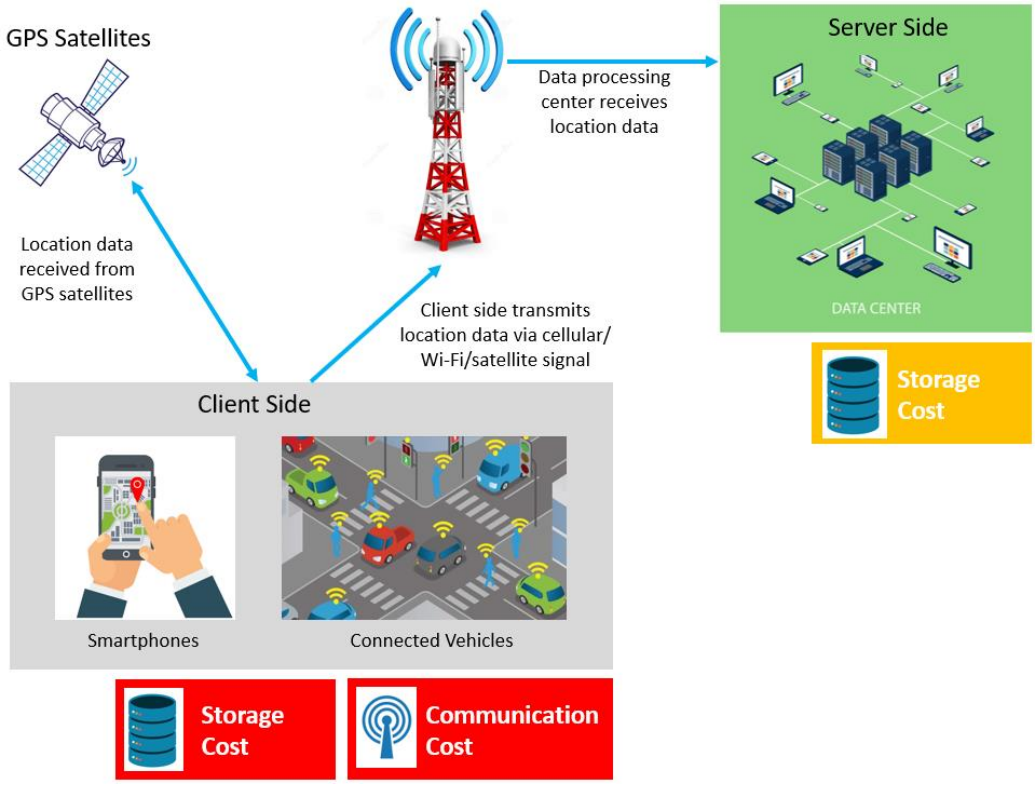


Figure 1.1. The schematics of GPS based location data collection process and related costs on server and client side

To balance the data transmission cost and the needs to collect detailed mobile sensing data, the current practice is to use relatively low sampling rate: e.g., 30 seconds or a few minutes or even longer for fleet vehicle monitoring (Chen et al., 2017) or a few seconds for ridesourcing vehicle

monitoring (e.g., the 3-second sampling rate is used by Didi, a ridesourcing company based in Beijing, China; see http://www.didichuxing.com/en/). Under a lower sampling rate, however, the collected data might not fully capture important vehicle moving conditions, resulting in lower accuracy or information loss. Therefore, there is a need to design adaptive sampling mechanisms for mobile data collection, which can achieve storage and data transmission cost savings while maintaining the accuracy level required for traffic/transportation applications. Note that battery use is one of the important concerns when collecting mobile data, which has also been studied in the past, e.g., by periodically turning off GPS sensors (Paek et al. 2010).

This dissertation research focuses on finding a better way to compress GPS based vehicle trajectory data from mobile sensors to reduce the data size. A trajectory is a path that a moving object follows through space as a function of time, which is one of the most important types of data one can collect via mobile sensors. Data compression aims to reduce the size of data to cut down the memory space and improve the efficiency of transmission, storage, and processing without severely losing critical information. When applied to trajectory data, data compression is also called trajectory compression. Figure 1.2 is a schematic diagram of trajectory compression, where the original trajectory is represented by black lines and the compressed trajectory consists of red lines (namely, S_1, S_2 ,S_3 , and S_4). There are 9 points in the original trajectory, but only 5 points are retained to approximately represent the original trajectory after compressing with a compression ratio close to 50%. Thus, it can be seen that trajectory compressions play an important role in the storage and analysis of data. Trajectory compression tends to cause a certain loss of information, and therefore, various trajectory algorithms existing in literature balance the tradeoff between accuracy and data reduction.

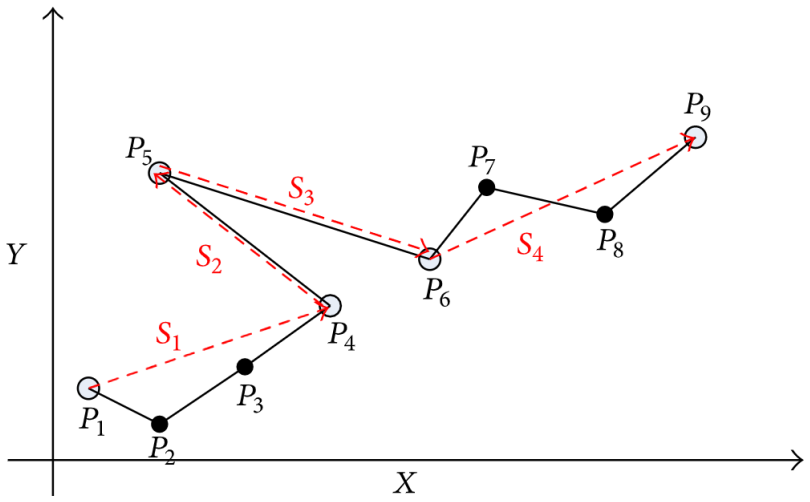


Figure 1.2. A schematic of the trajectory compression

In the literature of mobile data collection and management, most existing data reduction/compression techniques rely on the geometric properties of trajectory data (Keogh et al., 2001; Meratnia and Rolf, 2004; Lovrić et al. 2014). The existing trajectory compression algorithms can be categorized as either "batch" algorithms or "online" algorithms based on whether they can run in real-time or not. Batch algorithms compress a trajectory (i.e., resample the trajectory at a lower rate) after the original trajectory has been fully collected at a high sampling frequency (e.g., one data point every second) (Lee and Krumm, 2011; Lawson et al., 2011). Online trajectory compression algorithms work in real-time as the vehicle travels and the data are collected, such as the sliding window algorithm (Shatkay, 1995; Park et al., 1999). An important limitation of existing trajectory compression algorithms is that they are purely trajectory-geometry (shape) based, as they often use the perpendicular distance of a data point to a proposed generalized line as the condition to discard or retain the data point. This may result in inadequate data records in some cases as discussed in Zheng (2015)

In transportation, Lim et al. (2017) proposed a sampling scheme to collect real-time locations and speeds from in-vehicle GPS-based devices over a road network. The target scenario is in the connected vehicles (CV) environment when data from most vehicles may be collected for estimating the traffic state of a road network. The sampling method can be considered as a centralized scheme where data from most vehicles can be collected. In this case, the data collector (server) designs the entire sampling method with the objective to minimize the transmission cost overall vehicles while satisfying the accuracy in estimating average link speeds and the timeliness of the obtained traffic data. The sampling rates are then sent from the center to individual vehicles to sample vehicle-specific data. Such a centralized scheme may not be easily applicable to other applications, such as mobile app-based navigation systems (e.g., google map) that are currently widely deployed. For such systems, users/vehicles are more ad-hoc and dynamic and the systems only have access to data of a small fraction of the vehicles. For those systems, a light-weighted, decentralized sampling method is preferable to the centralized scheme.

1.2. RESEARCH OBJECTIVES

This dissertation aims to propose a decentralized method of collecting/sampling vehicle trajectories based on the state of individual vehicles. The goal of this research is, instead of just using the geometric properties of the trajectory, to focus on the intrinsic features of vehicle trajectories over time in the form of vehicle flow state (VFS) that can better reveal the underlying vehicle moving conditions. Such ‘traffic knowledge’ revealed by vehicle trajectories can be then used in vehicle trajectory compression to not only reduce the data size but also to make sure the reduced trajectory can be properly used for traffic/transportation applications.

In summary, the specific objectives of this research are:

- To develop and evaluate a method to better understand the traffic situation an individual vehicle is experiencing. This method will later provide a foundation for developing data reduction strategies.
- To develop and evaluate a classifier to identify the stop-and-go situation in traffic from the point of view of an individual vehicle.
- Testing and validating the VFS classifiers using a large dataset, where trajectories were collected in different traffic conditions.
- Propose two adaptive sampling methods for GPS based vehicle trajectory data based on the estimated vehicle flow state and identified stop-and-go information.
- Comprehensive performance evaluation of the reduced trajectory data (which is obtained after applying the proposed sampling methods) while compared to the results using a few existing trajectory compression algorithms.
- Testing and validating the effectiveness of reduced trajectory data (which is obtained after applying the proposed sampling methods) on traffic/transportation applications.
- Testing and validating the effectiveness of the reduced trajectory data in enhancing user's privacy.

1.3. RESEARCH CONTRIBUTIONS

The contributions of this research can be summarized as follows.

- A complete understanding of the motion of an individual vehicle in the form of Vehicle Flow State (VFS). This understanding will help to identify critical segments in a trajectory.

- A method to identify ‘stop and go’ state of an individual vehicle using location data from. The proposed method is from the perspective of an individual vehicle which is different from the existing methods on traffic oscillation where most studies considered vehicle platoons (i.e., multiple vehicles).
- Two novel data compression methods for resampling GPS based trajectory data. These methods utilize the estimated VFS of the trajectory and reconstruct the trajectory by only keeping the most critical data points.
- Extensive performance evaluation of the proposed trajectory resampling methods on multiple datasets which includes a large dataset with millions of trajectories collected from ridesourcing vehicles.
- Comprehensive performance evaluation of the reduced trajectory data (which is obtained after applying the proposed resampling methods) while compared to the results using a few other existing trajectory compression algorithms using multiple datasets.

1.4. Study Scope

This research is focused on solving a set of issues related to GPS based trajectory data collection and sampling. The trajectory data were collected using mobile devices on probe vehicles in mostly urban setting. While many of the concepts and methods developed in this research may also apply to GPS data collected from other modes (such as bike, trains, etc.) and non-transportation applications (such as fitness tracking), a great deal of additional complexity may be introduced for those applications, which is beyond the scope of this research. That said, the work described here may provide useful insight into a wide range of scenarios that may be studied in the future.

There are numerous factors that can complicate the process of compressing GPS vehicle trajectory data while making sure the compressed data is still useful for transportation applications. The location accuracy of GPS data is one such factor, along with communications reliability, application design, and performance, among others. The work described here does not deal with these issues, which should be investigated in future research.

1.5. DISSERTATION ORGANIZATION

The work described in this dissertation has four major components: 1) The development and evaluation of a 'batch' algorithm to resample previously collected GPS-based vehicle trajectory data. This batch algorithm is called self-adaptive sampling (SAS) in this study. 2) The development and evaluation of an online algorithm to resample vehicle trajectory data as they are being collected. This online algorithm is called self-adaptive online trajectory sampling (SAOTS). 3) Comparison of SAS and SAOTS with three existing trajectory compression algorithms. And 4) The development of a semi-supervised technique to identify VFS of the probe vehicle. Figure 1.3 illustrates the relationship between different components of this study and the connection between SAS and SAOTS.

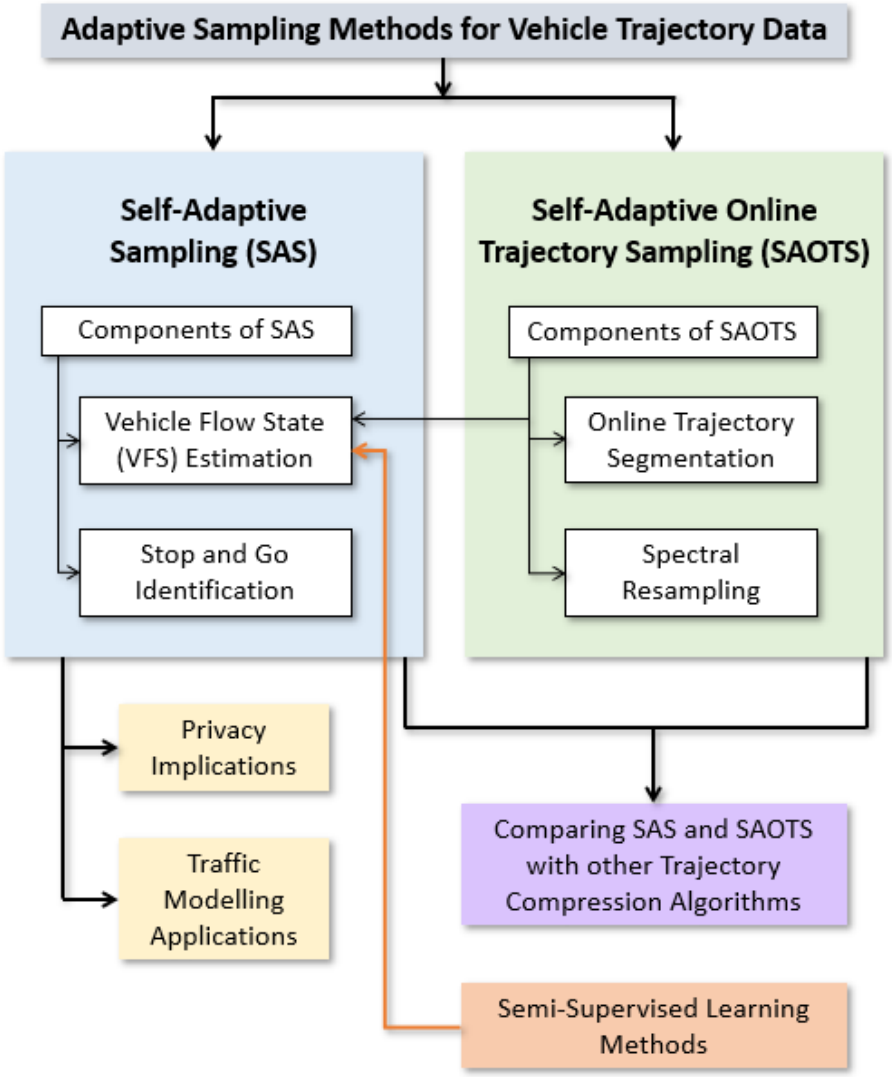


Figure 1.3. Schematics of different components of the dissertation

The remainder of this dissertation is structured as follows. Chapter 2 covers relevant pieces of literature on trajectory compression algorithms, short-term estimation and prediction of traffic flow state, traffic system performance measurement using mobile data, and privacy Issues in traffic data collection. Chapter 3 describes the SAS strategy to resample a GPS based trajectory data. This chapter introduces the concept of vehicle flow state (VFS) and an HMM-based classifier to

identify the VFS of an individual vehicle. Moreover, to better understand the nature of the traffic experienced by the probe vehicle, an SVM based model to identify ‘stop and go’ segments in the trajectory is presented in this Chapter. Chapter 4 presents the SAOTS algorithm. Different components of the SAOTS algorithm which include online segmentation of a trajectory and sampling based on spectral-domain properties are discussed in detail in this chapter. In Chapter5, SAS and SAOTS are compared with three other existing trajectory compression algorithms using two datasets. A semi-supervised method for VFS estimation is investigated in Chapter 6. Chapter 7 concludes the dissertation with a short discussion on the contributions of this study and the scopes of future research.

Chapter 2. LITERATURE REVIEW

Recent research development in mobile sensing data focuses on several major areas such as predicting short term traffic conditions, modeling the system performance measures (such as travel time, delay, queue length, etc.), and assessing privacy issues related to the collection and use of mobile sensing data. The scope of this research includes all of these areas as we are interested in developing state-dependent sampling method for mobile sensing data by estimating traffic state and evaluating the effectiveness of the reduced data in traffic modeling applications and privacy protection. The following sections provide a literature review for each of these research areas.

2.1. TRAJECTORY COMPRESSION METHODS FOR GPS-BASED VEHICLE TRAJECTORY DATA

In recent years, GPS technology has become increasingly available and more accurate. However, collecting vehicle trajectory data from individual GPS devices still costs a considerable amount of battery power and imposes overhead for communication, computing, and data storage. Furthermore, many real-world applications do not really need location data sampled at high frequency (e.g., every second). To address this issue, trajectory compression (or resampling) strategies have been proposed, which aim to reduce the size of a trajectory without compromising much precision in its new data representation (Lee and Krumm, 2011). Existing trajectory compression algorithms can be categorized using two different criteria. The first criterion is whether an algorithm can run online (or near real-time) or not. The second criterion is whether traffic/transportation knowledge is used in the trajectory compression algorithms. Based on the first criterion, trajectory compression algorithms can be categorized as “batch” algorithms or

"online" algorithms. Using the second criterion, trajectory compression algorithms can be categorized as "trajectory geometry-based" algorithms or "knowledge-based" algorithms.

Batch algorithms reduce the size of a trajectory after the original trajectory is fully collected at a high sampling frequency (e.g., one data point every second). Among the batch algorithms, perhaps the most common is the Top-down (Douglas and Peucker, 1973; McMaster, 1986) and the Bottom-up algorithms (Keogh et al., 2001). Top-down algorithms start with considering a non-segmented trajectory as one major segment. Next, it finds the best location for the partition of the trajectory into two segments. For each of the newly formed segments, the process of division into two new segments is repeated in an identical manner until some of the pre-defined stopping criteria (e.g., the maximum number of segments, and/or the approximation error (a user-specified threshold) is satisfied. Bottom-up algorithms are a natural complement to the Top-down algorithms, which begin by dividing the original trajectory into a large number of very small segments with equal lengths. In the next step, based on the comparison of each pair of consecutive segments, the pairs that cause the smallest increase in the error are being identified, and consequently merged in one new, bigger segment. The algorithm repeats these steps until some of the defined stopping criteria (similar to those of the top-down algorithms) is satisfied.

Online algorithms compress a trajectory almost instantly as the vehicle travels and the data are collected. The most popular online algorithm is the Sliding Window algorithm (Shatkay, 1995; Park and Lee, 1999). The algorithm works by anchoring the left point of a potential segment as the first data point of a trajectory, then attempting to approximate the data to the right with increasing longer segments. At any point i, an error is defined. If the error at point i is larger than a user-specified threshold, the subsequence from the anchor to the $(i-1)$th point is transformed into a segment. Only the first and last data points of the segment are retained in the reduced

trajectory. The anchor is then moved to location i, and the process repeats until the entire trajectory is transformed into a piecewise linear approximation.

In general, it is seen that batch algorithms tend to be superior in terms of trajectory reconstruction from the compressed data, while online algorithms can ensure that the collected/reduced trajectories can be accessed near real-time. Keogh et al. (2001) proposed an algorithm to combine the benefits of the two types of methods, called the SWAB (Sliding Window and Bottom-up) algorithm. Its name indicates that it is a direct result of the combination of the Bottom-up approach and the Sliding Window method to retain both trajectory reconstruction quality and online nature.

Although most existing trajectory compression algorithms are computationally efficient, they have a few limitations. One of the prime issues is that they frequently eliminate important points, such as sharp angles. A secondary limitation is that straight lines are still over-represented (Douglas and Peucker, 1973) unless small differences in angles are used as another discarding condition. An algorithm to overcome the first limitation was reported by Jenks (1981) and involved evaluating the perpendicular distance from a line connecting two consecutive data points to an intermediate data point against a user threshold. To tackle the second disadvantage, Jenks (1985) utilized the angular change between three consecutive data points.

A unique feature of the above-reviewed trajectory compression methods (either the batch algorithms or the online algorithms) is they are solely based on the geometry of a trajectory. Out of traditional trajectory geometry-based compression algorithms, many scholars have focused on different perspectives. For example, Birnbaum et al. (2013) proposed a trajectory simplicity algorithm based on sub-trajectories and their similarity. Long et al. (2013) proposed a polynomial-

time algorithm for optimal direction-preserving simplification, which supports a border application range than position-preserving simplification. Nibali and He (2015) proposed an effective compression system for trajectory data called Trajic, which can fill the gap of good compression ratio and small error margin. Muckell et al. (2011) put forward the Spatial QUalIty Simplification Heuristic method. Chen et al. (2012) proposed a Multiresolution Polygonal Approximation algorithm, which compressed trajectories by a joint optimization on both the LSSD and the ISSD criteria. In 2014, Muckell et al. proposed a new algorithm, SQUISH-E, which compresses trajectories with provable guarantees on errors. An extensive review of trajectory geometry-based batch and online algorithms can be found in (Keogh et al. 2001; Meratnia, and Rolf, 2004; Lovrić et al. 2014). An important limitation of geometry-based trajectory compression algorithms is that, since they are purely geometry-based, they often use the perpendicular distance of a data point to a proposed generalized line as the condition to discard or retain the data point. This may result in inadequate data records in some cases as discussed in Zheng (2015).

In the literature, another type of algorithms used map-matching methods to compress vehicle trajectories (Cao and Wolfson, 2005; Kellaris et al., 2013; Liu et al., 2014; Song et al., 2014). These algorithms first used map-matching methods to project GPS data points into a road segment, then transformed the locations of these points into road segment IDs and offsets. An important feature with such algorithms is that they require additional road network information which may not be available in some cases. In addition, using these algorithms, compression is achieved on the map-matched trajectory points in the road segment, instead of the original data points of the trajectory. This could be an issue if a vehicle is not traveling on a well-defined roadway network. For these reasons (primarily for the reason that the underlying road network for the datasets used in this research is unknown), it is difficult to compare the map-matching-based trajectory

compression algorithms with other compression algorithms that do not require road network information.

