The University of Hong Kong

Faculty of Engineering

Department of Computer Science

COMP7704
Dissertation Title
A Large Incident Management System (IMS) for Intelligent City Development

Submitted in partial fulfillment of the
requirements for the admission to the degree of
Master of Science in Computer Science

By
Andres Namm
HKU student no. 30351.5541.584

Supervisor's title and name:
Dr Reynold Cheng
Date of submission: 01/11/2017

Contents

1 Introduction

The goal of this research is to develop and apply an automatic real time incident detection system and test out a current state-of-the art traffic incident detection method in Hong Kong using Probe Car GPS observations and Speed Panel data.

More concretely, this dissertation pursues to further develop incident detection in Hong Kong by:

- Designing a framework which performs the preprocessing steps necessary to prepare speed panel data and GPS data to build a model for incident detection.
- Implementing Latent Dirichlet Allocation (LDA) equivalent Incident Detection method using topic models and analyzing the results.

Accurate incident detection is important mostly for two reasons: routing and Acknowledging. For routing algorithms, incident detection knowledge gives ground to a better estimate of travel times for different routes. Automatic incident detection can as well be useful for informing important agents like the police or traffic authority about the incident so they could take necessary action faster [18].

Failures in preprocessing the geospatial data can significantly affect the end result. Due to complexities of the transformations that are required to be performed on the data, it is very easy to make mistakes. Even if no logical errors are made when preprocessing the data, the geoprocessing algorithms and methodologies, if not chosen carefully, eliminate the possibility to perform incident detection well. Based on this, the first part of this dissertation focuses on developing a well-documented incident detection data preprocessing pipeline. This means the generation of a platform where data required for detection of incidents is stored and transformed in a systematic manner so it could be used for building models necessary for incident detection. Because a lot of incident detection algorithms share similar preprocessing steps, this part aims to simplify that process as much as possible. This phase starts by describing the process of choosing the right map format to use as a basis for storing necessary information for incident detection. After that, it describes how the chosen map is transformed through segmentation into the right form so geospatial data like accidents, GPS records and speed panel recordings could be matched with it in an appropriate way for the incident detection algorithm. The dissertation then goes on with the process of map matching for GPS record data and accident data. The matched GPS data is directed to the interpolation phase which allows the generation of new records and also the elimination of inconsistent records. The result of linear interpolation is integrated with speed panel data. The integration of speed panel and GPS data is itself a large topic with multiple steps and takes a large part under this dissertation. After the accidents, GPS records and speed panels have been matched to the map, labelling of the data is performed as the last step before starting the model training. To make the preprocessing of the data more reliable and also more informative, several tests and visual analyses are developed.

As this dissertation bases itself on previous work done in Hong Kong University [13], refactoring, redesigning and debugging is done on the already existing code.

After generation of the incident detection preprocessing pipeline, topic modeling based method [6] [18] is going to be tested, developed and analysed for Hong Kong traffic. This algorithm is implemented to use the results of the preprocessing steps pipeline as an input for training. The incident detection mathematical model used is the same as in Kinoshita et al. [18], however that paper only tested its model on GPS record data. This dissertation integrates speed panel data to the model as well.

2 Methodology

This dissertation falls into the category of developmental and experimental research. First, a platform is developed for incident detection in Hong Kong, implementing well-documented and modular preprocessing of the data and then training an incident detection model based on LDA, a state-of-the-art algorithm. The dissertation then proceeds to analyse this algorithm and compare its results with different setups in regard to data and parameters, and then compare the results with the results of other authors.

3 Literature Review

3.1 Difference in Data Sources

Currently, it can be said that real time traffic incident detection methods divide into groups based on multiple attributes. There exists a split between works which use GPS Probe Car Data (PCD) for building their models, and works which base their analysis on Road Sensors (RS). In this segment, Road Sensors mean sensors installed aside a road, which collect data mainly about the speed and volume of the traffic. Some methods, however, although having been tested on one data source, are claimed to be applicable to both PCD and RS data [18]. A lot of work has been done in regard to methods that use RS as a data source [18] [17]. RS gains information from each observation that travels through the road segment the RS has been installed on; thus the information provided is of high accuracy. There are two main types of RS: segment level RS and sensor level RS. Sensor level RS collects all the observations and stores them in a certain database whereas segment level sensors aggregate the observations and store an average value. The major weakness of RS is that installing them requires a lot of resources [37] [41] [38]. PCD data is much more easily accessible and a lot of public transport already uses GPS tracking. Although PCD cannot monitor all cars, it can follow specific car patterns [18]. Each car can provide data about all the roads it travels on; in addition, this data is formed sequentially on the path the car travels on, so information on neighboring locations can be also taken into account. However, the problem here is low sampling, i.e. scarcity of data. Some authors claim that this characteristic inevitably requires additional data sources like social media to be integrated intor the model of prediction [37]. However, in literature, there exist plenty of methods which analyse PCD data as a standalone entity, e.g. [41] [18] [28] [19] [14].