Table 2.1 summarizes the features and representative studies for the trajectory compression methods in the literature. This study focuses on algorithms that do not rely on road network information. It needs to be emphasized first that such compression methods share two similar major steps: (i) segmentation to divide an entire trajectory into multiple segments; and (ii) sampling to decide which data points to keep or discard. These algorithms differ in the details of how segmentation and sampling are done. The geometry-based algorithms segment a trajectory using the shape (geometry) of the trajectory only, which break the trajectory at data points where certain error criterion is met (e.g., the Euclidean distance from the point to a line approximation of the segment).The distinction of this research from the existing literature is that, in this study, instead of just using the geometric properties of the trajectory, the focus is on the intrinsic features of vehicle trajectories over time in the form of vehicle flow state (VFS) that can better reveal the underlying vehicle moving conditions. Such ‘traffic knowledge’ revealed by vehicle trajectories is used in vehicle trajectory compression to not only reduce the data size but also to make sure the reduced trajectory can be properly used for traffic/transportation applications. Such use of transportation domain knowledge in form of VFS is unprecedented in trajectory compression literature, which distinguishes this research from existing research.

Table 2.1. Summary of the existing trajectory compression methods (algorithms).

TC Algorithm	**Representative Studies**	**User Can Specify**	**Real-Time Application?**	**Traffic Knowledge**	**Road Network Information**
SAS	Siddique and Ban (2018)	Sampling interval	No	Yes	No
Sliding Window	Shatkay (1995), Park and Lee (1999)	Maximum Segment Error	Yes	No	No
Bottom Up	Hunter and McIntosh (1999), Keogh and Pazzani (1999)	Total Segment Error, Number of Segments	No	No	No
SWAB	Keogh et al. (2001)	Total Segment Error, Maximum Segment Error, Number of Segments	Yes	No	No
MMTC-App	Kellaris et al. (2013)	Compression Rate	Yes	No	Yes
PRESS	Song et al. (2014)	Compression Rate	No	No	Yes
SUTS	Liu et al. (2014)	Distance Deviation	No	No	Yes

2.2. Short-term Estimation and Prediction of Traffic Flow State

Numerous tools and algorithms have been applied in the area of traffic flow state estimation and short-term traffic prediction. Kalman Filtering is one of the most commonly used techniques. The Kalman Filtering (KF) method was used for traffic volume prediction by Okutani and Stephanedes (1984). Yang et al. (2004) proposed a Recursive Least Square (RLS) approach for short-term traffic speed prediction by means of KF to adapt to changing patterns quickly, based on the maximum likelihood method and Bayesian rule. Xie et al. (2006) conducted a study to combine the Wavelet decomposition with the KF method for short-term traffic speed prediction. They showed that the wavelet KF model consistently outperformed the KF model in terms of both accuracy and stability and that a higher data decomposition level was more advantageous for non-stationary data prediction. In another study by Xia and Chen (2009), a dynamic short-term corridor travel time prediction model was developed using the KF method, which involves a multi-step-ahead prediction of traffic condition with a seasonal autoregressive integrated moving average model. In conjunction with KF, Wang and Papageorgiou (2008) proposed an approach for real-time adaptive estimation of traffic flow variables based on stochastic macroscopic traffic flow modeling. Recently, Guo and Williams (2010) proposed an autoregressive moving average plus generalized autoregressive conditional heteroscedasticity structure for modeling the station-by-station traffic speed series to forecast the short-term traffic condition level and uncertainty. They employed an online algorithm based on layered KF for processing the heteroscedasticity structure in real-time.

Another method for traffic estimation and forecasting is based on Time Series models. Hamed et al. (1995) developed a Time Series model to predict future traffic volumes on urban arterials using the Box–Jenkins approach. Lee and Fambro (1999) applied the subset Autoregressive

Integrated Moving Average (ARIMA) model for short-term freeway traffic volume prediction. Similarly, Williams and Hoel (2003) modeled univariate traffic condition data streams as seasonal Autoregressive Integrated Moving Average processes. In other another study, Smith et al. (2002) compared the parametric modeling approach of ARIMA and non-parametric regression models for short term traffic flow forecasting. It is concluded that heuristic forecast generation methods did significantly improve the performance of nonparametric regression, but they did not equal the performance of seasonal ARIMA models and traffic condition data is characteristically stochastic rather than chaotic. More recently, Shekhar and Williams (2007) suggested adaptive seasonal models for univariate traffic flow forecasting through the use of three well-known filtering techniques: the Kalman filter, recursive least squares, and least mean squares. Time Series models were also used for short-term traffic speed prediction. Using two algorithms, Expectation-Maximization (EM) and the Cumulative Sum (CUSUM) algorithms, into ARIMA Time Series model, Cetin, and Comert (2006) proposed an adaptive approach for traffic speed prediction. The proposed approaches were tested on a publicly available loop dataset collected by the California PATH with detailed records of all incidents. The study showed that compared to the ARIMA model, the two adaptive techniques provide more accurate results when the data generation process is not stable. Chandra and Al-Deek (2009) developed a vector autoregressive Time Series model to predict traffic speed and volume of a 4-km (2.5-mile) segment of I-4 Orlando, Florida.

In addition to Time Series models, Neural Network (NN) models are another class of methods successfully used for short-term traffic prediction. Smith and Demetsky (1994) introduced the back-propagation neural network model for traffic volume prediction. Park et al. (1998) conducted a study to apply a Radial Basis Function (RBF) neural network in prediction short-term freeway traffic volumes. Yin et al. (2002) developed a fuzzy-neural model (FNM) to predict the traffic

flows in an urban street network. The model consisted of two modules: a gate network (GN) and an expert network (EN). Ishak et al. (2003) optimized short-term traffic prediction performance by using multiple topologies of dynamic neural networks under various parameters and traffic-condition settings. In order to improve the performance of the NN models, many hybrid models were proposed. For instance, Abdulhai et al. (1999) developed a system based on Time Delay Neural Network (TDNN) model synthesized using Genetic Algorithm (GA) for short-term traffic prediction. Alecsandru and Ishak (2004) proposed a hybrid model-based and memory-based methodology to strengthen predictions under both recurrent and non-recurrent conditions. The model-based approach relied on a combination of static and dynamic neural network architectures to achieve optimal prediction performance under various input and traffic condition settings. Vlahogianni et al. (2005) incorporated GA to optimize the learning rule as well as the network structure. The GA approach was composed of three steps: selection, crossover, and mutation, built on the principles of genetics. Many other NN models on short-term traffic prediction were found in the literature (see for instance Jiang and Adeli, 2005; Ishak and Alecsandru, 2004, and Shen and Hadi, 2010).

In summary, the literature review showed that KF, Time Series, and NN models are the most commonly used methods for short-term traffic estimation and prediction; most approaches aimed to construct models to estimate traffic speeds or volumes using fixed location sensor data. In this study, the traffic flow state estimation model uses mobile sensing data, which focuses on identifying the states of small segments in vehicle trajectories and attempts to capture the characteristics of traffic states based on the speed variation of the vehicle. This study proposes a stochastic approach, the Hidden Markov Model (HMM), for estimating traffic flow state. HMMs are commonly used for speech, handwriting, and gesture recognition. Recently, Kwon and Murphy

(2000) used coupled HMMs to model freeway traffic and predict traffic speed. Their study defined two states (congestion and free flow) using the mean speed. Statistical inference algorithms were required to train the model, which was not computationally effective even with a small dataset (10 x 60 observations). The study concluded that the predicted speeds using this method were not very accurate. In this study, an HMM and SVM based model is developed and calibrated using real-world mobile traffic data.

2.3. Traffic System Performance Measurement Using Mobile Data

Queue length is one of the most crucial performance measures for signalized intersections (Balke et al., 2005). Many early studies assumed discrete arrivals and integer cycle lengths, and Markov chain or similar statistical analysis techniques were applied to estimate the mean or distribution of queue lengths (Haight, 1959; Newell, 1960; Darroch, 1964; McNeil, 1968). Data collected from mobile sensors are principally different from those from fixed location sensors: fixed location sensors collect aggregate information such as flow and occupancy from the entire traffic stream, while mobile sensors collect finer location information of a vehicle from a sample of the traffic stream (Sun and Ban, 2013). Therefore, traffic volume or occupancy information, which is the main input to most existing fixed-location-data-based queue estimation methods, is generally not available from mobile data. Recently, investigating the feasibility of using mobile sensors for performance measurement of traffic systems such as signalized intersections, has received much attention. Early research focused on freeway traffic (Lu and Skabardonis, 2007; Herrera et al., 2010; Herrera and Bayen, 2010). The methods were based on shockwave theory or Kalman filtering for continuous flow. Comert and Cetin, (2009); Izadpanah et al., (2009); Cheng et al., (2010) focused on queue length estimation from vehicle trajectories directly. Ban et al. (2009) showed that sample intersection travel times can be used to estimate real-time intersection

delay patterns. Ban et al. (2011) further proposed a method to estimate real-time queue lengths of a signalized intersection using sample travel times. Cheng et al. (2012) constructed real-time queue length by analyzing the detailed vehicle trajectories from mobile sensors. Argote et al. (2011) developed models for evaluating several measures of effectiveness using Connected Vehicle data. Most mobile-data-based queue length estimation methods were mainly developed by assuming uniform arrivals at the intersection, and that acceleration and deceleration of vehicles can be ignored (Cheng et al., 2012; Ban et al., 2011). Hoifeiner et al. (2013) presented a hybrid modeling framework based on traffic flow theory and machine learning for estimating and predicting arterial traffic conditions using streaming GPS probe data. The results indicate that this approach is a significant step forward in estimating traffic states throughout the arterial network using a relatively small amount of real-time data.

Hao et al. (2013) proposed a kinematic equation-based method to estimate the location of a vehicle in the queue based on the vehicle's travel time traversing a signalized intersection without any prior assumptions about vehicle arrivals. The method focuses on the discharging process of the vehicle by investigating its acceleration starting from the queued location to downstream locations passing the intersection. Three cases for through movements and six cases for left-turn movements were discussed to estimate a vehicle's location in a queue. For each case, the authors illustrated the trajectory, analyzed the unique feature of the trajectory, and developed equations and inequalities to calculate the acceleration rate and queue location simultaneously.

2.4. Privacy Issues in Traffic Data Collection and Use

The growth of mobile devices also comes with concerns about location privacy. Questions have been asked regarding the risks if location data leaks to an unwanted third party and how we can avoid the adverse consequences of a location leak. Beresford and Stajano (2003) define location privacy as the ability to prevent other parties from learning one's current or past location. This definition captures the idea that the person whose location is being measured should control who can know it. It also recognizes that past location information is important to protect. Duckham and Kulik (2006) refined the concept of location privacy by defining it as a special type of information privacy which concerns the claim of individuals to determine for themselves when, how, and to what extent their location information is communicated to others.

Progress in computational location privacy depends on the ability to quantify location privacy. There is not yet a standard for this; it is even rare for two different research projects to use the same method of quantification. Since location can be specified as a single coordinate, one way to measure location privacy is by how much an attacker might know about this coordinate. For instance, Hoh and Gruteser (2005) quantify location privacy as the expected error in distance between a person's true location and an attacker's estimate of that location. The consequences of a location leak range from the uncomfortable creepiness of being watched, to unwanted revelations of a person's activities, to actual physical harm. Hoh et al. (2006) used a database of week-long GPS traces from 239 drivers in the Detroit, MI area. Examining a subset of 65 drivers, their home-finding algorithm was able to find plausible home locations of about 85%, although the authors did not know the actual locations of the drivers' homes. A similar attack was simulated against two weeks of GPS data from 172 drivers in Krumm (2007). The drivers' home latitude and longitude were first determined with simple algorithms, giving a median error of about 61 meters

compared to the drivers' actual home addresses. Using a reverse white pages lookup, these coordinates correctly identified 13% of the drivers' home addresses and 5% of their names. Gruteser and Hoh (2005) worked with GPS data that had been completely anonymized in that not even a consistent pseudonym was supplied with the time-stamped latitude and longitude coordinates. They used a standard technique from multi-target tracking proposed by Blackman (1986) to accurately cluster the measured GPS points from three people. This demonstrates that even mixing together coordinates from different people is not enough to prevent an attacker from reassembling them into coherent, individual tracks.

One of the common privacy techniques is anonymization (Sweeney, 2002; Rass et al., 2008; Stenneth and Yu, 2010), which guarantees the anonymity of an object using one pseudonym, i.e., a randomly generated ID, throughout the dataset or dynamic pseudonyms (periodically updating the current ID with randomly generated pseudonym), and pure anonymity (removing the IDs for all the data points completely). However, pseudonyms are subject to privacy breaches with hidden information and domain knowledge. For example, Machanavajjhala et al. (2006) pointed out when the sensitive attributes in a dataset are of little diversity, or when the adversaries have access to external data sources, pseudonyms can be easily breached. More sophisticated approaches have been developed to enhance anonymity, mainly by perturbing data accuracy or restricting the release of certain location information data points; called obfuscation (Ardagna et al., 2007; Agrawal and Srikant, 2000; Kargupta et al., 2003; Gruteser and Grunwald, 2003; Gedik and Liu, 200)5. Location data perturbed by such methods, however, can rarely be used for applications that require fine-grained location traces. Similarly, reducing sampling frequency (Tang et al., 2006) or using dummies (Kido et al., 2005; Lu et al., 2008; Nergiz et al., 2009) may also severely degrade location information.

Sun et al. (2013) developed privacy mechanisms that would simultaneously satisfy privacy protection and data needs for fine-grained urban traffic modeling applications using mobile sensors. To accomplish this, the authors proposed a virtual trip lines (VTLs) zone-based system and related filtering approaches. Traffic-knowledge-based adversary models were developed to evaluate the performance of the VTL zone method, especially in terms of the filtering algorithms. The results show that this “Privacy-by-design” approach ensures an acceptable level of privacy, and the released datasets from the privacy-enhancing system can also be applied to urban traffic modeling with satisfactory results.

Chapter 3. THE SELF-ADAPTIVE SAMPLING (SAS) METHOD

3.1. OVERVIEW OF SAS METHOD

This section provides an overview of the proposed self-adaptive sampling (SAS) method. On the sensor side, trajectories of individual vehicles are originally recorded at a relatively high sampling rate (e.g., every second). The objective of developing SAS method is to decide which of the original data points will be retained and sent to say a central server, based on the estimated VFS. The VFS in this study can be considered as the state of either a trajectory data point or a trajectory segment (that contains multiple consecutive data points), where the vehicle undergoes a specific type of motion. The VFS of a segment is determined by the VFS of its constituent data points to indicate that data points in the segment represent a distinct motion from those of the neighboring segments. For example, a segment in a trajectory where the vehicle cruises in free-flowing condition will have the VFS to indicate "free-flowing" until it starts to decelerate/accelerate consistently. Furthermore, VFS here closely resembles the kinematic motion of an individual vehicle, not the overall traffic condition of the roadway location, which can be captured by the speed (also acceleration/deceleration) of the vehicle. There are two reasons for using the VFS of an individual vehicle instead of the overall traffic state. First, the state of a vehicle better reflects what that specific vehicle is experiencing (which might be different from what the other vehicles are experiencing), and thus is more appropriate to use to design the sampling scheme for that vehicle. Secondly, since the proposed method is implemented on the vehicle (sensor) side, information/data about the other vehicles are often not available, making it extremely difficult for the method to focus on the state of the overall traffic. This is especially true when the data are collected in a crowdsourcing manner from an ad-hoc, dynamic user base (e.g., most mobile app-based applications such as google map). Notice here that by focusing on individual vehicle

trajectories, the proposed method may not accurately reflect the actual state of the entire traffic flow. This is an important feature of the proposed SAS method. For example, even the overall traffic is light and free-flowing, a specific vehicle might drive slow or even stop for certain reasons. In this case, the slowdown and stop states of the vehicle should be collected (e.g., by the proposed method) for later analysis. On the other hand, if the proposed method had been based on the overall traffic state, such information would be lost as the sampling rate would be small (since the traffic is free-flowing). It may also be true that VFS may be associated with external features such as lane changing and driving behavior. However, such data are not directly available from vehicle trajectory data and are thus not used in the SAS method in this study. Figure 3.1 illustrates the model architecture of SAS.

In this study, four types of VFS are considered:

- Free Flow: in this state, the vehicle moves in a relatively constant speed.
- Stopped: when the vehicle is stationary.
- Acceleration: when the vehicle is speeding up.
- Deceleration: when the vehicle is slowing down.

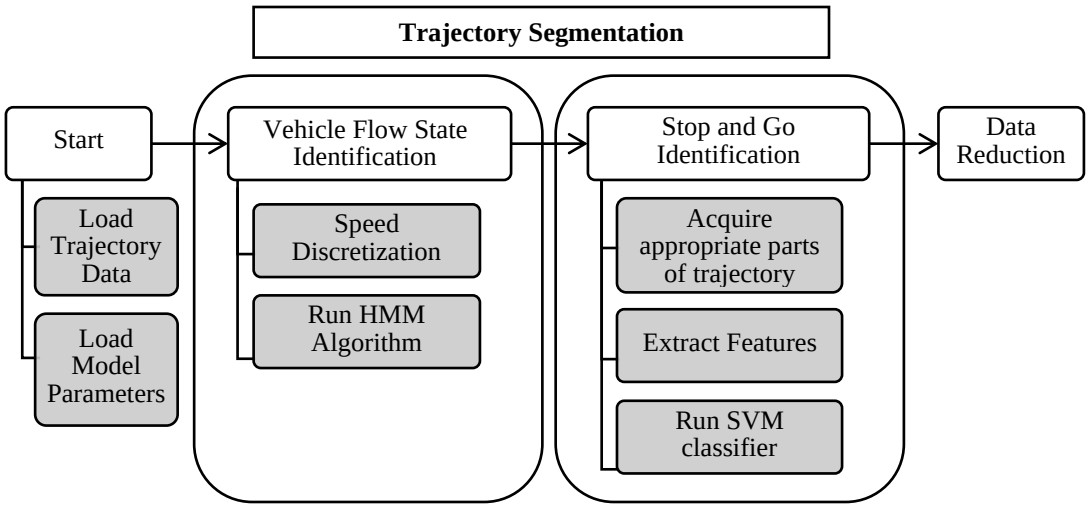


Figure 3.1. SAS model architecture

Notice that these four states are "hidden" since they are not easily observable based on the vehicle trajectory data (e.g., instantaneous speeds and accelerations/decelerations). The SAS first groups each trajectory into segments (i.e., smaller pieces of the trajectory), called "trajectory segmentation" in this research. The segmentation is done in two steps, as shown in figure 3.1. In the first step, the VFS for each data point of the trajectory is classified. A trained hidden Markov model (HMM) is used for this purpose. The HMM classifier labels each individual points of a trajectory in one of four states listed above: Free Flow, Stopped, Deceleration, and Acceleration. After this first classification step, data points with the same VFS are grouped together. At this point, the entire trajectory is broken into small segments with known VFS. The next step is to identify potential stop and go segments in the trajectory. To achieve this, the parts of the trajectory that lies between the free flow segments are investigated. A Support Vector Machine (SVM) classifier is used to identify stop-and-go segments in the trajectory. Once the segmentation is done, different sampling strategies can be applied to the segmented trajectory to reduce the data size. The HMM model and SVM model developed in this study are trained in advance using historical

data. The following sections (section 3.2 and section 3.3) provide more details of the HMM and SVM models respectively. Note that HMM and SVM are used in this study since they help achieve reasonable performances of the SAS method. Other learning methods may also be used and may achieve similar or even better performances, which however will not change the main contribution or results (i.e., the SAS related concept and methods) of this research.

3.2. Vehicle Flow State identification of Individual Trajectory Points Using HMM

An HMM is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. An HMM can be considered the simplest dynamic Bayesian network. The mathematics behind the HMM was first developed in Baum et al. (1966). The main reason to choose the HMM method is its simple, 'memoryless' property, which helps to apply the model in applications where real-time feedback can be crucial.

To apply the HMM in this study, let us consider a vehicle that may be described at any time as being in one of the four (hidden) states: Stopped, Free Flow, Acceleration, and Deceleration. At regularly spaced discrete times, the vehicle undergoes a change of states according to a set of probabilities associated with the state. We cannot see the process of the state transitions, but we can see a set of observable outputs that are related somehow to the hidden states. In the case, the observations are the trajectories (here the focus in on the instantaneous speeds) of vehicles. There are two main reasons for using the instantaneous speed of the vehicle as the only observations in HMM. First, it can decently reflect the kinematic motion of the vehicle (which is closely related to the hidden state) without using any additional variables. Secondly, the instantaneous speed of the vehicle can be easily accessible in real-time.

To apply the HMM, the continuous speed is discretized into three separate categories as shown in table 3.1. Since in the traditional HMM the observations are discrete categories, different ways to discretize the instantaneous speeds are extensively investigated. The number of discrete categories and the threshold values for discretizing instantaneous speeds are selected after carefully analyzing speed changing patterns from some probe vehicle trajectories. The analysis showed that using the values given in Table 3.1 provides a balanced model with a satisfactory result and comparatively less complicated parameters. A vehicle can roughly be considered as 'Stopped' when its instantaneous speed is less than $3\ mph$; in an urban roadway, where the speed limit is usually $30\ mph$, a vehicle with its speed larger than $20\ mph$ may be considered as traveling in "high speed," while a vehicle with its speed between 3 *mph* and 20 *mph* may be considered as traveling in "low speed". It should be pointed out there here that, these threshold values may change depending on the vehicle, the location of the study site, and other characteristics of the traffic system, which could be calibrated using real-world data. For example, in the case of freeways, where the speed limit is higher than urban roadways, the upper threshold reflecting "high speed" could be larger than $20\ mph$. To calibrate those threshold values for a specific location/vehicle, vehicle trajectories need to be collected and analyzed as described above.

Table 3.1. Speed discretization rule.

Category	**Discretization Rule**
1 (“stopped”)	Speed ≤ 3 mph
2 (“low speed”)	3 mph < Speed ≤ 20 mph
3 (“high speed”)	Speed > 20 mph

An HMM consists of three parameters: the prior vector (denoted as π), the state transition probability matrix (denoted as A), and the observation matrix (denoted as B). In real applications, these parameters need to be trained/calibrated using data. In this study, the model parameters were trained (calibrated) using the Next-Generation SIMulation (NGSIM) dataset (Cambridge Systematics, 2007). For this, the VFS of each data point of 142 vehicle trajectories in the NGSIM dataset were first manually identified. The HMM parameters are then calibrated based on the identified VFS. In the following, details of each parameter are presented, together with the calibration results of the parameters used for the HMM model. Further details about the NGSIM dataset are given in section 3.5.

Prior vector (π) contains the probability of the (hidden) model being in a particular hidden state at time $t = 1$. It is the probability of the initial distribution. Here it is assumed that the vehicle trajectory always begins from a stopped state. If the trajectory starts from a different state, the prior vector can be changed accordingly.

State transition Probability matrix(A) is the probability of a hidden state given the previous hidden state. The matrix $A = \{a_{ij}\}$ is $N \times N$ with:

$$a_{ij} = P\left(state\ S_j\ at\ time\ (t+1)\ \middle|\ state\ S_i\ at\ time\ t\right). \quad (3.1)$$

Here *N=4* is the number of hidden states in the process. As we already know the actual hidden state sequence (in form of manually identified VFS) in the vehicle trajectory, the state transition probability from state i to state j can be calculated by the following formula (Durbin et al., 1998):

$$a_{ij} = \frac{Number\ of\ times\ the\ transition\ from\ state\ i\ to\ state\ j\ is\ taken}{Number\ of\ times\ a\ transition\ from\ state\ i\ to\ any\ state\ is\ taken} \quad (3.2)$$

Table 3.2 shows the State Transition matrix for the trained HMM, using the NGSIM data. From the table, it is evident that the transition probability from state 'Free Flow' to state 'Stopped' and vice versa is 0 ($a_{23} = a_{32} = 0$). That is, if a vehicle is in Free Flow state, it cannot directly change to the Stopped state in the next time instant, or vice versa. For such kinds of transitions to happen, the vehicle must undergo an appropriate Deceleration or Acceleration state first. If a vehicle is cruising in the free-flow state in a given time instant, there is a 96.3% probability that the state will remain unchanged in the next time instant ($a_{22} = 0.963$), a 2.2% probability that the vehicle will decelerate in the next time instant, and 1.5% probability that the vehicle will accelerate. Similarly, if the vehicle is stationary in a given second, there is a 95.9% probability that the state will remain unchanged ($a_{11} = 0.959$) and a 3.7% probability that the vehicle will accelerate in the next time instant. The state transition probabilities are also shown graphically in figure 3.2. Note that NGSIM data are for urban arterials, and the obtained parameters here may be applied to urban streets as well. For other types of roadways, similar data collection and calibration process can be applied. Note also that there may be a long-term trend in the state transition probabilities due to changes in traffic and roadway conditions. In this case, the transition probabilities may be recalibrated periodically (e.g., every week or every month depending on the temporal characteristics of the change) to capture such trends.

Table 3.2. State transition matrix.

		Traffic Flow State of Next Data Point			
		Stopped	**Free Flow**	**Deceleration**	**Acceleration**
Traffic Flow State of Current Data Point	**Stopped**	0.959	0.000	0.005	0.037
	Free Flow	0.000	0.963	0.022	0.015
	Deceleration	0.070	0.012	0.919	0.000
	Acceleration	0.021	0.064	0.000	0.914

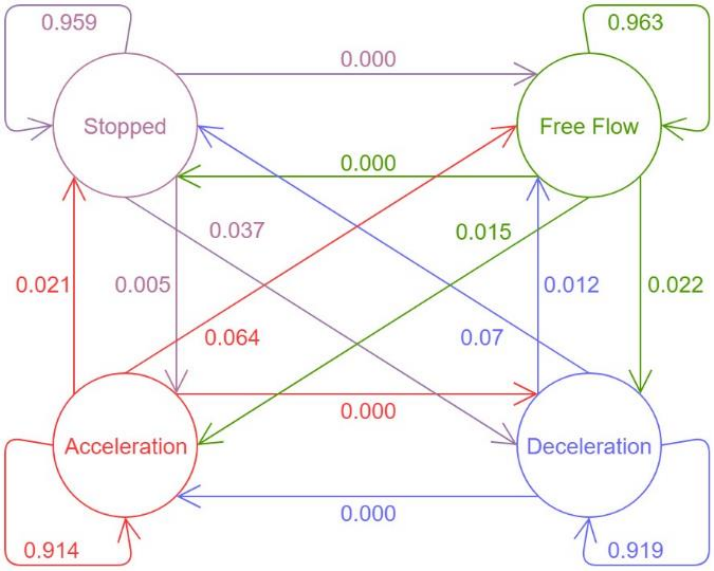


Figure 3.2. HMM for the traffic flow states with state transition probabilities

The observation matrix (B) contains the probability of observing a particular observable state given that the hidden model is in a particular hidden state. $B = \{b_j(k)\} is\ an\ N \times M$ matrix with (M denotes the number of discretized speed categories)

$$b_j(k) = P\ (observation\ k\ at\ time\ t\ |state\ S_j\ at\ time\ t). \quad (3.3)$$

Here N = 4, and M = 3 for our study. Similar to the state transition matrix, the observation probability for an observation k from state j can be calculated by the following formula:

$$a_{ij} = \frac{Number\ of\ times\ the\ observation\ k\ is\ emitted\ from\ state\ j}{Total\ number\ of\ times\ an\ observation\ is\ emitted\ from\ state\ j} \quad (3.4)$$

Table 3.3 illustrates the observation matrix of the trained HMM. This matrix contains the probability of having a particular hidden state for a specific observation. For example, for the "Stopped" state, 95.9% of the time the observation should be 1 ("stopped"), and the rest of the time the observation should be 2 ("low speed"). Similarly, for the "Free Flow" state, 99.6% of the time the observation should be 3 ("high speed"). For the "Acceleration" and "Deceleration" states, observation probability is 100% for observation 2 ("low speed"). It means the vehicle speed is always going to be between 3 mph and 20 mph if the underlying vehicle flow state is either acceleration or deceleration.

Table 3.3. Observation matrix.

		Observation Category		
		1 ('stopped')	**2 ("low speed")**	**3 ("high speed")**
State	**Stopped**	0.959	0.041	0.000
	Free Flow	0.000	0.004	0.996
	Deceleration	0.000	1.000	0.000
	Acceleration	0.000	1.000	0.000

Using these parameters, the sequence of VFS (hidden state) in any vehicle trajectory can be estimated using the Viterbi algorithm (Rabiner, 1989). The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states –called the Viterbi

path – that results in a sequence of observed events, especially in the context of Markov information sources and HMMs. The terms Viterbi path and Viterbi algorithm are also applied to related dynamic programming algorithms that discover the single most likely explanation for an observation. Details of the algorithm are omitted here. After the sequence of VFS is estimated, the consecutive data points with the same VFS are grouped together as a segment. Detailed results of applying the HMM for segmentation are presented in section 3.5.

In this research, HMM was chosen as it can classify each vehicle trajectory point in real-time using a comparatively simpler model. The study has tried using a simple speed/acceleration-based heuristic to determine where the vehicle is traveling at free-flow speed or stopped. However, they were not able to classify vehicle flow state (VFS) as defined in the paper with sufficient accuracy. This is because the VFS in this study is defined as the underlying state of probe vehicle which is different from just observing the instantaneous of speed/acceleration. For example, while in the free-flowing state, the vehicle may have instantaneous acceleration or deceleration, but the VFS should still remain the same as it describes the general trend of the vehicle's kinematic motion. The trained HMM model, on the other hand, was able to classify the VFS with higher accuracy than a simple heuristic and is thus applied in the research.

3.3. Identification of 'Stop and Go' Segments Using SVM:

After the segmentation of a vehicle's trajectory, the model further identifies the 'stop and go' segments in the trajectory. In the literature, the 'stop and go' condition is characterized by periodically enforced stops (or severe slowdowns), caused by heavy traffic congestion or traffic signals (Yeo and Skabardonis, 2009). This phenomenon is also known as traffic oscillation. During 'stop and go', the vehicle decelerates, followed by acceleration, repeatedly. Between the slowing

down and speeding up, there may or may not be a completely stopped segment. For example, on highways due to heavy traffic or accidents, a vehicle may need to slow down and then speed up again without stopping. Other types of ‘stop and go’ can be found in congested urban roadways or near intersections where vehicles are compelled to stop completely. In order to identify ‘stop and go’, the main challenge is to distinguish between traffic oscillation and stops at intersections when the traffic signal is red. This problem is dealt with by developing an SVM classifier. SVM is a widely used supervised learning technique, which can be applied for binary and multi-class classification (Vapnik, 1995). Comprehensive studies of SVM can be found in Burges (1998) and Cristianini and Shawe-Taylor (2000).

The key for designing a successful SVM classifier is to find appropriate features from the data. Since this research aims to further label some of the obtained segments from the previous step into ‘stop and go’ segments, the data here are the segments of a trajectory. The goal here is to classify whether a given segment is a ‘stop and go’ segment or not, i.e., a binary classification problem. It was found that the following features related to the variation of speeds and the frequency of stops with respect to time and space are the most effective in terms of identifying ‘stop and go’ segments in a trajectory, which are salient characteristics of stop and go.

1. Number of stops in a segment divided by the time duration of the segment
2. Number of stops divided by the total distance traveled in that segment
3. ‘Peak’ speed between stops divided by the maximum speed of the segment
4. Maximum stop time of a single stop inside the segment divided by the time duration of the Segment
5. Total stop time divided by the time duration of the segment

Notice that in some cases (e.g. feature 1, 4, 5), the actual features of the segment are divided by the time duration of the segment. The division helps when the duration of the potential stop and go segment vary substantially (from several seconds up to several minutes). Extracting the features this way makes the model more robust as it can handle large variations in the segment duration.

The development of an SVM classifier contains a training step and a testing step. The training step is to calculate the optimal model parameters using historical data, while the testing step is to apply the calibrated SVM to determine the class (i.e., 'stop and go' or not) of a given sample (here a segment). This study uses a training dataset, denoted as $(x_1, t_1), \dots, (x_i, t_i), \dots, (x_N, t_N)$, where $N = 281$ is the total number of training samples which were curated from Field dataset (See section 6.1). Here $x_i \in R^5$ is the input vector of extracted features for stop/go identification of the ith sample, with $t_i \in \{1, -1\}$ as the corresponding label, with 1 for 'stop and go' segments -1 for segments that are not. Let us denote $\varphi(x)$ a fixed feature space transformation also called the "kernel" (Cristianini and Shawe-Taylor, 2000), which transforms a vector $x = (x_i)_{i=1,\dots,N} \in R^5$ in the original feature space to the transformed feature space. The reason for this transformation is to deal with classification problems that are not linearly separable (Lauer and Bloch, 2008). In this case, data need to be mapped into a higher dimensional feature space in which the transformed data are linearly separable in the feature space. Then a separating hyperplane (in the 2-D space, this will be a line) $w^T\varphi(x) + b = 0$ can be defined to separate samples of 'stop and go' $(t = 1)$ and not $(t = -1)$. Here w and b are parameters that define the separating hyperplane (e.g., the slope and intercept if the hyperplane is a line in the 2-D space).

The training step aims to use a set of samples to find the optimal values of *w* and *b*, denoted as (w^*, b^*). This is usually done by maximizing the margin of the two classes. For a separable

case, a margin is defined as the minimum distance between the points of the two classes, which is measured perpendicularly to the separating hyperplane. In our case, the extracted features are not strictly separable. In other words, it is impossible to correctly separate/classify all the samples using a separating hyperplane. To deal with this, the above problem can be extended by introducing the concept of soft margin that accepts some misclassification of the training samples. To accomplish this, a control variable *C* is incorporated to penalize the misclassified data points (i.e., stop/go segments misclassified as non 'stop and go' segments and vice versa). Parameter *C* is used to control the trade-off between the penalization of the errors and the maximization of the margin, which is determined using cross-validation techniques. For our particular problem, using $C = 10$ yields the best result with around 12% cross-validation error. It was also found that a polynomial kernel performs the best. The study used cross-validation to find the order of the polynomial kernel. It turns out that using the 6^{th}-order polynomial kernel maximizes the accuracy of the classifier on training and testing data, which is evident in figure 3.3. In summary, the 'stop and go' segment identification error is about 12% if the 6^{th}-order polynomial kernel and *C*=10 are used.

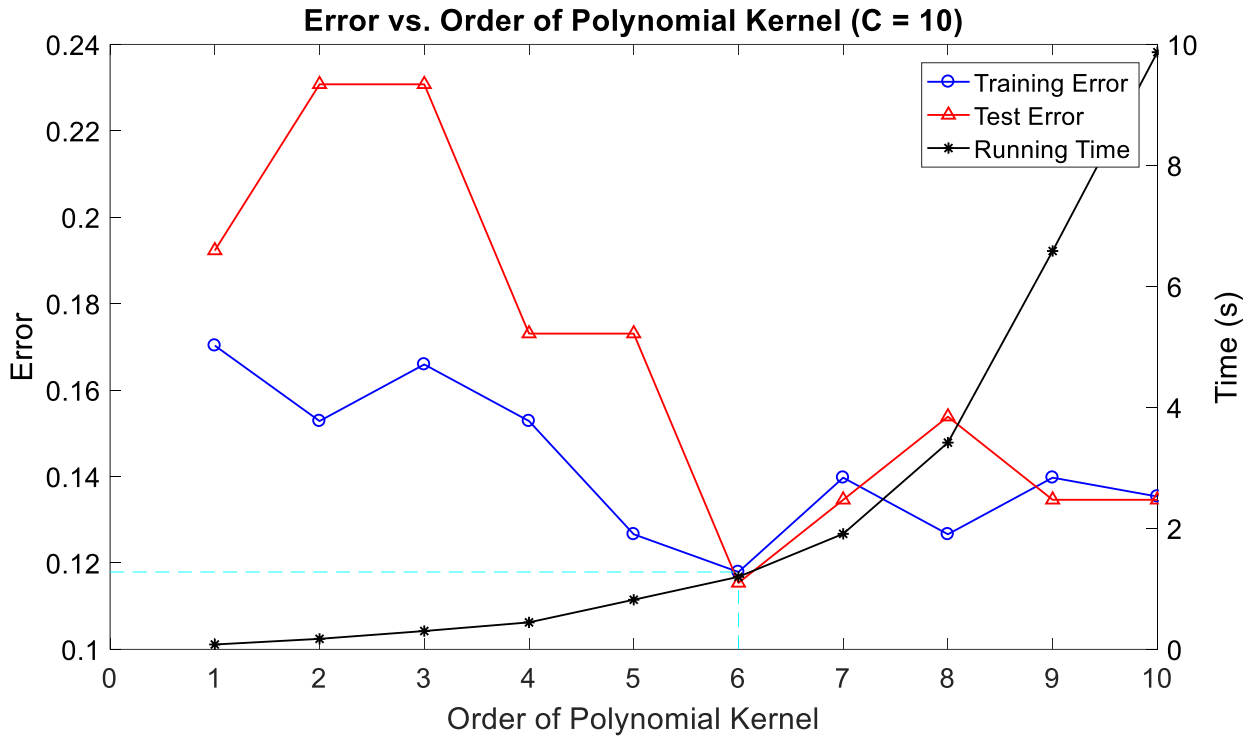


Figure 3.3. Optimizing SVM Parameters.

After training the classifier, the testing step is to determine the proper class ('stop and go' or not in this research) for a given testing data sample x_t (trajectory segment in this research). Figure 3.4 illustrates an example of 'stop and go' identification in a sample trajectory. Segments that are classified as 'stop and go' are shaded. It is noticeable that these trajectory segments undergo a rapid succession of deceleration, stop and acceleration – which is caused by 'stop and go'.

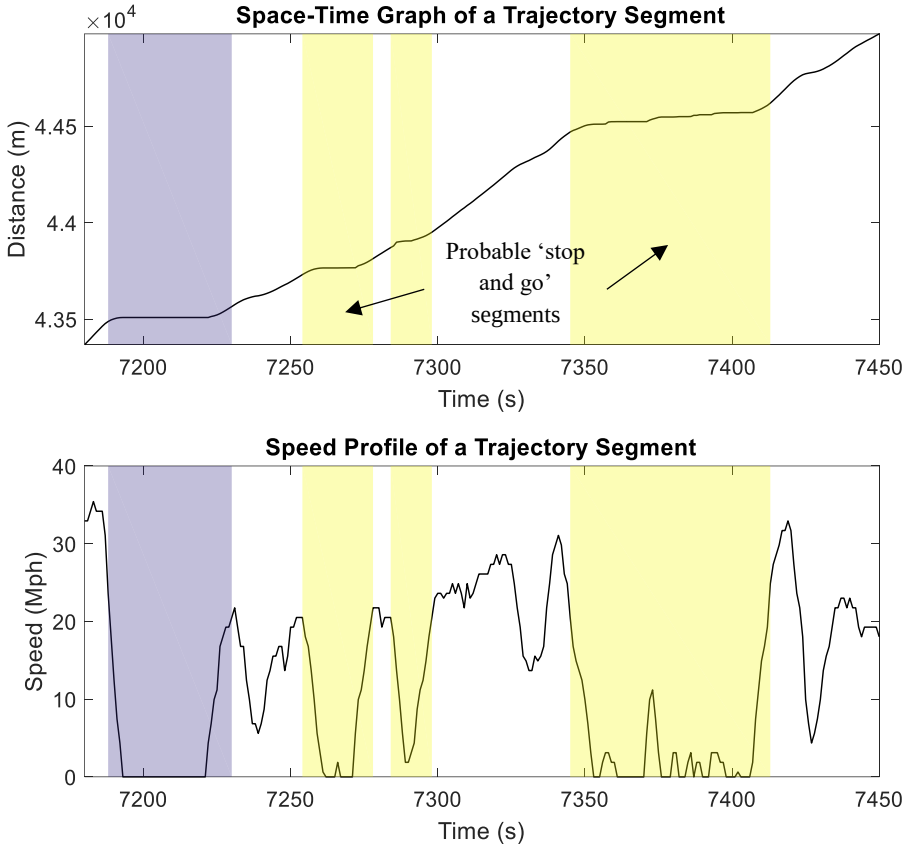


Figure 3.4. Results of 'stop and go' identification.

3.4. THE SAS STRATEGY

After grouping the trajectory of a vehicle into segments labeled with specific VFS, a state-dependent sampling strategy is applied to the trajectory. This means for different types of trajectory segments, different sampling rates are used. Table 3.4 illustrates the sampling strategies. It is important to note here that the SAS strategy presented in this study follows a rudimentary, rule-based approach. The idea is to illustrate the effectiveness of the proposed SAS method in data reduction and traffic modeling applications. A more sophisticated strategy based on the proposed VFS estimation method could be a part of future research. It turns out that the state-dependent SAS strategy proposed here can reduce 67% -77% data records of the original vehicle trajectory, while

keeping the most critical data points, based on the datasets used in this paper as shown in the next section.

Table 3.4. SAS strategy.

Label of the segment	**Sampling rate**
Free Flow	0.1 Hz (i.e., keep one data point for every 10 data points if the raw data are sampled every second)
Stopped	Varying rate (Keep the first two and last two points, and reject all data points in between
Acceleration	1 Hz (i.e., keep all data points)
Deceleration	1 Hz (i.e., keep all data points)
Stop and go	1 Hz (i.e., keep all data points)

3.5. VFS ESTIMATION AND DATA REDUCTION

The performance of VFS estimation is tested using two different datasets. The first one is the NGSIM vehicle trajectory data, collected on Peachtree Street in Atlanta, Georgia from 4:00 pm to 4:15 pm on November 8, 2006. The original data sampling rate is 10Hz. The vehicle trajectories in the NGSIM dataset are shown in Figure 3.5. The second dataset consists of vehicle trajectories collected using GPS loggers from the Capital Region of New York. The sampling rate of these GPS traces is 1Hz. Figure 3.6 shows a sample trajectory of the Field dataset.

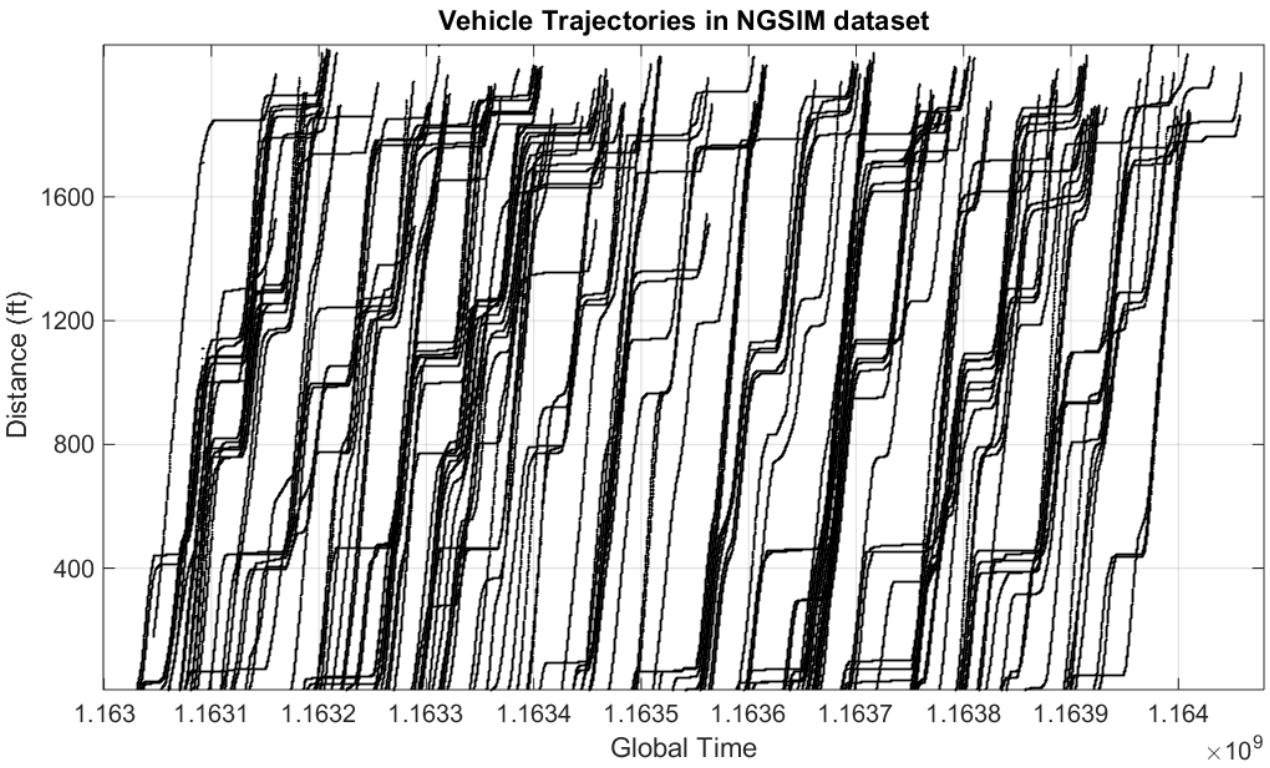


Figure 3.5. Space-Time graph of vehicle trajectories in NGSIM dataset.

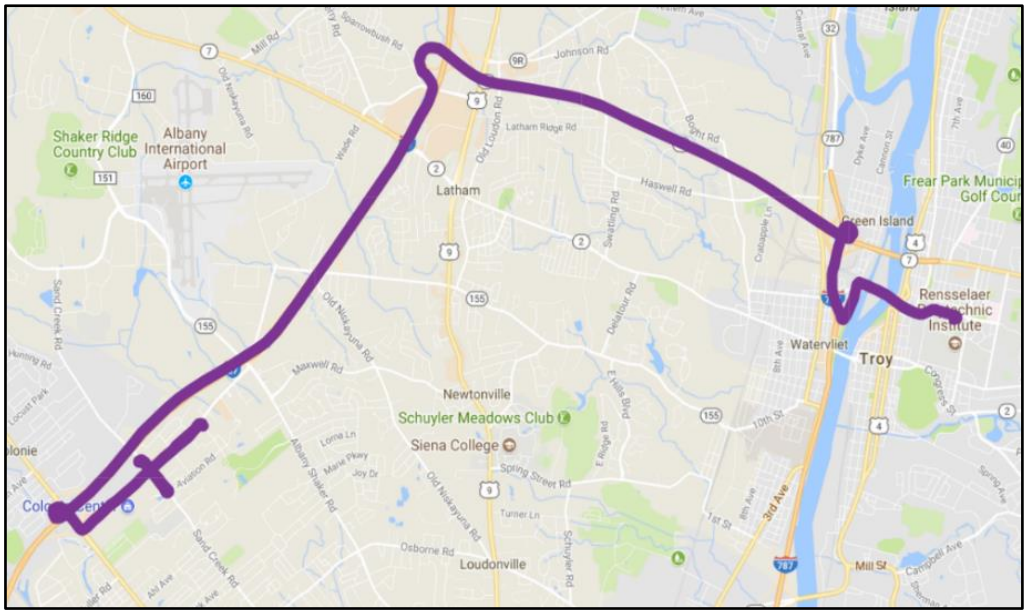


Figure 3.6. A sample trajectory in Field dataset.

To measure the accuracy of VFS estimation model, first, the actual VFS of all data records are manually classified using human judgment. Using the identified VFS, the HMM parameters are calibrated for the two datasets separately. This resulted in two sets of HMM parameters, which are then used for predicting the VFS from the two datasets respectively. When calibrating the HMM parameters, a 10-fold cross-validation technique was used to produce robust results. Note that the classifier first divides a trajectory into four distinct states ('Stopped', 'Free Flow', 'Acceleration', 'Deceleration') and then identifies 'stop and go' from these four states. Since the performances of 'stop and go' detection (i.e., the SVM method) has already been discussed in detail in Section 3.3, this subsection only focuses on the performances of the VFS estimation method for the other four states. Table 3.5 summarizes the classifier performance, together with the data reduction rates.

Table 3.5. Summary of VFS estimation results.

Training Dataset	**NGSIM**		**Field**	
Number of Data Points	189964		16153	
Testing Dataset	**NGSIM**	**Field**	**NGSIM**	**Field**
Accuracy	87.92%	85.46%	83.02%	88.86%
Data Reduction Rate	67.74%	76.34%	67.39%	74.89%

It is seen that in general the accuracy, which is defined as the average match between the model estimates and human judgment, ranges between 83%-89% for different cases. Accuracy is higher when the model parameters are calibrated and tested using the same dataset, although the performance difference is not very significant (about 5-10%). For example, in the case of HMM parameters calibrated using the field dataset, the average accuracy after 10-fold cross-validation is

around 89% while the accuracy is around 83% when tested on NGSIM dataset. Data reduction rates show similar performances (about 75% if trained/tested using the same dataset, and about 67% if trained/tested using different datasets). The results in Table 3.5 clearly show some trade-offs between available resources and model accuracy: if more resources are available, it is certainly desirable to train the models for different types of roadways/vehicles to achieve better performance; however if resources are not available, training the models using datasets collected from one site may still be used for other (similar) sites with reasonable (and degraded) performances.

A low matching percentage between model prediction and human judgment is not desirable from the state estimation point of view, which however should not be a major concern for applying the SAS method to mobile data collection for two reasons. First, human judgment is just a benchmark to compare the proposed models, and not the real "ground truth" especially when acceleration/deceleration states are considered, as will be shown in more detail later. Secondly, as it is demonstrated next, such mismatch will result in less data reduction and more data points retained, which is more beneficial for traffic modeling applications, compared with human judgment.

We further see that, in general, the data reduction rate is higher for the field dataset. The reason for this is probably the road network pattern in this dataset, which mainly consists of highways. In a highway setting, the vehicle is more likely to be on a free flow state compared to frequent acceleration and deceleration in an urban roadway. This often results in higher data reduction based on the SAS strategy. The NGSIM data, on the other hand, were collected on an urban roadway with frequent stops, which result in comparatively lower data reduction.

Figure 3.7 compares the model prediction results for the four different VFS with corresponding human judgment for all data points in all the vehicle trajectories in the NGSIM dataset. The total number of data records that are classified as ‘Free Flow’ is lower according to model prediction than that from human judgment. This is compensated by the VFS ‘Acceleration’, where the model predicted more data points to be in this state when compared to human judgment. Based on Table 3.4, the sampling rates are high for all the data records that are classified as either ‘Acceleration’ or ‘Deceleration’, while the sampling rates are lower for data points identified as ‘Stopped’ or ‘Free Flow’ state. The fact that the proposed model identifies more ‘acceleration’ points than human judgment implies that more data points (information) will be retained if the proposed method is applied for vehicle trajectory collection, which is beneficial for traffic modeling applications, while the data reduction rate is still very significant. As a result, the proposed method and human judgment represent slightly different trade-offs between data reduction and information retaining. While it is hard to argue which one is better, it is certainly true that the proposed SAS method may be applied for large-scale mobile data collection, but human judgment cannot.

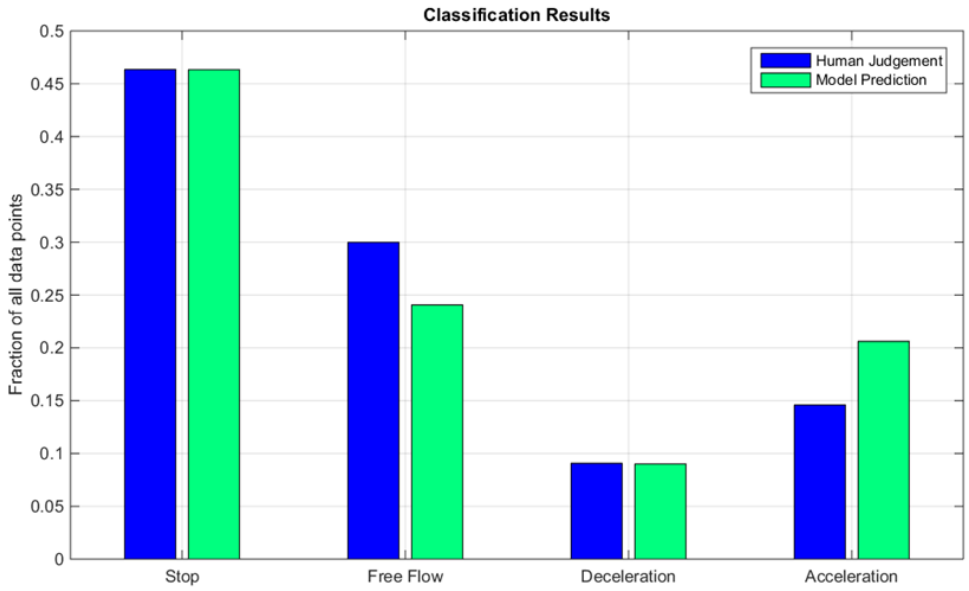


Figure 3.7. Comparison of model prediction with human judgment for different vehicle flow states

To further illustrate the performance of the vehicle state estimation method of individual data points, the model estimation results are compared against the VFS estimated by human judgment at each time instant. Table 3.6 gives the 'confusion matrix' of the classification for the NGSIM data. Each row of the matrix represents a state by human judgment, while each column a state by the proposed model. It was found that the most prominent type of mismatch occurs when individual data records identified as 'Free Flow' by human judgment are misclassified as 'Acceleration' by the model. Such mismatches are mainly from the times when a free-flowing vehicle further accelerates. According to human judgment, such data records are often considered to be in a 'Free Flow' VFS, whereas the proposed model predicts otherwise. In those cases, the model predictions can be actually more accurate than human judgment. In other words, human judgment is not necessarily the real "ground truth" especially when acceleration/deceleration states are considered.

Table 3.6. Confusion matrix for the classification of individual data records

Number of time instances = 189964		**Predicted Vehicle Flow State**			
		Stopped	**Free Flow**	**Deceleration**	**Acceleration**
Human Judgment	**Stopped**	87907	0	0	145
	Free Flow	0	45697	0	11270
	Deceleration	34	3	17107	82
	Acceleration	57	4	0	27658

The percentage of individual data records where ‘Free Flow’ or ‘Stopped’ VFS are classified as either ‘Acceleration’ or ‘Deceleration’ by the proposed HMM model, or vice versa were investigated. In 99% of cases, ‘Free Flow’ or ‘Stopped’ are classified as ‘Acceleration’ or ‘Deceleration’. This again implies that the SAS method will result in more retained data points and less data reduction, which is helpful for modeling applications, compared with human judgment.

Figure 3.8 shows the two-step classification results for a single-vehicle trajectory. It is a space-time graph of the trajectory. The slope of the graph denotes the instantaneous speed. Individual points are shown in different markers corresponding to their respective vehicle states. Figure 3.9 shows the result of data reduction by SAS. The dashed line shows the trajectory and the green markers denote the (retained) data points after reduction by applying SAS. Notice that, most of the reduction occurs in the ‘free flow’ and ‘stopped’ segments.

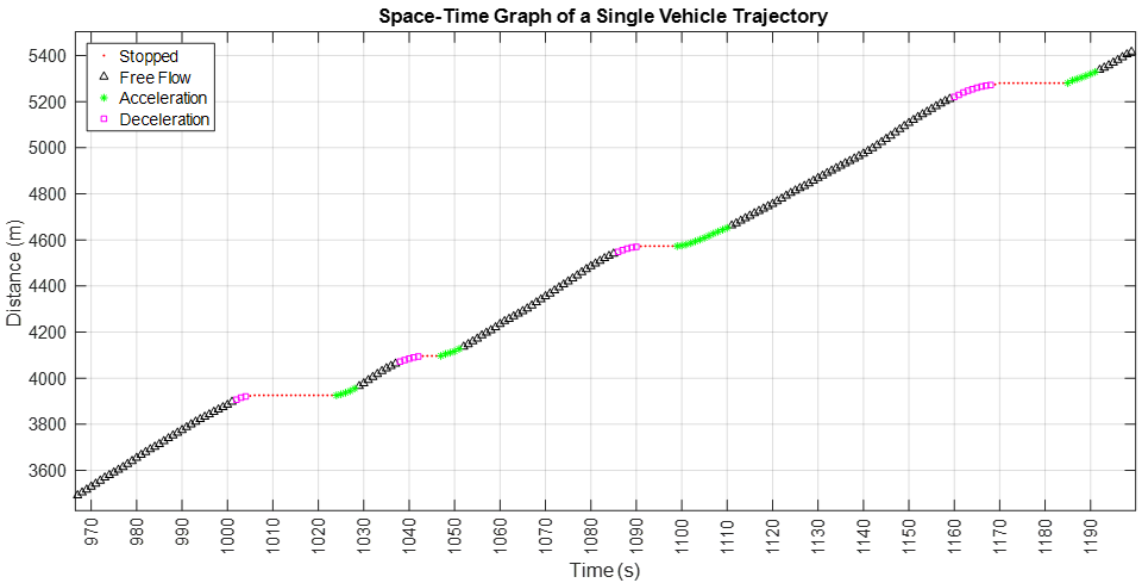


Figure 3.8. Space-Time graph of a single vehicle trajectory showing VFS estimation results

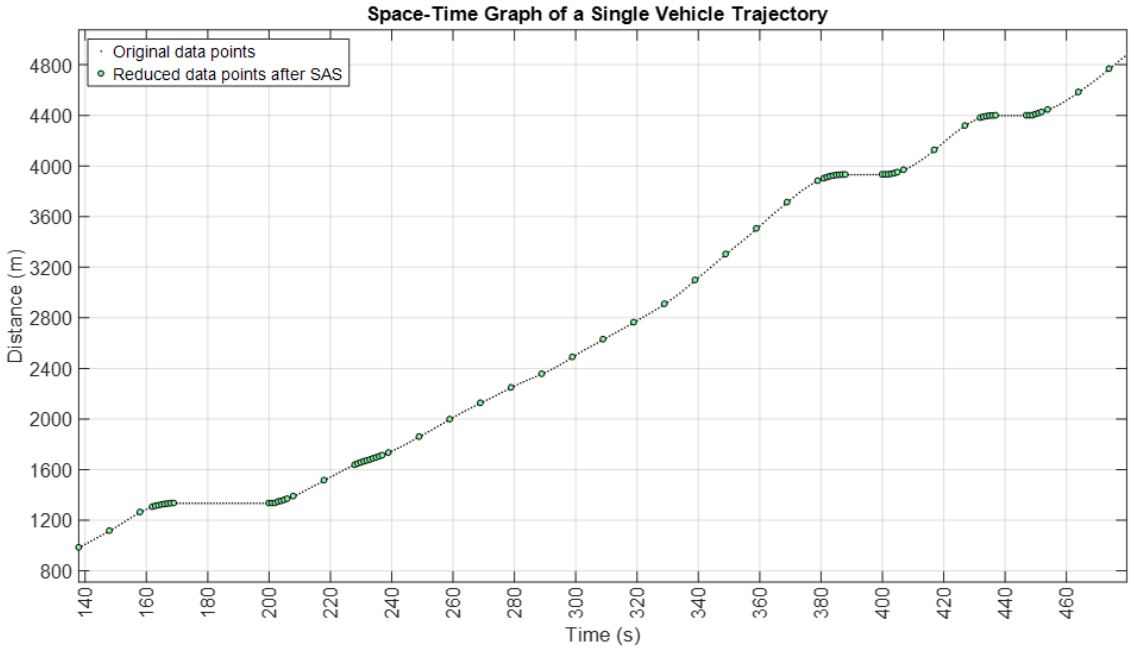


Figure 3.9. GPS data after SAS shown in a space-time graph of a vehicle trajectory.

3.6. TRAFFIC MODELLING APPLICATIONS

The effectiveness of the reduced data is further evaluated by certain transportation modeling applications in this subsection. In particular, this study focuses on the application of queue location estimation in Hao et al. (2015). This method studies the queue discharging process, which calculates the location (in the queue) and the acceleration rate of the vehicle simultaneously using kinematic equations. More details of the estimation method can be found in Hao et al. (2015) and are omitted here. The performance of the proposed method is measured by the Success Rate (the percentage of traffic signal cycles that the queue estimation method can be successfully applied to produce proper results), and the Mean Absolute Error (MAE) of estimated queue locations and queue times compared with observations, which is shown in the following equation.

$$MAE = \frac{\sum_{i=1}^{N} |\Delta x_i|}{N} \tag{3.5}$$

Here $\bar{x}$ is the estimated value, x^* is the observed value, Δx_i is the error term defined as $\Delta x_i = \bar{x} - x^*$.

Table 3.7 summarizes the queue location estimation results. In addition to the field dataset and NGSIM dataset discussed above, this study also uses trajectory data from a simulation model developed for Fresno, CA as part of the Corridor Management Plan Demonstration project for California Department of Transportation (Liu and Jabari, 2008). It can be seen that the success rates are essentially the same for the original data and the reduced data when used for the queue location estimation method in Hao et al. (2015). The MAEs calculated using the reduced data are slightly higher than the original data in cases of field test and simulation data, which are much higher for the NGSIM data. The high MAEs associated with NGSIM data might be due to the fact

that the trajectories were processed from video data and contain noticeably errors for some trajectories, e.g., a vehicle going backward for a short period of time, which was reported previously in the literature (e.g., Ban et al., 2011).

Figure 3.10 shows the variation of the Mean Absolute Errors and Success Rates with varying penetration rates. Simulation results are used to generate data for variable penetration rates. It is found that the success rates are consistent around 96% when the penetration rate is larger than 40%. The MAEs for queue length and queue location decrease with the increase of penetration rates. The results indicate that the reduced data can produce slightly degraded but very similar results compared with the original data.

Table 3.7.Queue Length estimation results for different data sets.

	Field Data		**Simulation Data**		**NGSIM Data**	
	Original data	**Reduced data**	**Original data**	**Reduced data**	**Original data**	**Reduced data**
MAE (Queue Location) (ft)	35.3853	41.1284	25.6831	27.846	37.7888	111.7089
MAE (Queue Length) (# of vehicles)	4.1168	4.3515	1.4326	1.5757	0.931	2.197
Success rate (%)	80%	80%	96.67%	96.67%	97.96%	100%

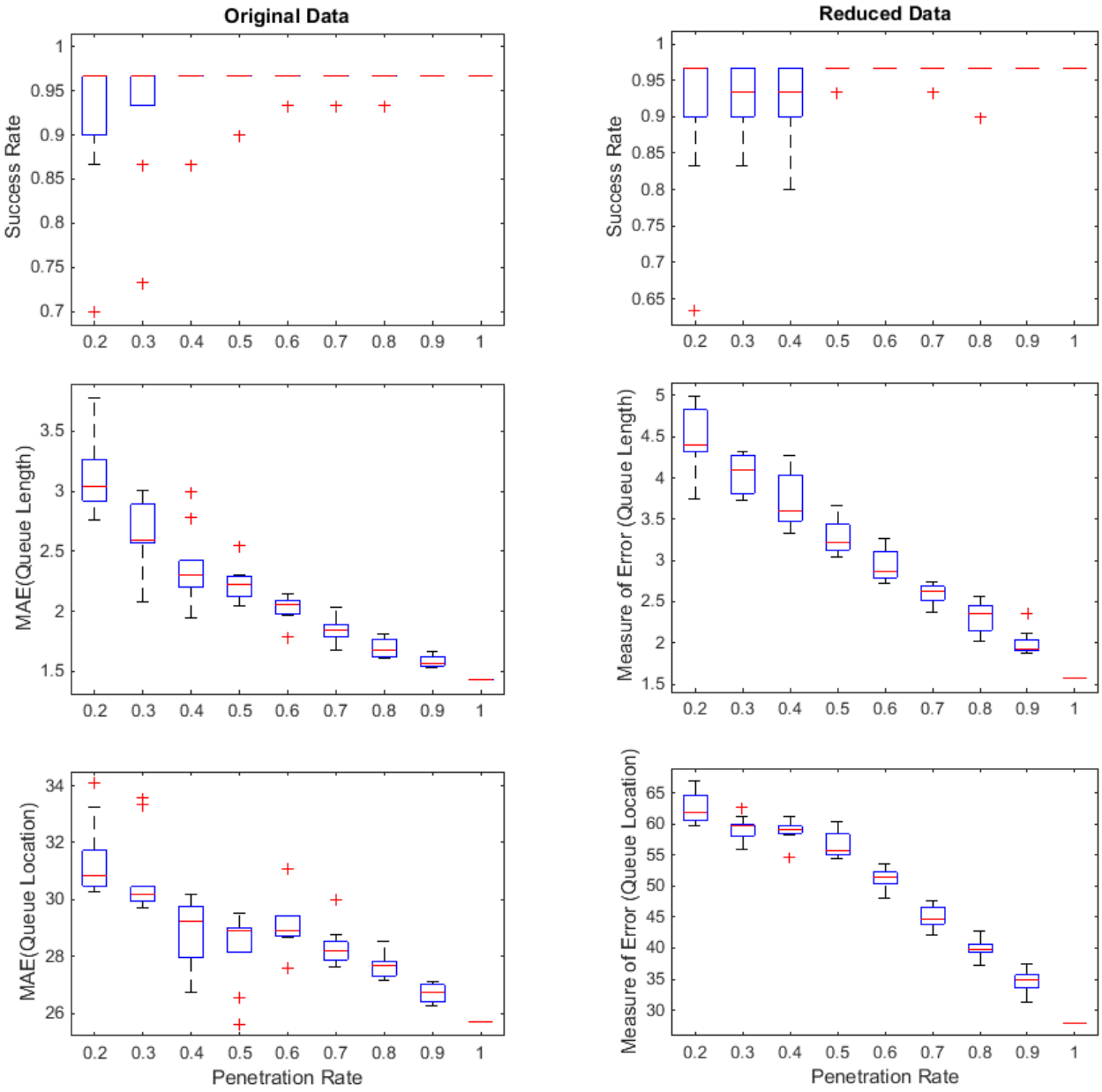


Figure 3.10. Comparison of queue location estimation results: original vs reduced data (simulation).

3.7. PRIVACY IMPLICATIONS OF SAS

Privacy can be defined as the *unlinkability* of a moving object for a certain distance/time (Sun et al., 2013). For urban environments, this study assumes that to satisfy modeling needs, some short traces of vehicles (say a few hundred feet) around an intersection are available. Then, privacy can be defined as the *unlinkability* of the short traces of the same vehicle over multiple (say M) intersections. In this study, M is set to two, which provides the highest level of privacy protection. This means that one can collect short vehicle traces around one intersection, but such short traces should not be linked together for the same vehicle for two or more intersections. Notice that the proposed SAS method itself does not consider privacy protection. This subsection simply shows whether the reduced dataset after applying SAS can help to improve protecting privacy, by applying the VTL-zone method in Sun et al. (2013).

To measure the effectiveness of the reduced data after applying SAS for privacy protection, the VTLzone-based system and related filtering approaches developed by Sun et al. (2013) were used, where traffic-knowledge-based adversary models were proposed and tested to evaluate the effectiveness of such a privacy protection system by making privacy attacks. Sun et al. (2013) referred to releasing all the traces collected in a VTL zone as the 'baseline approach'. It provides the least privacy (while the VTL zone system is applied), but the most usable data. To enhance privacy, they further proposed to filter out part of the location traces in the VTL zone. The filtering approaches include random sampling, individual probability-based, and entropy-based approaches. In terms of privacy protection, the performance of the privacy models can be evaluated by applying adversary models. In their paper, two criteria were proposed for privacy evaluation purposes, namely, the percentage of correctly tracked traces (P1), and the percentage of correct inferences (P2). P1 indicates the probability that the traces of one vehicle can be successfully

linked at two VTL zones, obtained by using the number of correct inferences divided by the total number of traces going through both the VTL zones. P2 is obtained using the number of correct inferences divided by the total number of inferences, which indicates how accurate the inferences are. These two measures are used in this study.

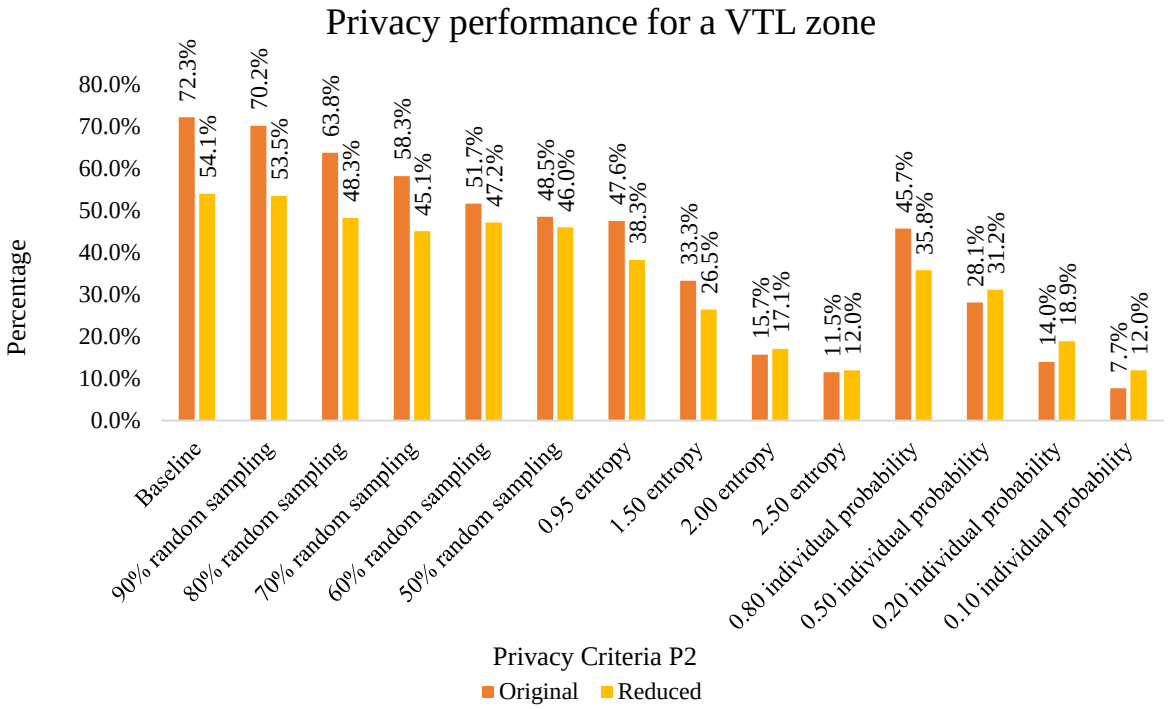
The adversary models were implemented using different filtering approaches for both original and reduced data after applying the SAS method. This study uses the NGSIM dataset for this testing. First, SAS is applied to the raw data. A list of released traces in VTL zones is found from different filtering approaches. The adversary model developed by Sun et al. (2013) is then used to make privacy-based attacks on each of the released traces in the northbound direction of both original and reduced data. The comparison of results is given in Table 3.8.

Figure 3.11 illustrates the summary of the privacy evaluation criteria P1 and P2 for different filtering approaches using the original and reduced datasets. Compared to the original data, the reduced data after applying SAS shows better performance in terms of privacy protection as both P1 and P2 has lower values for the reduced data in most cases. It means the SAS technique can not only be applied to the baseline data but also can be combined with privacy-based filtering approaches to further improve its privacy protection. The original data outperforms the reduced data only in cases of individual probability-based models where the individual tracking probability is smaller than 0.8.

Notice that enhancing privacy protection sometimes may degrade the data quality for certain applications, as discussed in detail in Sun et al. (2013). This also applies to the reduced dataset after applying SAS, in a way similar to those shown in Sun et al. (2013). The details are omitted here for brevity.

Table 3.8. Privacy performance of the reduced Data vs. baseline Data.

Filtering approaches	**Data type**	**No. of released traces (both zones)**	**No. of released traces - Upstream**	**No. of released traces - Downstream**	**No. of inferences**	**No. of correct inferences**	**Percentage of tracked traces (P1)**	**Percentage of Correct inferences (P2)**
Baseline	Original	131	138	134	137	99	75.57%	72.26%
	Reduced	130	137	133	135	73	56.15%	54.07%
90% random sampling	Original	118	132	126	131	92	77.97%	70.23%
	Reduced	118	132	126	129	69	58.47%	53.49%
80% random sampling	Original	93	118	112	116	74	79.57%	63.79%
	Reduced	93	118	112	114	55	59.14%	48.25%
70% random sampling	Original	72	104	97	103	60	83.33%	58.25%
	Reduced	72	104	97	102	46	63.89%	45.10%
60% random sampling	Original	56	91	84	91	47	83.93%	51.65%
	Reduced	56	91	84	89	42	75.00%	47.19%
50% random sampling	Original	40	74	70	68	33	82.50%	48.53%
	Reduced	40	74	70	63	29	72.50%	46.03%
0.95 entropy	Original	64	112	88	103	49	76.56%	47.57%
	Reduced	64	112	88	94	36	56.25%	38.30%
1.50 entropy	Original	32	92	68	84	28	87.50%	33.33%
	Reduced	32	92	68	68	18	56.25%	26.47%
2.00 entropy	Original	10	59	71	51	8	80.00%	15.69%
	Reduced	10	59	71	41	7	70.00%	17.07%
2.50 entropy	Original	4	34	85	26	3	75.00%	11.54%
	Reduced	4	34	85	25	3	75.00%	12.00%
0.80 individual probability	Original	59	119	80	105	48	81.36%	45.71%
	Reduced	59	119	80	95	34	57.63%	35.79%
0.50 individual probability	Original	31	105	66	89	25	80.65%	28.09%
	Reduced	31	105	66	77	24	77.42%	31.17%
0.20 individual probability	Original	9	61	67	43	6	66.67%	13.95%
	Reduced	9	61	67	37	7	77.78%	18.92%
0.10 individual probability	Original	4	42	50	26	2	50.00%	7.69%
	Reduced	4	42	50	25	3	75.00%	12.00%



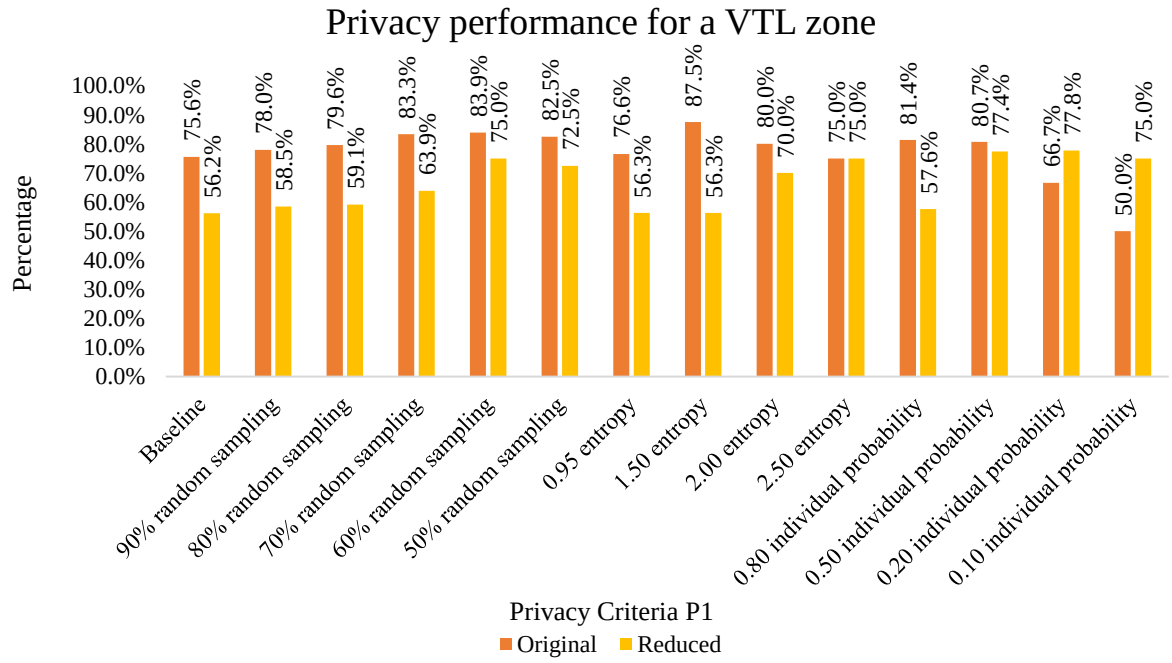


Figure 3.11. Comparison of privacy performance.

Chapter 4. THE SELF-ADAPTIVE ONLINE TRAJECTORY SAMPLING (SAOTS) METHOD

4.1. OVERVIEW OF SAOTS ALGORITHM

A major limitation of SAS is that it is unable to run online or near real-time. In the case of SAS, the entire trajectory data is assumed to be collected first and then used to find the resampling strategy. By design, it is a batch processing method, which is suitable for post-processing applications that can tolerate large latency. To improve the proposed SAS method, the target is for near real-time (i.e., "online") applications that require minimum latency to receive the data. For example, a mobile device is sending trajectory data to a central server; the server is using the data for (near) real-time decisions, such as volume or speed estimation for ramp metering control or signal timing control. For those online applications, the server cannot wait until the entire vehicle trajectory is finished. Therefore, the data compression algorithm has to be online to perform data compression and then send the compressed data (to minimize data transmission) as quickly as possible to the server for such decisions. Moreover, the Hidden Markov Model (HMM) used for VFS identification relies on manually classified (labeled) training data. Labeling trajectory segments manually is a time-consuming process which also depends heavily on human judgment. As a result, developing separate custom models for different roadway condition and traffic network using a large amount of manually labeled trajectory segments would demand a considerable effort.

This section presents a self-adaptive online trajectory sampling (SAOTS) method for vehicle trajectory resampling and reduction. First, an online trajectory segmentation method based on the estimated VFS is presented. The next section explores the spectral domain properties using a

trajectory database that includes different types of trajectory segments. A spectral-domain property called *critical frequency* is introduced that reflects the nature of speed variations experienced by an individual vehicle. It was seen that this property could be utilized to determine the proper resampling frequency. Finally, SAOTS applies the critical frequency of each segment to sample the data records in that segment. As opposed to fixed sampling schemes, the sampling strategy proposed here is adaptive to the VFS. Results show that resampling the trajectory segments based on the critical frequencies can vastly reduce the size of the GPS data while conserving most of the useful information.

SAOTS has three-fold benefits over SAS: (i) instead of using HMM that requires sizeable training data, SAOTS applies a semi-supervised HMM method that can significantly reduce the size of the training data; (ii) instead of some fixed sampling rules, SAOTS uses the underlying spectral-domain property of a trajectory segment to automatically determine a proper sampling interval, and (iii) unlike SAS, SAOTS is designed to operate online, and thus able to transmit trajectory data points to the central server near real-time. Compared with existing trajectory compression methods, SAOTS is designed to integrate traffic knowledge, in particular, VFS to segment and sample trajectories which can significantly improve its performances. The implementation of online trajectory compression algorithms including SAOTS requires local computation and storage capability, which certainly represents an engineering trade-off with the amount of data transmitted. However, with the high specifications (computational power and storage capabilities) of mobile devices these days, implementing such an online algorithm on the client side should not create any major issues.

4.2. Online Trajectory Segmentation based on Estimated VFS

A trajectory segment is defined as a portion of the trajectory, where the vehicle undergoes a specific type of motion (Siddique and Ban, 2018). Each segment (and its constituent data points) is labeled with a vehicle flow state (VFS) to indicate the type of motion. Unlike the batch method in Siddique and Ban (2018), for online trajectory sampling, we cannot wait until the entire vehicle trajectory information is collected to start segmentation. Instead, a segmentation buffer is defined to receive trajectory data and perform the segmentation almost instantly. As shown in Figure 4.1, the segmentation buffer essentially defines the maximum length of a segment (in terms of its time duration). The segmentation will be done (and a trajectory segment identified) when either the segmentation buffer is reached or the HMM identified a different VFS in the new trajectory point. In either case, a trajectory segment is identified; the sampling algorithm will then run on the segment (see the flowchart in Figure 4.3) and the reduced data will be transferred from the client-side (e.g., a mobile device) to the central server. Thus, the segmentation buffer also represents the maximum latency of the SAOTS.

As shown in Figure 4.1, there are clear tradeoffs for the length of the segmentation buffer. Longer buffers will always lead to better segmentation, which will also bring larger latency; shorter buffers mean smaller latency, which however may result in inaccurate segmentation or fragmented segments (i.e., one long “stopped” segment may be identified as two short “stopped” segments). Figure 5.4 (a) in Section 5.5 shows that around 98% of the segments observed in a Didi dataset are shorter than 100 seconds and therefore in this study, the length of the segmentation buffer is set as 100 seconds. However, it needs to be clarified that it is just an upper bound of the actual length of a segment and thus is more like a constraint to check the excessive latency of the online resampling algorithm. Therefore, it does not need to be super accurate and some rough estimate can work

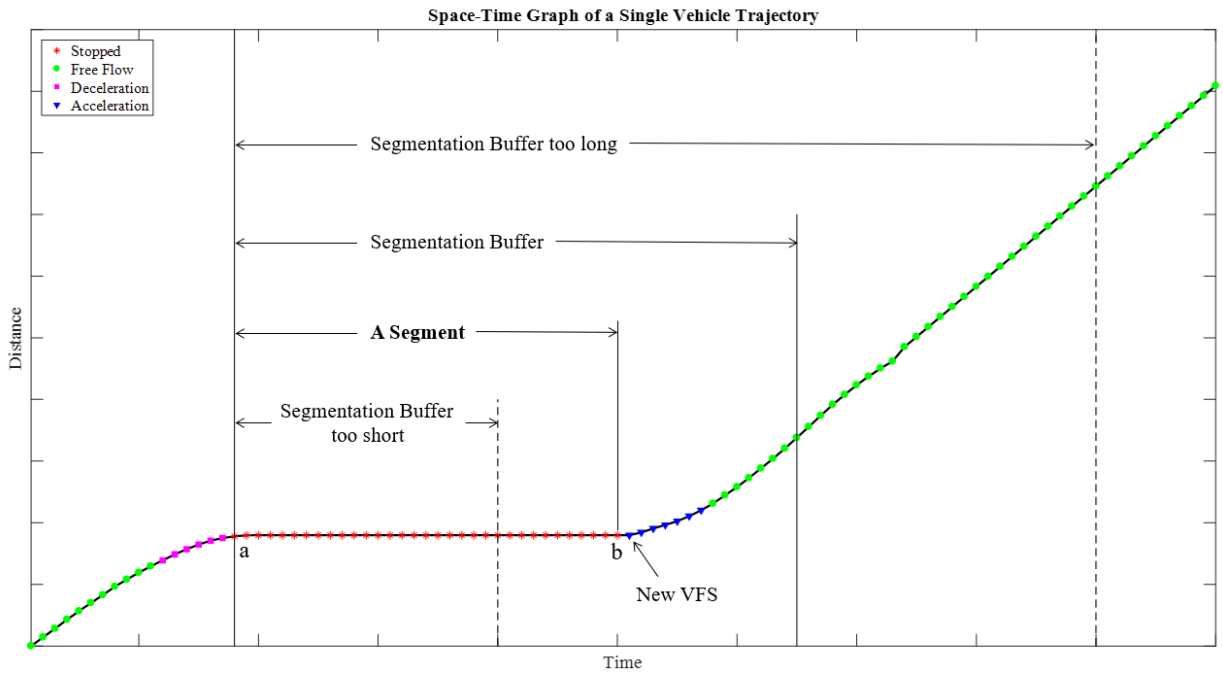


Figure 4.1. Illustration of Segmentation Buffer.

satisfactorily. The actual length of a segment is determined by the SAOTS algorithm and often shorter than this segmentation buffer. The latency of the resampling is decided by the length of a segment (not this buffer).

4.3. Sampling based on Spectral Domain Analysis

Probably due to its inherent spatiotemporal nature, in the field of traffic data analysis, spectral domain-based methods have received much less attention compared with time domain-based methods. Li et al. (2010) proposed a data analysis framework to extract traffic oscillation attributes such as the period and magnitude using spectral-domain signal processing techniques. The approach is effective in unveiling oscillation properties from noisy raw data collected using loop detectors. More recently, Zhao et al. (2014) proposed a spectral envelope method to analyze traffic oscillations using simulation and detector data. They found that spectral envelopes can reveal not only the salient frequencies of periodic oscillations of traffic flow but also the relative strength of

these oscillations at different locations. This section presents a spectral-domain analysis method to determine the sampling frequency of a trajectory segment identified in Section 5.2.

Any signal whose amplitude is a function of time has a corresponding frequency spectrum. When the signal is viewed from the frequency (or spectral) domain, certain aspects of the signal or the underlying processes producing it can be revealed. In some cases, the frequency spectrum may include a distinct peak corresponding to a sine wave component. However, for real-world signals, such as the trajectory of a moving vehicle studied in this paper, there may be multiple peaks indicating a signal that is not simply sinusoidal. Energy spectral density (ESD) describes how the energy of a signal or a time series is distributed in the spectral domain. In other words, it shows at which frequencies time-domain variations are strong and at which frequencies time-domain variations are weak (Oppenheim et al., 1978). Let's assume $x(t)$ denotes a signal as a function of time, where t is the (continuous) time. Then, the ESD of signal $x(t)$ is defined as:

$$ESD = \int_{-\infty}^{\infty} |x(t)|^2 dt \tag{4.1}$$

Now, for the purpose of this study, let's focus on a segment of a vehicle trajectory. The spectral domain analysis focuses on the instantaneous speeds of the vehicle. In this case, the signal is discrete (e.g., one data point per second) and the duration of the segment is finite (e.g., 5 minutes), leading to a finite number of data points (e.g., 300). Denote f the sampling frequency of the signal, then according to Parseval's theorem (Stein, 2000), the ESD of a discrete signal $x(n)$ can be defined as (n here denotes the discrete-time instant):

$$ESD = |\hat{x}(f)|^2 \tag{4.2}$$

where $\hat{x}(f)$ is the discrete Fourier transform (DFT) of $x(n)$. DFT converts a finite sequence of equally-spaced samples of a signal into a same-length sequence of an equally spaced complex-valued function of frequency. The DFT is, therefore, a spectral domain representation of the original input sequence (signal). In this case, the DFT transforms a trajectory segment with N sample points $\{x(n) := x(0), x(1), x(2) \ldots\ldots x(N-1)\}$, into a sequence of complex numbers, which is defined by:

$$\hat{x}(f) = \sum_{n=0}^{N-1} x(n) e^{-2\pi i f n/N} \tag{4.3}$$

The DFT of the signal (i.e., the speeds of a vehicle trajectory segment in this study) was performed using an algorithm called the Fast Fourier Transform (FFT). Because the FFT of a real signal is symmetric, the energy at a positive frequency is the same as the energy at the corresponding negative frequency. In this study, therefore, we only look at the positive side of the spectrum. Once the single-sided ESD is plotted using Equations (4.2) and (4.3) above, the area under the ESD curve represents the total energy of the signal according to Parseval's theorem (Oppenheim et al., 1978). The ESD array values are proportional to the amplitude squared of each frequency component making up the time-domain signal. As a result, the shape of the ESD shows the nature of the dominant frequencies; see Figure 4.6 (b) below for an example. In theory, a signal normally contains all frequencies from $-\infty$ to $+\infty$. In practice, considering only a small fraction of the frequencies and ignoring very high frequencies of the signal is probably sufficient to conserve the majority of the signal. The energy of a signal up to a certain frequency, i.e., the area under the ESD curve of the signal, can be used as a metric to measure how good the original signal can be conserved if we only consider the signal up to that frequency.

In case of the speed profile of a vehicle trajectory segment, the ESD usually reveals that the majority of the 'peaks' occur on smaller frequencies. This implies that considering up to a small portion of the frequencies may be sufficient to conserve most of the energy in that signal, which can be used to recover the majority of the signal. This study defines the critical frequency of a signal that corresponds to a certain percentage of the total energy (for example, 90%) of the signal. Using this frequency, we can find the sampling interval to resample the signal (i.e., the trajectory segment). The resampled signal often contains much fewer data points than the original signal, thus significantly reducing the data size, while at the same time ensuring that most of the energy of the signal is retained. The estimated trajectory segment can be then reconstructed using piece-wise linear interpolations. To evaluate how close the estimated trajectory segment is to the original trajectory segment, the mean absolute percentage error ($MAPE$) of the reconstructed trajectory segment is considered. The $MAPE$ is calculated using the following equation:

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\bar{x}_i - x_i^*}{\bar{x}_i}\right| \times 100\% \tag{4.4}$$

Where, $\bar{x}_i$ is the actual value and x_i^* is the estimated value of the speeds in the trajectory segment. A demonstration of the method is illustrated in Figure 4.2. Figure 4.2 (a) shows the speed profile of a vehicle trajectory segment in the time domain. The length of the segment is 144 seconds. The data contain location and speed information for every second; thus, the original sampling frequency of the 'signal' is 1 Hz. Figure 4.2 (b) shows the ESD of the signal in Figure 5.2 (a). The y-axis of the ESD plot represents the energy of the frequency. We can see that smaller frequencies often have higher energies, which implies that the majority of the energy of the spectrum can be covered if only a small portion of the frequencies is considered. Figure 4.2 (c) illustrates the increase in the cumulative energy conserved by increasing frequencies. In this case,

considering only up to 0.2 Hz can yield 90% energy of the signal. This frequency translates into a sampling frequency of 0.4 Hz according to Nyquist–Shannon sampling theorem (Oppenheim, 1999), i.e., a sampling interval of about 3 seconds. The trajectory segment is then resampled by considering 1 data point every 3 seconds, as shown in Figure 4.2 (d). The piece-wise linear interpolation of speeds provides a $MAPE$ of nearly 6% for the resampled trajectory. This demonstrates that using only 33% of original data records, a trajectory segment can be resampled with satisfactory results. In this study, once a trajectory segment is identified the above spectral analysis of the segment is performed in order to determine the proper resampling interval of the segment. This is the sampling step of the trajectory compression method proposed in this study.

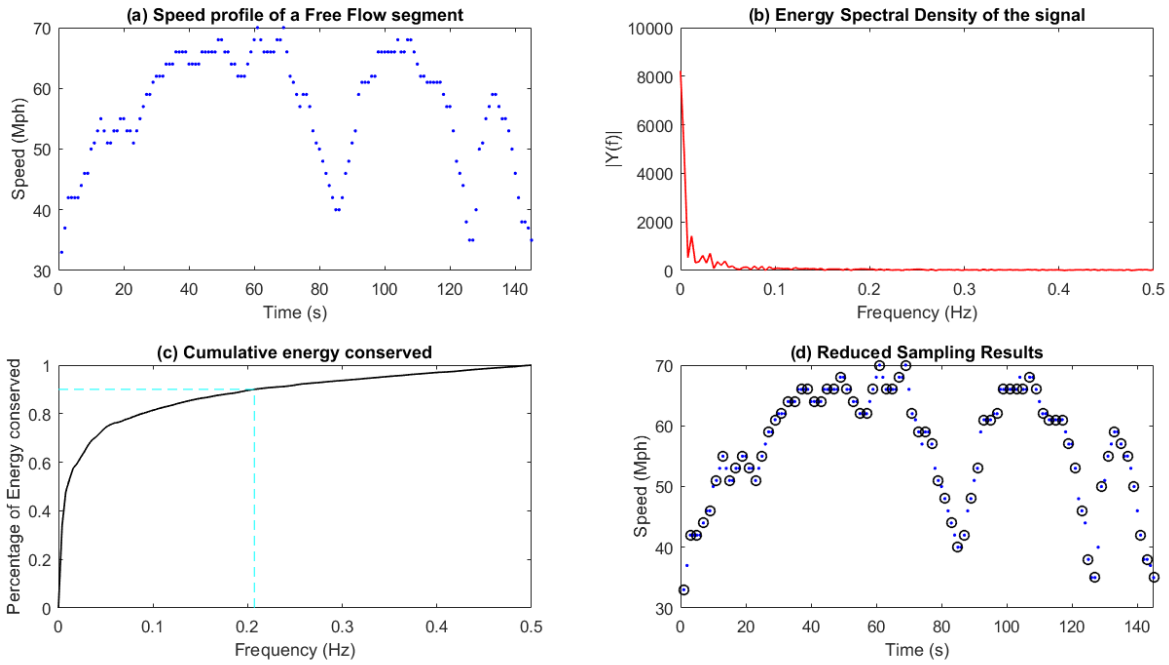


Figure 4.2. Demonstration of frequency spectrum analysis results in a trajectory

4.4.SAOTS ALGORITHM

Integrating the online segmentation method in Section 4.2 with the spectral domain analysis method in Section 4.3 leads to a Self-Adaptive Online Trajectory Sampling (SAOTS) algorithm. Figure 4.3 shows the flowchart of the algorithm. First, there are two parameters that need to be set up before the SAOTS algorithm is initiated. They are the initial buffer and the segmentation buffer. SAOTS initiates with a small initial buffer. The initial buffer size is chosen to create several typical segments, which is chosen as 60 seconds in this paper. This buffer is used to initiate the incoming data, to accommodate the occasional 'cold start' of GPS sensors, and to warm up the HMM classifier to classify individual data records. The second parameter, the segmentation buffer, is used to ensure that the latency of the online application is not too long, as discussed in Section 4.2. Since the resampling happens after a segment is identified, the length of the segment (in time) also represents the latency of the algorithm, i.e., how soon the data from a trajectory segment can be released after the first data point of the segment is received. Limiting the segmentation buffer length helps control the latency of the algorithm in real-time applications. In this study, the segmentation buffer length was chosen as 100 seconds as detailed in Section 4.2.

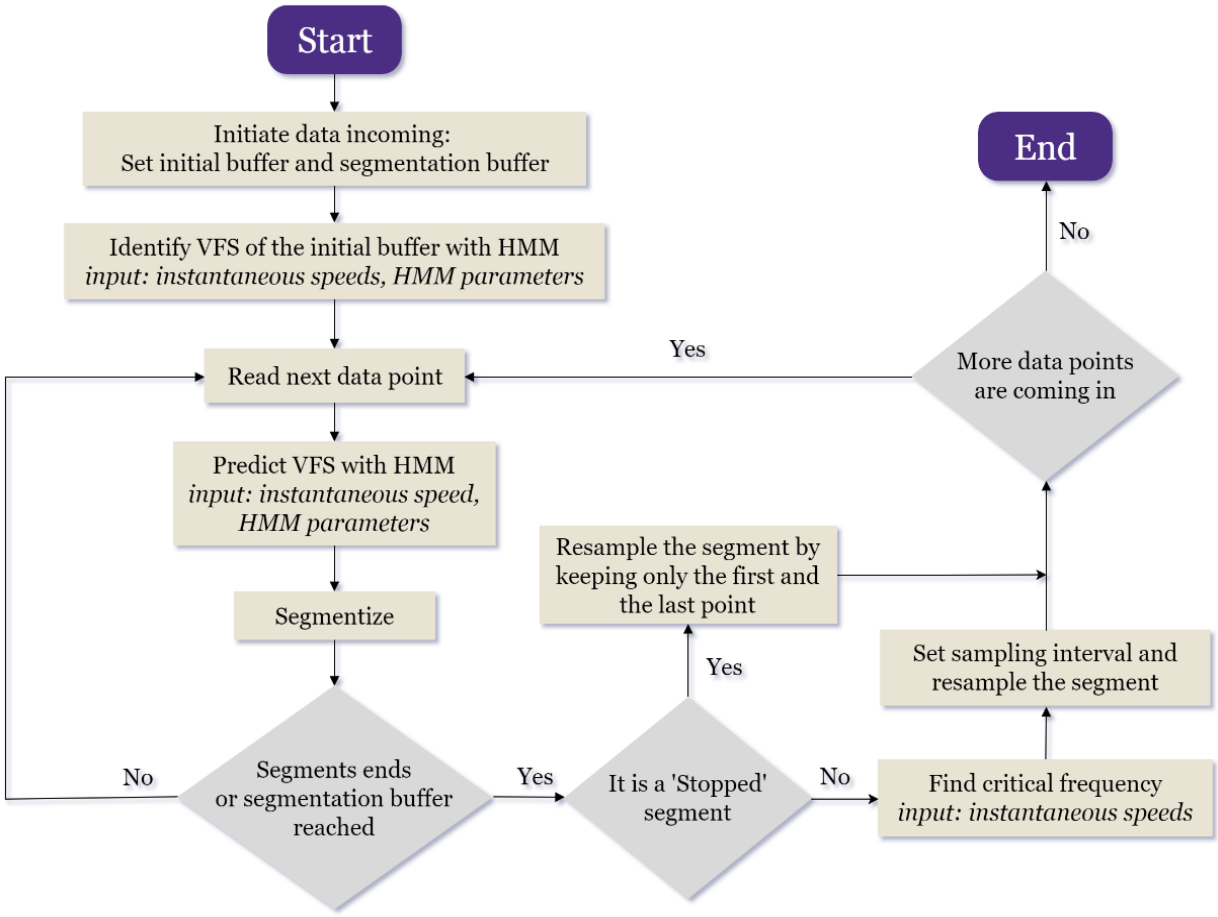


Figure 4.3. Flowchart of SAOTS algorithm.

Once the algorithm is initialized, it waits until the initial buffer is filled. The individual data points inside this buffer are then classified using the HMM classifier. This classifier uses the instantaneous speeds of the data as the input to label each incoming data point. If the label of a new data record is different from the previous record, this implies the start of a new segment. The start of a new segment is also triggered when the segment reaches the segmentation buffer length. Once any of these two criteria are satisfied, a segment is identified with a given VFS. If the VFS of the segment is identified as “Stopped”, then only the first and the last data records are sampled. For other types of segments, SAOTS then conducts spectral domain analysis for the identified segment as discussed in section 3.8 to find the critical frequency of the segment. This critical frequency is used to find the sampling interval of the data records of the segment. The same process repeats itself until no trajectory data point is received, as shown in Figure 4.3. The latency of the algorithm varies depending on the length of the segment that is just identified. The average latency among all segments of the trajectory can be used as the overall latency of the algorithm.

The proposed SAOTS method relies only on the collected trajectory data, for which road network information is not needed. As a result, it can deal with vehicle trajectories from either known or unknown road networks.

4.5. Exploring Spectral Domain Properties

This section explores the spectral domain properties using a dataset of 58 different vehicle trajectories. This trajectory dataset was recorded from vehicles in Albany, New York, and Seattle, Washington, covering various traffic conditions. The original sampling interval of these trajectories is 1 second. To investigate the spectral domain properties, the trajectories were broken into segments based on the VFS of the individual data points. A total of 3,285 segments are identified after the segmentation process. The distributions of the segment lengths and segment types are demonstrated in Figure 4.4. The length of the segments varies from 2 seconds to 995 seconds. However, the majority (around 98%) of the segments are less than 100 seconds long (See Figure 4.4 (a)). The distributions of four types of segments are almost the same as seen in Figure 4.4 (b). The distribution of the number of data points in different types of segments is also very similar, which is omitted here.

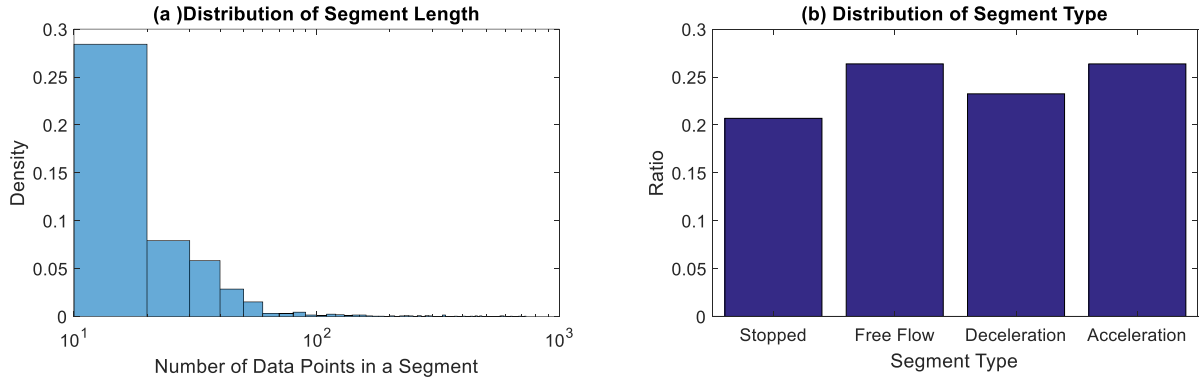


Figure 4.4. Distribution of segment length and type

Figure 4.5 illustrates the results of the analysis of MAPEs and sampling intervals. The $MAPEs$ for different sampling intervals and segment lengths are plotted in Figure 4.5 (a). It shows that, considering high energy conservation results in lower $MAPE$. This is quite expected, as conserving

higher energy translates into a smaller sampling interval (see Figure 4.5 (b)). Longer segments seem to exhibit smaller $MAPEs$. The $MAPE$s vary most for the 'stopped' segments (see Figure 4.5 (c)). It is also observed that 'stopped' segments often tend to retain the original sampling rate of 1 second (see Figure 4.5 (d)) based on the spectral domain analysis. This is quite expected, as these segments are mostly composed of zero speed values, and the sampling based on spectral-domain analysis seems to struggle due to near-zero energy of the signal. The 'free flow' segments exhibit lower MAPEs when compared to other types of segments. This means that in the trajectory segments where the vehicle moves in a consistent speed, the reconstructed trajectory yields more accurate results compared to other types of segments.

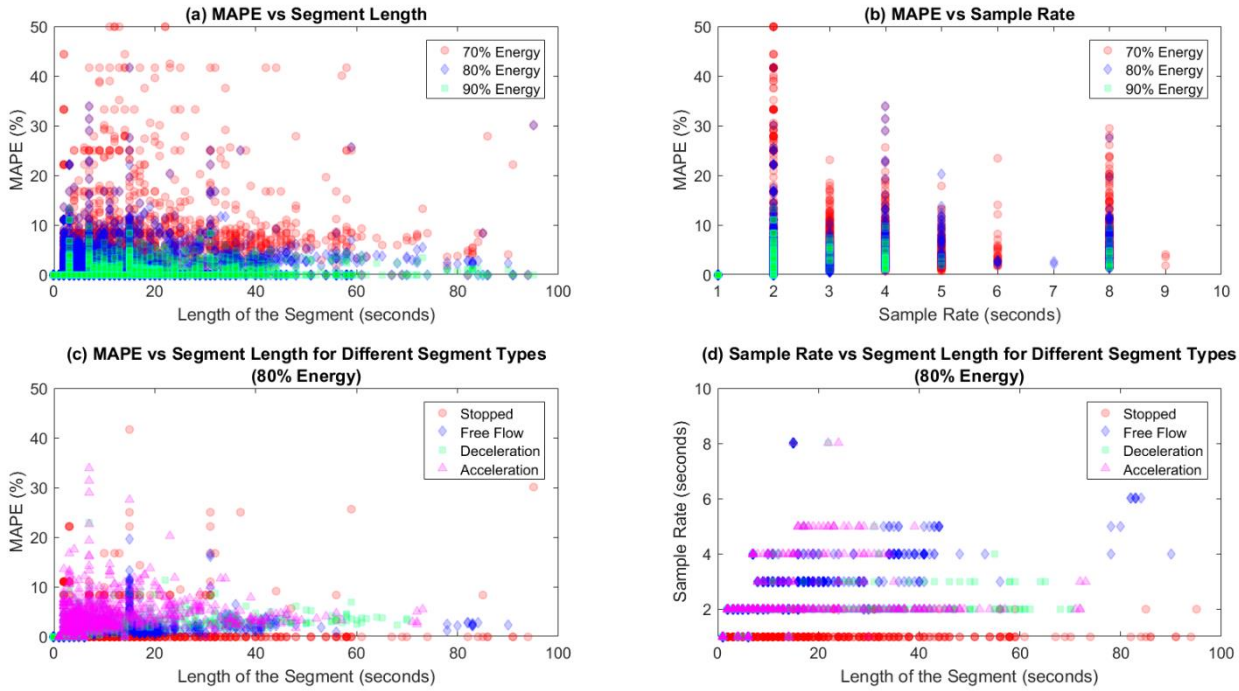


Figure 4.5. Results of analyses considering different energy consideration and segment type.

Chapter 5. Comparing SAS and SAOTS with Other Trajectory Compression Algorithms

This chapter presents the results of different numerical analyses that were performed to compare SAS and SAOTS with other existing trajectory compression algorithms. Two different trajectory datasets were used for these studies, a small, 'field' dataset, and a large dataset. The field dataset contains 58 different vehicle trajectories (supplemented by a 111-trajectory dataset for queue length estimation) that were recorded from vehicles in Albany, New York, and Seattle, Washington, covering various traffic conditions in an urban setting. The original sampling interval of these trajectories is 1 second. On the other hand, the large dataset is from a Chinese ridesourcing company, Didi Chuxing (https://outreach.didichuxing.com/research/opendata/en/). This dataset contains approximately 60 million data points, which were collected from the City of Chengdu and the City of Xi'An in China. The sampling interval of the data points in this dataset varied between 1 to 10 seconds.

This chapter provides an empirical comparison of five trajectory compression algorithms. They include SAS and SAOTS proposed in this study and three other trajectory compression algorithms: two online algorithms and one batch processing algorithms. The other three algorithms are listed below.

- Bottom-Up (BU): Starting from the finest possible approximation (considering every two points as a segment), segments are merged until the stopping criteria are met. This study sets a maximum segment error = 0.0003 as the stopping criteria.

- Sliding Window (SW): Using this algorithm, a trajectory segment is grown until it exceeds some error bound This study sets a maximum segment error = 0.0003 as the error bound.

- Sliding Window and Bottom-Up (SWAB): The SWAB algorithm keeps a small buffer of data points. Bottom-Up is first applied to data in the buffer and the leftmost segment is reported. The data corresponding to the reported segment is removed from the buffer and more data points are read in. The number of data points to read in depends on the structure of the incoming data, which is performed by the Sliding Windows algorithm. These points are incorporated into the buffer and Bottom-Up is applied again. The process of applying Bottom-Up to the buffer, reporting the leftmost segment, and reading in the next data subsequence by applying Sliding Windows is repeated as data arrive in real-time. This research uses an initial buffer of 100 data points. Other parameters are kept the same as those in Keogh et al. (2001).

A "spectral resampling" batch compression method was also tried that performs the spectral analysis on the entire trajectory (i.e., a batch algorithm) to determine its critical frequency and then apply the frequency to resampling the trajectory. It turns out that such a method does not perform well when compared with the other batch methods (SAS and BU). Therefore, the detailed results of this spectral resampling method are not included in this dissertation. The field dataset is used for all the comparison except queue length estimation. To measure the error in queue length estimation, signal timing information is needed in addition to the trajectory data. The field dataset described in the previous section does not contain any signal timing information, and thus it could not be used for queue estimation. A separate trajectory dataset with relevant signal timing information is then used for comparing the performance of the trajectory compression algorithms for queue length estimation. Furthermore, the linear interpolation method is used to reconstruct the resampled trajectory. Fitting a polynomial to recreate the vehicle trajectories may yield better results in many cases than the piecewise linear approximation. However, in this case, different orders of polynomials might be needed to recreate different segments accurately. This may create

unnecessary complexity and computational burden to reproduce the trajectory, which can be an issue especially for online algorithms studied in this paper. On the other hand, the assumption of linearity may have an inherent error, but it is still the most basic and straight forward way to reconstruct the trajectory with fewer data points (especially for online algorithms), and thus is widely used by many trajectory compression algorithms (Shatkay, 1995; Park and Lee,1999; Keogh et al.,2001). Table 5.1 provides a qualitative comparison among these compression algorithms.

Table 5.1. Comparison of Trajectory Compression (TC) Algorithms.

TC Algorithm	**Online Algorithm**	**VFS**	**Controlling Parameters**
SAS	No	Yes	The sampling interval for different types of trajectory segment
SAOTS	Yes	Yes	Percent energy conserved
Sliding Window	Yes	No	Maximum Spatial error in a segment
Bottom Up	No	No	Maximum Spatial error in a segment/ Number of segments
SWAB	Yes	No	Maximum Spatial error in a segment/ Number of segments

5.1. COMPARISON USING FIELD DATA

Several evaluation criteria were used to compare these algorithms, which are described below. This section also discusses the performance of each trajectory compression algorithm using the filed dataset under these evaluation criteria, based on the results in Table 5.2.

- Percentage Data Reduction (PDR): This is the ratio of the number of data records reduced in the resampled trajectory and the number of data records in the original trajectory. A high PDR means a high percentage of data reduction and is thus preferred. SAOTS has the

highest average PDR of 72.29% among all three online algorithms (actually among all five algorithms tested here) for the trajectories used in the numerical experiments here.

- Spatial Error: This is the summation of the perpendicular Euclidean distances between the Lat/Lon pairs of original data records and that of the resampled trajectory segment. Among the three online algorithms, SWAB has the lowest spatial error of 0.0002 degrees, and SAOTS exhibits the second-lowest Spatial Error of 0.0003 degrees.

- $MAPE$ of Speeds: The formula to calculate $MAPE$ of speeds in a resampled trajectory is given in Equation (4.4). Generally, the batch algorithms have lower $MAPE$s than online algorithms. We find that SAS provides the lowest $MAPE$ (2.15%) for the resampled trajectories while compared with other trajectory compression algorithms. SAOTS has a MAPE of 4.3%, which is better than the $MAPE$s of the two other online algorithms, the Sliding Window (6.49%) and SWAB (7.55%).

- Latency: This evaluation criterion is crucial and only applicable to online algorithms. These algorithms need to create a buffer while making resampling decisions. Latency is defined here as how soon the data from a trajectory segment can be released after the first data point of the segment is received. This is also the time difference between the actual time of reading the first data record of the segment and the time of making the decision of how to form and resample the segment. A compression/segmentation algorithm with larger latency will result in a larger delay while processing the trajectory data, which is less desirable. In the case of SAOTS, the length of the segmentation buffer is the maximum latency of a trajectory segment. The actual latency is determined by the length of the trajectory segments, which is often smaller than the segmentation buffer. For all three

online algorithms, the computation time of the algorithm is very small compared to the segment length/segmentation buffer and thus not considered in the latency calculation. Figure 5.1 illustrates the latency distributions of the three online algorithms. From the distributions, it is evident that SAOTS has the highest number of data records sampled in very low latency, resulting in an average latency of about 29.73 seconds. Sliding Window has the lowest overall latency, with an average of 9.09 seconds. SWAB shows a wide range of latencies, which are almost uniformly distributed between 1 to 600 seconds, with an average latency of 290.72 seconds.

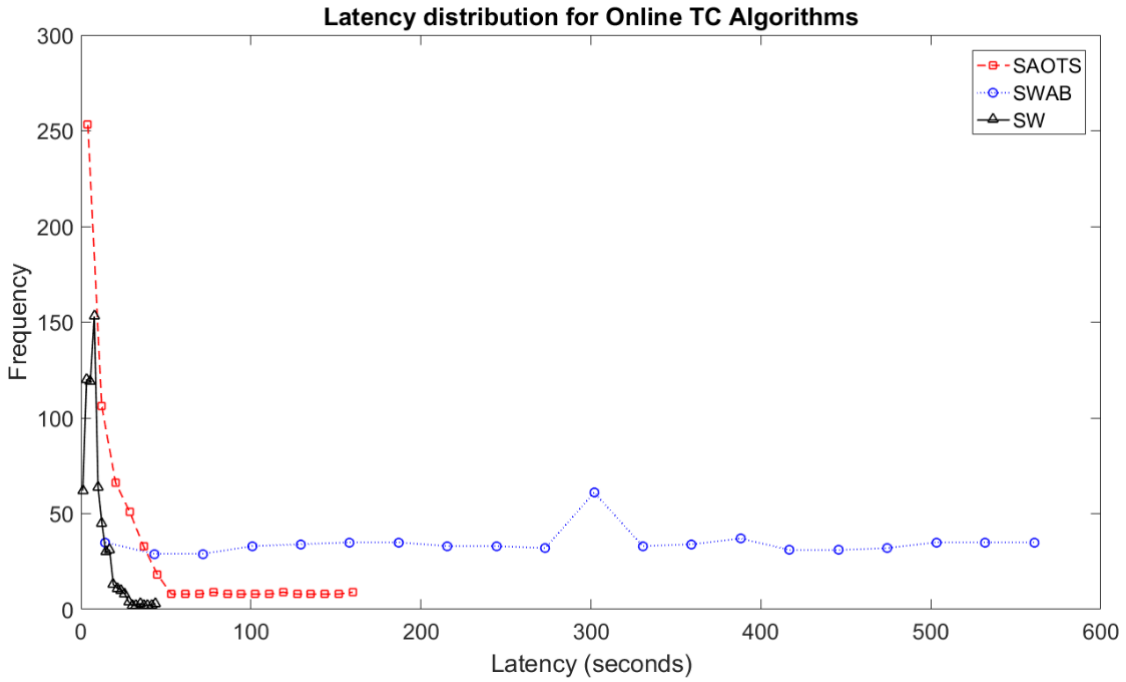


Figure 5.1. Latency distribution of online trajectory compression algorithms on field data.

- Error in queue length estimation: The reconstructed trajectories from all five trajectory compression algorithms were used for estimating queue lengths (in the number of vehicles) and queue locations (in feet) using the method proposed by Hao et al. (2015). For this particular comparison, this study used a dataset with 111 vehicle trajectories and related

signal timing information from a signalized intersection in Albany, New York. A detailed description of the test site and data can be found in Ban et al (2011). Note that both queue length and queue location are related to the maximum location of the back of the queue. Our objective here is to evaluate how the resampled data using these algorithms perform for traffic applications. Queue length estimation is just one of such applications and used here to illustrate the performance of the algorithms. There are three evaluation criteria for the queue length estimation application (Hao et al., 2015):

- Mean Absolute Error (MAE) of queue length Estimation (in number of vehicles)
- Mean Absolute Error (MAE) of queue location Estimation (in feet)
- Success Rate (in percentage)

Details of the evaluation criteria of queue estimation are given in Hao et al. (2015) and omitted here. We find that SAOTS (along with SAS and Sliding Window) excels in the Success Rate (81.82%) among all five algorithms. Compared with the other two online algorithms (Sliding Window and SWAB), SAOTS exhibits the lowest MAE in both queue length and queue location estimation (5.05 vehicles and 46.088 ft respectively).

Table 5.2. Comparison of Trajectory Compression Algorithms using field data.

TC Algorithm	**PDR (%)**	**Spatial Error (degree)**	***MAPE* of Speed (%)**	**Average Latency (s)**	**Queue Length Estimation**		
					***MAE* of Queue Location (ft)**	***MAE* of Queue Length (veh)**	**Success Rate**
SAS	58.94	0.0006	2.15	-	39.579	4.589	81.82%
SAOTS	**72.29**	*0.0003*	**4.3**	*29.73*	**46.088**	**5.047**	**81.82%**
Sliding Window	67.18	0.0008	*6.49*	**9.09**	*67.374*	*7.316*	**81.82%**
Bottom Up	72.06	0.0.004	14.11	-	36.373	4.155	78.18%
SWAB	*67.46*	**0.0002**	7.55	290.72	118.21	8.193	61.82%

It is clear from the results in Table 5.2 that the performances of the proposed SAOTS are either the best or the second-best among the three online algorithms. Furthermore, even for the criteria that SAOTS is the second-best (spatial error), the performances are somewhat close to the best performances. Arguably, the spatial error is the least important as long as the other performances such as speed MAPE, latency, and application-related performances are satisfactory. Therefore, the proposed SAOTS method is well balanced and considered as the best online algorithm among all evaluation criteria. The other two online algorithms suffer from degraded application performances (both the Sliding Window algorithm and the SWAB algorithm) and/or excessively long latency (the SWAB algorithm).

When comparing SAOTS with the other two batch algorithms (SAS and Bottom-Up), we can see that the PDR of SAOTS is better, while the other performances of SAOTS are close to those of the two batch algorithms. This further confirms that SAOTS is a well-balanced algorithm which

can run in near real-time with reasonably good performances in data reduction, latency, speed error, and real-world applications. Moreover, the ability to identify VFS of the segments is an additional feature in SAOTS, which is not available in previous algorithms with an exception of SAS. Overall, even compared with batch trajectory compression algorithms, we can find that SAOTS achieves a fine balance among PDR, spatial accuracy, latency, and application performances.

Figure 5.2 demonstrates a side by side comparison of resampling of a randomly selected vehicle trajectory using different trajectory compression algorithms. In this figure, the data points resampled using different trajectory compression algorithms are plotted with a distance offset. The

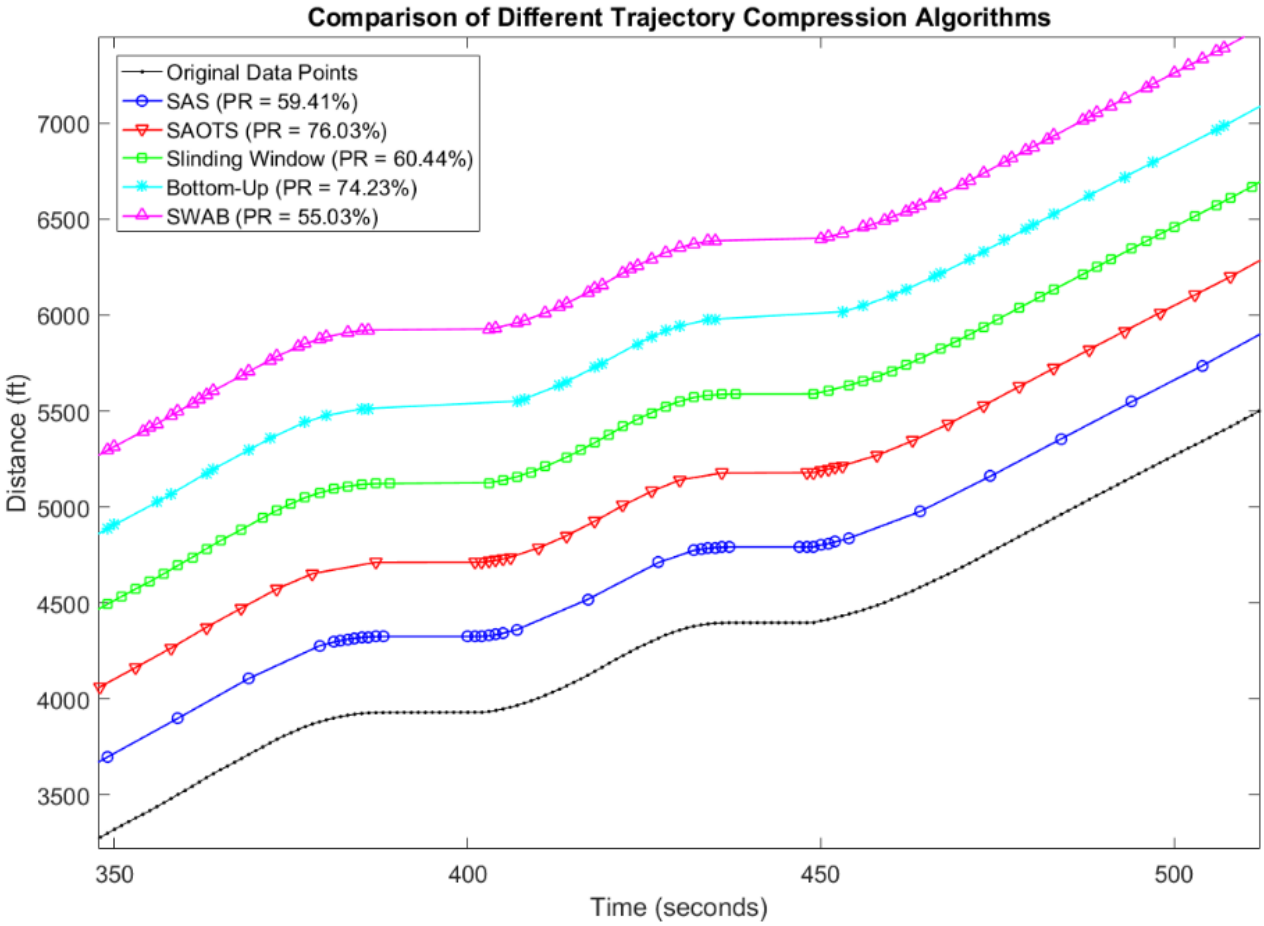


Figure 5.2. A comparison of resampling using different trajectory compression algorithms.

original data points are plotted in black, whereas other colors indicate the same trajectory, but resampled using different compression algorithms.

5.2. Performance of SAS and SAOTS on Didi Dataset

This section presents the performance of SAS and SAOTS on Didi dataset and compares it with other trajectory compression algorithms. The dataset contains trajectories of Didi's ridesourcing vehicles for the City of Xi'An in China for November 2017. Since the data size is huge, this study randomly selected vehicle trajectories from four days for analysis in this dissertation, representing about 15% of the total Didi dataset. The dataset used in this study contains nearly 60 million data point from about 0.25 million trips (trajectories). For this study, trajectories with a minimum of 100 data points were chosen to use, resulting in 0.21 million trajectories. Figure 5.3 (a) illustrates the distribution of the total number of data points in the trajectories used in our analysis. Distance-wise, the length of each trajectory varies between 0.1 miles to 9.07 miles (see figure 5.3 (b)). Notice that there is no fixed sampling interval for the Didi dataset. The sampling interval varies mostly between 1 to 10 seconds, with around 90% of data points sampled at an interval of 3 seconds. Furthermore, the raw Didi dataset does not have

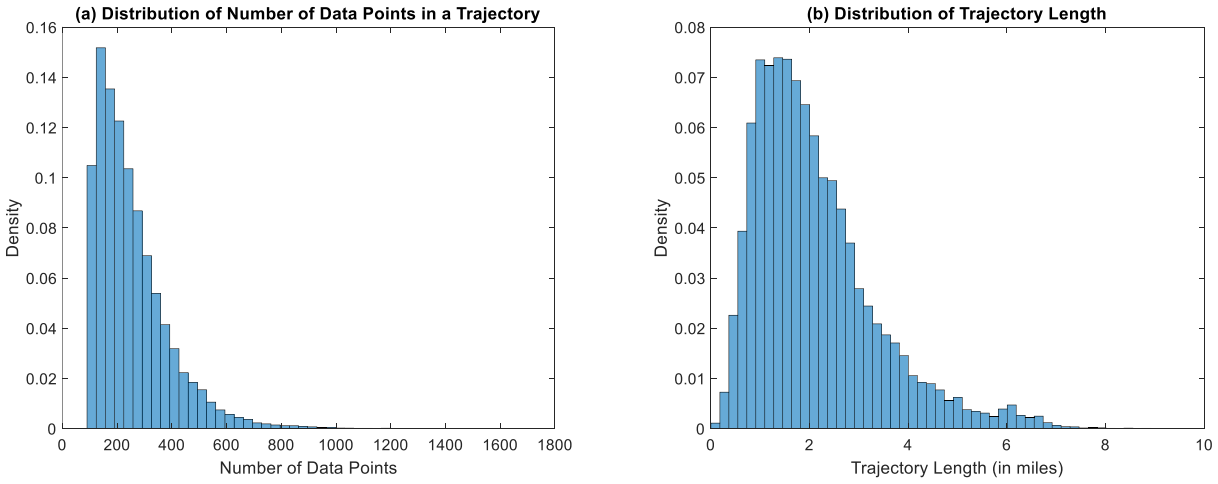


Figure 5.3. Distribution of number of data points and trajectory length in trajectories in Didi dataset.

instantaneous speed information. The change in location over each time step was used to estimate the speed. The estimation process is trivial and omitted here.

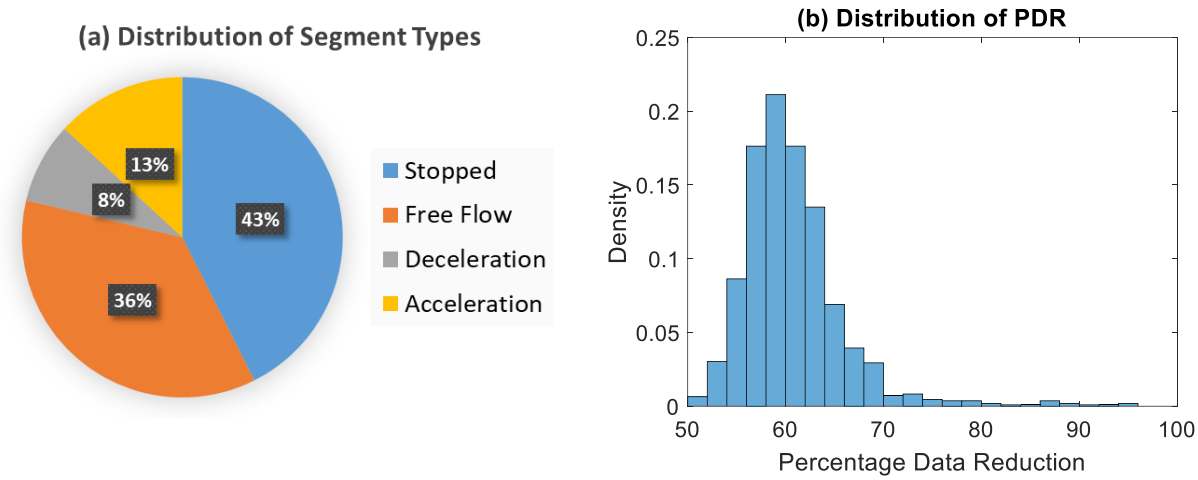


Figure 5.4. Distribution of segment types and PDR in Didi dataset.

Figure 5.4 (a) illustrates the distribution of different VFS segments estimated from the dataset. It is seen that around 43% of the segments are categorized as 'stopped', whereas 36% of the segments are classified as 'free-flow'. The percentages of "acceleration" and "deceleration" segments are relatively small. The PDR after applying the resampling method ranges mainly from 50% to 70% (see figure 5.4 (b)).

The same evaluation criteria as defined in section 5.1 is used to compare the trajectory compression algorithms, with the exception of queue length estimation errors. The reason for excluding the queue length estimation is the lack of signal timing information in the Didi dataset which is essential for such analysis (Hao et al., 2015). However, using Didi dataset, this study tested the performance of directly using the spectral analysis to select a resampling rate of the whole trajectory. It is important to note here that such spectral resampling method skips the segmentation part of SAOTS and focuses on the entire trajectory to find its critical frequency. This critical frequency is then used to find the resampling rate for the whole trajectory. In this way, the

spectral resampling can be considered as a batch compression method. The performance of SAOTS with respect to four other trajectory compression methods and spectral resampling is summarized in Table 5.3 and are discussed below. Note that for the three online algorithms (SAOTS, Sliding Window, and SWAB), similar as in Table 5.2, best performances are highlighted in bold text and the second-best performances are highlighted using italic text.

- Percentage Data Reduction (PDR): SAOTS has an average PDR of 64.66% for the trajectories in the Didi dataset, next to Sliding Window that has the highest PDR of around 91.25%. The distribution of PDR among different trajectories by applying the SAOTS algorithm is shown in figure 5.4 (b).

- Spatial Error: Among the five algorithms, SAS has the lowest spatial error of 0.0018. SAOTS performs the best among the three online algorithms with a spatial error of 0.0033

- $MAPE$ of Speeds: We find that SAS provides the lowest $MAPE$ (25.23%) for the resampled trajectories while compared with other trajectory compression algorithms. SAOTS has a MAPE of 30.91%. Similar to the Spatial Error, this is the best among all three online algorithms. In general, the MAPE of speeds is higher in the Didi dataset compared to the field dataset in Section 5.1. This is probably due to the fact that the sampling interval in Didi data is higher than that of the field data, and the instantaneous speed is estimated indirectly from the change in location over time.

- Latency: Figure 5.5 illustrates the latency distributions of three online algorithms when processing the Didi dataset. Similar to the field data, SAOTS has the highest number of data records sampled in a very low latency (1 second), resulting in an average latency of 14.12 seconds as shown in Table 5.3. Sliding Window has the second-lowest overall

latency, with an average of 20.92 seconds. SWAB showed a wide range of latencies, with an average latency of 86.52 seconds. In general, for all three online trajectory compression algorithms, the average latencies are lower in case of the Didi dataset when compared to the field data in the previous section. This is likely because in general, the trajectories in the Didi dataset have fewer data points than those in the field dataset.

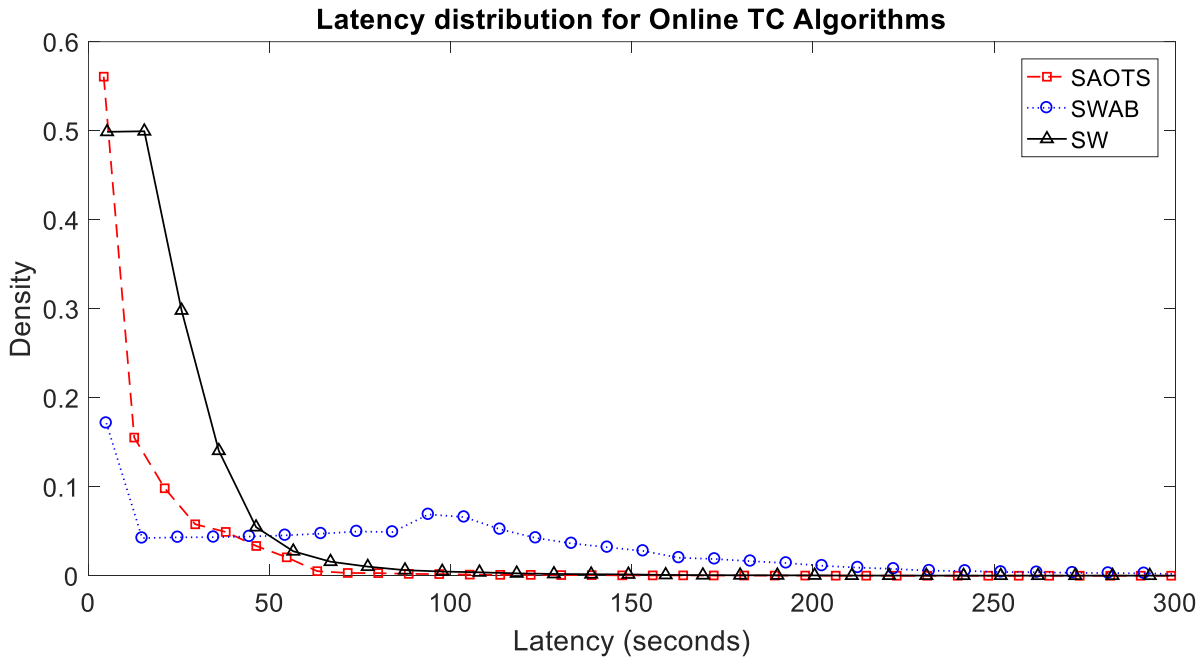


Figure 5.5. Latency distribution of online trajectory compression algorithms on Didi data

It was found that the spectral resampling yields decent performance, especially when compared with other batch trajectory compression algorithms. In fact, with a PDR of 69.16% and Spatial Error of 0.0019 degrees, the spectral resampling stands as second-best compression method among all trajectory compression methods considered in this study. Overall, we observed higher values for MAPE of speed while testing the algorithms using the Didi dataset. This is probably because the instantaneous speeds were not directly provided in the dataset; rather it was calculated by the change in location over time. This may result in inaccurate estimation of instantaneous

speeds, which may cause higher speed errors for the Didi dataset. The 127% MAPE was observed using the Sliding Window method, which reduces almost 91% of the original data points. This extremely high data reduction has most probably caused such high MAPEs. For the small dataset for which speeds are provided directly, as shown in Table 5.2, the speed error ranges from about 3% - 19% which is more reasonable.

Table 5.3. Comparison of Trajectory Compression Algorithms using field data.

TC Algorithm	**PDR (%)**	**Spatial Error (degree)**	***MAPE* Of Speed (%)**	**Average Latency (s)**
SAS	51.87	0.0018	25.23	-
Spectral Resampling	69.16	0.0019	39.70	-
SAOTS	*64..66*	**0.0033**	**30.91**	**14.12**
Sliding Window	**91.25**	0.0091	127.56	*20.92*
Bottom	58.41	0.0035	50.46	-
SWAB	61.13	*0.0047*	*73.30*	86.52

Chapter 6. SEMI SUPERVISED LEARNING METHOD FOR VFS ESTIMATION

6.1. SEMI-SUPERVISED LEARNING METHOD

This section explains a method to improve the HMM classifier using a semi-supervised approach, where unlabeled trajectories are added to the labeled training data. Using unlabeled data to help supervised learning has become an increasingly attractive methodology and proven to be effective in many applications such as text categorization and gene expression analysis where expert knowledge and costly biological experiments are often required to manually label the data. This situation is also common in GPS based vehicle trajectory data, where the huge amount of unlabeled trajectory contradicts the relatively few numbers of trajectories with experimentally estimated VFS. The intent of this study is to estimate the VFS of vehicle trajectory data with minimum human labeling efforts. The semi-supervised learning method used in this study is similar to those in Zhong (2005). Let's denote the labeled data by (x^l, y^l) and the unlabeled data by (x^u). The following steps can formulate the semi-supervised learning algorithm:

1. Use the completely labeled data (x^l, y^l) to train an initial model (θ)
2. Use θ to predict the labels (y^*) of the unlabeled data
3. Use the newly labeled data (x^u, y^*)along with the completely labeled ones to train a new model (θ^*)

The above-mentioned algorithm is also known as the self-training approach in some literature (Tamposis et al., 2018). Self-training is a wrapper method, which means in general, any classifier can be trained using this method. Since the classifier uses its own predictions to teach itself, it is

possible that a classification mistake can reinforce itself. A possible solution to this is by discarding the unlabeled sequences if the prediction confidence drops below a threshold. This solution is more restrictive since the optimal threshold needs to be identified but has the additional advantage of fewer computations since only a fraction of the unlabeled data is included in the training phase. However, this solution is not applicable in this study, as the HMM classifier does not provide a prediction confidence interval. Instead, a simple but crude method of including all predictions from step 2 is implemented in the training process.

Traditionally, the parameters of an HMM is calibrated using one sequence. In this case, the data consists of multiple individual vehicle trajectories. Each trajectory can be considered as a sequence in HMM. In order to accommodate multiple sequences, the instantaneous speeds of multiple trajectories are concatenated into one long sequence of data points. Before the concatenation, the trajectories were trimmed at stopped segments to prevent the abrupt change in the speed profile of the trajectory.

6.2. Numerical Studies

This section evaluates the performance of the HMM classifier trained using semi-supervised learning method. Then two case studies are presented to better understand the implications of this method in VFS estimation. The dataset used in this study is from a Chinese ridesourcing company, Didi Chuxing (https://outreach.didichuxing.com/research/opendata/en/). The dataset contains 758 vehicle trajectories of Didi's ridesourcing vehicles from the City of Chengdu in China for November 2017. From these trajectories, 54 trajectories were randomly selected, and the VFS segments in these trajectories were manually labeled using human judgment. Table 6.1 illustrates

the number of trajectories, the number of VFS segments and the number of data points in the labeled and unlabeled data used in this study.

Table 6.1. Datasets used to train the semi-supervised HMM

	Number of trajectories	**Number of VFS Segments**	**Number of data points**
Labeled data	54	769	10,417
Unlabeled data	704	10,911	165,215

The accuracy of the regular and semi-supervised HMM classifiers were compared for different sizes of training data. In order to train the regular HMM classifier, the number of labeled trajectories was varied from 1 to 50 with an increment of 10. For the semi-supervised HMM classifier, just 1 labeled trajectory was used, and an increasing number of unlabeled trajectories were gradually incorporated. Figure 6.1 illustrates the change in the accuracy of the classifiers with the increase in training data. It was found that, when compared to human judgment, the best classifier in terms of accuracy (90.5%) was obtained when 31 labeled trajectories were used to train the HMM parameters. The performance of semi-supervised HMM classifier increased with the increase of unlabeled data. It was found that the best performance was found when all 700 unlabeled trajectories were used. From this experiment, it can be concluded that the semi-supervised learning method can achieve a very similar accuracy using a very small amount of labeled data when compared to regular HMM which relies on a large number of labeled data.

In order to better understand the difference in nature of these two classifiers, this study looked at their predictions on individual trajectories, side by side. It was found that, when compared to the regular HMM classifier, the semi-supervised HMM classifier had an accuracy of 96.3%. Table 6.2 illustrates the confusion matrix. The prediction results of two classifiers are demonstrated using a sample trajectory in figure 6.2. It can be seen that, in general, the models developed using semi-supervised method seem to be more "precise" in VFS estimation than the regular model. The semi-supervised model tends to distinguish smaller acceleration and deceleration segments where the regular HMM model considers them as one long segment.

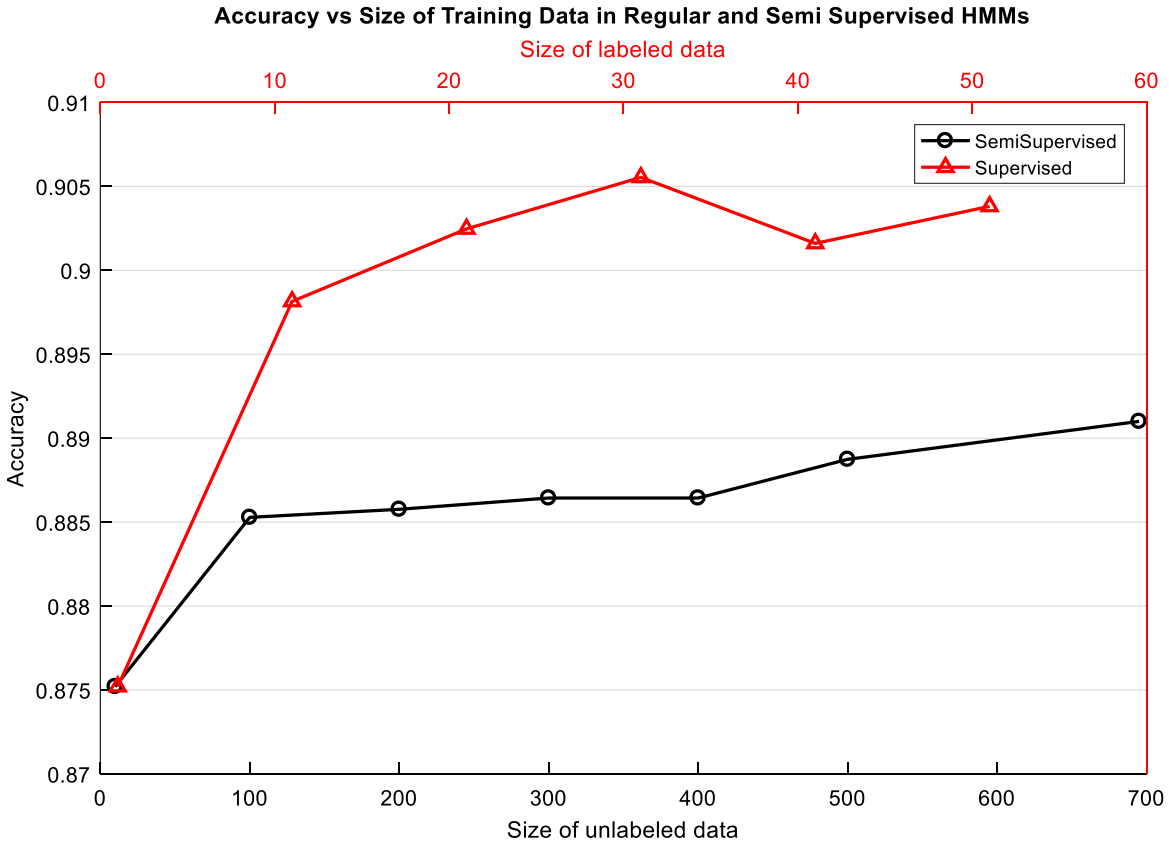


Figure 6.1. Accuracy vs size of training data for regular and semi supervised HMMs

Table 6.2. Confusion Matrix for Semi-Supervised HMM.

		Predicted by semi-supervised HMM			
		Stopped	**Free Flow**	**Acceleration**	**Deceleration**
Predicted by regular HMM	**Stopped**	287	0	5	2
	Free Flow	1	277	11	35
	Acceleration	2	0	36	2
	Deceleration	4	6	0	288

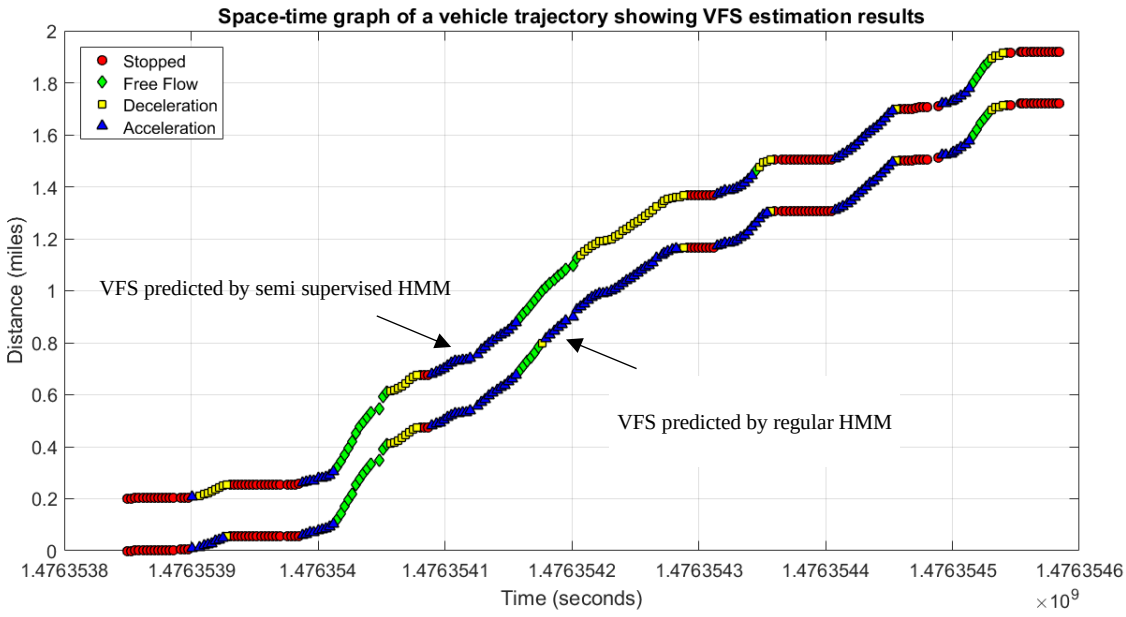


Figure 6.2. Comparison of VFS estimation results by the regular HMM and the semi supervised HMM

Chapter 7. CONCLUSION

This research develops two self-adaptive sampling methods for vehicle trajectory data compression. The methods target mobile sensors that can provide highly accurate locational information (such as those equipped with GPS devices) of vehicles. The proposed methods are applied to the detailed trajectory of an individual vehicle separately and independently. Both methods discussed in this dissertation utilize the VFS of the sensor/vehicle (e.g., free-flowing, accelerating, etc.) from its trajectory data, and uses the estimated VFS to adjust the sampling rate of the trajectory accordingly. It is found that using the proposed data compression strategies, the total amount of data can be reduced by about 50%-70% for the datasets tested in this study while keeping the most critical data points. This is helpful, as the reduced data can yield much less communication cost, energy consumption, and data storage. It is also seen that the reduced data can not only be used satisfactorily for traffic modeling applications, such as queue location and queue length estimation but also be effective in protecting user's privacy.

The practical applications of this dissertation research are summarized as follows:

- When integrated into the client-side devices (such as smartphones and CAVs), reduction of GPS based location data using the proposed methods will help reduce data size, which will be tremendously helpful in reducing the storage and communication cost in the client side. With the proposed resampling methods integrated, smartphone applications and location-based services like Google Maps, Yelp and Gas Buddy will benefit from reduced storage and communication cost at the client side. The reduced cost in the data collection process will eventually promote the crowdsourcing of mobile data.

- The proposed method of identifying VFS and ‘stop and go’ can be used to better understand and explain the intrinsic features of a probe vehicle’s motion. Such additional information extracted from multiple individual vehicle trajectories may be combined to achieve a clearer picture on the performance of underlying road network and its bottlenecks.

- At the server side, vehicle trajectory data that have already been collected can be compressed using the proposed SAS strategy to free-up storage spaces. Without severely compromising the performance, the resampled data by the proposed method can be used in various transportation applications. One of such applications is modeling the human mobility pattern and origin-destination (OD) estimation. In recent years, OD estimation using GPS data is gaining traction among researchers. The compressed data using methods proposed in this dissertation can be used in this case since the origin and the destination of the trajectories are preserved. For transportation applications where detailed and fine-grained trajectory information is desired (such as queue length estimation), the compressed data using proposed methods still provides reasonable performance as discussed in section 3.6.

- Last but not least, apart from the vehicle trajectory data, the proposed methods can also be applied to other types of location data, such as fitness tracking. Fitness tracking devices like Fitbit collects data from numerous sensors and then sends this information to the cloud server for processing. The proposed methods for VFS estimation and resampling may be modified for this use case.

With the rapid technology advancement and data explosion in transportation and related fields, how to properly compress (i.e., collect and resample) vehicle trajectory data presents a critical challenge. This dissertation presents some initial yet promising methods for vehicle

trajectory compression based on its Vehicle Flow State, which could motivate more research efforts in this area. Future research should further refine/expand SAS and SAOTS to improve its performances especially PDR, latency, and speed accuracy, which are crucial for online applications. More testing/comparison of the proposed algorithms should also be conducted in future research using more real-world vehicle trajectory data. In addition, the collected data may contain errors (e.g., GPS data have a few meters of location error in normal situations), which may influence the vehicle flow state estimation. This issue should be investigated in future research. The methods proposed in this study does not use elevation data from GPS sensors. Elevation data can provide means to better understand the vehicular motion and incorporating this information can increase the performance and accuracy of the model. After the algorithms are properly refined and tested, it may be integrated into real-world data collection applications to make a real impact. Furthermore, this research only applied queue length estimation as the target transportation applications to test and compare different trajectory compression algorithms. Although queue length estimation is an important urban traffic application, in the future, more relevant traffic/transportation applications should be used to test the proposed algorithms to obtain a more balanced view/testing of the algorithm.

There are recent conversations on “data tax” that charges companies for using people’s data. For example, recently the French government has released a draft bill on the French Digital Service Tax. According to this bill, a 3% tax will apply to the revenue generated from online intermediation services and the sale of targeted digital advertising in France. In other words, the tax is designed for big tech companies like Facebook, Google, and Amazon who rely on user data for business. According to a study by Deloitte, only 5% of the digital tax’s burden will fall on the large internet companies it aims to target. Instead, the study said consumers will absorb 55% of the cost and 40%

will be borne by businesses that use digital platforms (Pellefigue, 2019). If this applies to crowdsource mobile data collection (e.g., the way Google collects and uses data), it is expected that data collection should be done more intelligently on the client (or end-user) side who contributes the data. Companies will be more motivated to implement certain type of data compression methods to reduce the size of the data they receive from each client. The proposed vehicle trajectory compression concepts and methods in this dissertation research are expected to provide useful insight if this happens in the future.

References

Balzano, W., & Del Sorbo, M. R. (2014, September). Setra: A smart framework for GPS trajectories' segmentation. In Intelligent Networking and Collaborative Systems (INCoS), 2014 International Conference on (pp. 362-368). IEEE.

Ban, X., and Gruteser, M., 2012. Towards fine-grained urban traffic knowledge extraction using mobile sensing. In Proceedings of the ACM SIGKDD International Workshop on Urban Computing, pages 111-117.

Ban, X., Hao, P., and Sun, Z (2011). Real time queue length estimation for signalized intersections using sample travel times from mobile sensors. Transportation Research Part C, 19(6), 1133-1156

Baum, L. E., & Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. The annals of mathematical statistics, 37(6), 1554-1563.

Birnbaum, J., Meng, H. C., Hwang, J. H., & Lawson, C. (2013, July). Similarity-based compression of GPS trajectory data. In 2013 Fourth International Conference on Computing for Geospatial Research and Application (pp. 92-95). IEEE.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. Data mining and knowledge discovery, 2(2), 121-167.

Cambridge Systematics. Inc. (2007). NGSIM Peachtree Street (Atlanta) Data Analysis.

Chen, C., Ban, X., Wang, F, Wang, J., Siddique, N., Fan, R., Lee, J. (2017). Understanding GPS and Mobile Phone Data for Origin-Destination Analysis. Final Report Submitted to Federal Highway Administrations (FHWA).

Chen, M., Xu, M., & Franti, P. (2012). A fast $ o (n) $ multiresolution polygonal approximation algorithm for GPS trajectory simplification. IEEE Transactions on Image Processing, 21(5), 2770-2785.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297.

Crackberry. (2019). Retrieved 28 August 2019, from https://crackberry.com/real-time-traffic-information-google-maps

Cristianini, N., & Shawe-Taylor, J. (2000). An introduction to support vector machines and other kernel-based learning methods. Cambridge university press.

Douglas, D. H., & Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: the international journal for geographic information and geovisualization, 10(2), 112-122.

Durbin, R., Eddy, S. R., Krogh, A., & Mitchison, G. (1998). Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge university press.

Hao, P., Ban, X., & Whon Yu, J. (2015). Kinematic equation-based vehicle queue location estimation method for signalized intersections using mobile sensor data. Journal of Intelligent Transportation Systems, 19(3), 256-272.

Herrera, J. C., Work, D. B., Herring, R., Ban, X. J., Jacobson, Q., & Bayen, A. M. (2010). Evaluation of traffic data obtained via GPS-enabled mobile phones: The Mobile Century field experiment. Transportation Research Part C: Emerging Technologies, 18(4), 568-583.

Keogh, E., Chu, S., Hart, D., & Pazzani, M. (2001, November). An online algorithm for segmenting time series. In Proceedings 2001 IEEE International Conference on Data Mining (pp. 289-296). IEEE.

Lauer, F., & Bloch, G. (2008). Incorporating prior knowledge in support vector machines for classification: A review. Neurocomputing, 71(7-9), 1578-1594.

Lawson, C. T., Ravi, S. S., & Hwang, J. H. (2011). Compression and mining of GPS trace data: new techniques and applications. Technical Report Region II University Transportation Research Center.

Lee, W. C., & Krumm, J. (2011). Trajectory preprocessing. In Computing with spatial trajectories (pp. 3-33). Springer, New York, NY.

Lim, S. H., Chia, Y. K., & Wynter, L. (2017). Accurate and Cost-Effective Traffic Information Acquisition using Adaptive Sampling: Centralized and V2V Schemes. Transportation Research Procedia, 23, 61-80.

Liu, H., & Jabari, S. (2008). Evaluation of Corridor Traffic Management and Planning Strategies That Use Microsimulation: A Case Study. Transportation Research Record: Journal of the Transportation Research Board, (2088), 26-35.

Long, C., Wong, R. C. W., & Jagadish, H. V. (2013). Direction-preserving trajectory simplification. Proceedings of the VLDB Endowment, 6(10), 949-960.

Lovrić, M., Milanović, M. and Stamenković, M., 2014. Algoritmic methods for segmentation of time series: An overview. Journal of Contemporary Economic and Business Issues, 1(1), pp.31-53.

McMaster, R. B. (1986). A statistical analysis of mathematical measures for linear simplification. The American Cartographer, 13(2), 103-116.

Meratnia, N., & Rolf, A. (2004, March). Spatiotemporal compression techniques for moving point objects. In International Conference on Extending Database Technology(pp. 765-782). Springer, Berlin, Heidelberg.

Muckell, J., Hwang, J. H., Patil, V., Lawson, C. T., Ping, F., & Ravi, S. S. (2011, May). SQUISH: an online approach for GPS trajectory compression. In Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications (p. 13). ACM.

Muckell, J., Olsen, P. W., Hwang, J. H., Lawson, C. T., & Ravi, S. S. (2014). Compression of trajectory data: a comprehensive evaluation and new approach. GeoInformatica, 18(3), 435-460.

Nibali, A., & He, Z. (2015). Trajic: An effective compression system for trajectory data. IEEE Transactions on Knowledge and Data Engineering, 27(11), 3138-3151.

Paek, J., Kim, J., & Govindan, R. (2010, June). Energy-efficient rate-adaptive GPS-based positioning for smartphones. In Proceedings of the 8th international conference on Mobile systems, applications, and services (pp. 299-314). ACM.

Park, S., Lee, D., & Chu, W. W. (1999, November). Fast retrieval of similar subsequences in long sequence databases. In Proceedings 1999 Workshop on Knowledge and Data Engineering Exchange (KDEX'99)(Cat. No. PR00453) (pp. 60-67). IEEE.

Pellefigue, J. (2019). Retrieved 30 August 2019, from https://taj-strategie.fr/content/uploads/2020/03/dst-impact-assessment-march-2019.pdf

Prior-Jones M (2008) Satellite communications systems buyer's guide. British Antarctic Survey

Quartz. (2019). Retrieved 30 August 2019, from https://qz.com/344466/connected-cars-will-send-25-gigabytes-of-data-to-the-cloud-every-hour/

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE, 77(2), 257-286.

Shatkay, H., & Zdonik, S. B. (1996, February). Approximate queries and representations for large data sequences. In Proceedings of the Twelfth International Conference on Data Engineering (pp. 536-545). IEEE.

Wan, J., Liu, J., Shao, Z., Vasilakos, A. V., Imran, M., & Zhou, K. (2016). Mobile crowd sensing for traffic prediction in internet of vehicles. Sensors, 16(1), 88.

Wang, Y., Lin, J., Annavaram, M., Jacobson, Q. A., Hong, J., Krishnamachari, B., & Sadeh, N. (2009, June). A framework of energy efficient mobile sensing for automatic user state recognition. In Proceedings of the 7th international conference on Mobile systems, applications, and services (pp. 179-192). ACM.

Yeo, H., & Skabardonis, A. (2009). Understanding stop-and-go traffic in view of asymmetric traffic theory. In Transportation and Traffic Theory 2009: Golden Jubilee (pp. 99-115). Springer US.

Zheng, Y. (2015). Trajectory data mining: an overview. ACM Transactions on Intelligent Systems and Technology (TIST), 6(3), 29.