3.2 Difference in Methodology

To the author's knowledge, there is a slightly greater focus on unsupervised modeling methods in order to save time and cost compared to manually generating labels for traffic modelling. Manual labelling for traffic seems somewhat counterintuitive as well due to the nature of the data that traffic produces [18] [17] [41] [9] [21].

The drawbacks of such methods are very general. Like with all unsupervised learning methods, it is very complicated to actually check whether results are correct or not.

Supervised methods mostly use neural networks to perform classification. [31] [16] [20]

There exists a great number of methodologies for modelling traffic. The level of complexity varies considerably as well. The simplest methods perform just the detection of speed reduction and have specifc assumptions for features that occur when an accident has happened [41] [17].

However, this method is less readily applicable to local streets where the many crossings and traffic lights can cause cars to stop more frequently under normal circumstances. Furthermore, bottlenecks on freeways can also cause spontaneous congestion which happens when the traffic flow exceeds the expected volume. It is important to make a difference between spontaneous and abnormal congestion which is caused by a traffic accident [18].

To solve this issue, work has been focused on Hidden Markov Models and Topic Models to model the latent traffic states behind each PCD observation without explicitly assuming what are the features of a traffic state which relate to a traffic accident [18] [28] [19] [14].

Latent Dirichlet Allocation (LDA), which is classified as a topic modelLing technique learns the traffic speed model for each road segment, then calculates the divergence between the current traffic state and the model (usual traffic state) to detect incidents. The accuracy was higher on Tokyo highways [18] compared to the speed reduction based method mentioned earlier [41].

According to the initial experiments performed in Hong Kong before, LDA did not achieved the same level of accuracy. This is likely due to the lower sampling rate and a very heterogeneous dataset in Hong Kong compared to the data collected from Tokyo highways. As the proposed method is very generic, there are, however, multiple ways to fine tune it for a specific city [18] and therefore improve the predictions. This method can also be adapted for RS data, which is going to be incorporated to further experiments.

Liao et al. [20] proposed a method for detecting anomalies by modelling taxi probe data using conditional random fields (CRF). This work actually focuses on human-computer interaction, where computer detects an anomaly and a human has to classify it. Considering the Hong Kong data, a system like that could have a number of good uses. Hong Kong has an abundance of speed cameras available as well, which could be used to quickly identify whether the classification is accurate or not.

Many authors have tried to solve the scarcity issue of PCD based methods by including additional data into the model, mostly focusing on incorporating social media [37] [38] [40] [7]. Incorporating multiple data sources has shown to improve the prediction rate, but it also implies more specific analysis of each city the method is applied to in regard to the potential data sources. In Hong Kong, there is a lot of potential in using additional data sources like social media, points of interest, Road Sensors [11] and more.

Summarising the current works, one can say that ample work has been done on both PCD and RS data. However all these methods share a similar characteristic: they have applied and experimented with their methods based on the data available on some specifc locations. This is very intuitive considering that solving trafficc-related problems is a very localized task. For Hong Kong there are a lot of possibilities available to take advantage of the current works. However, specific adaption considering Hong Kong traffic and the nature of the data is required. Based on multiple considerations, like available data, complexity of application and claimed results, this dissertation takes a serious look into implementing a traffic incident detection method based on Kinoshita et. al[18]. Considering, however that different articles have implemented the proposed state-of-the-art algorithms on different datasets like Chicago, Tokyo, Beijing and other locations, then the futher aim is to test out multiple methods in Hong Kong to add valuable experiments to current works based on a new dataset and also find the most suitable method for Hong Kong. To make that process easier, guidelines and framework for preprocessing the geospatial data are developed.

4 Incident Detection Data Preprocessing Pipeline

4.1 General

This section of the thesis aims to present:

- The redesigned source code for preprocessing the data
- The general principles for preprocessing the data
- The automatic and visual-based tests that were implemented to test the correctness and quality of the preprocessing steps.

4.1.1 Motivation

According to multiple authors and data scientists, preprocessing of the data can take up to 80 percent of the time spent on a machine learning project [22] [27]. In that sense, this project is no different. Already due to natural uncertainty of the data GPS devices produce, the taxi dataset can by itself produce a lot of errors. The author of this dissertation strongly agrees with this claim based on the experience received with this project.

Luckily the analysis and preprocessing steps are very similar for multiple different incident detection algorithms [40] [38] [18]. This means a lot of time could be saved by building a project which does the preprocessing of the data before the model training in a modular and well-documented way. Based on this, a decision was made to redesign this project and create general guidelines for running through each of the preprocessing steps. Both of the above-mentioned steps aim to build a system which could be reused by multiple researchers to test and analyse various incident detection algorithms without investing too much time in understanding the technicalities of preprocessing.

Failures in preprocessing the data can affect the end result significantly. Due to the complexity of the transformations that are required to be performed on the data it is very easy for errors to occur. Even if no logical mistakes are made when preprocessing the data, the geoprocessing algorithms and methodologies, if not chosen carefully, eliminate the possibility of performing incident detection well. Due to the importance of the preprocessing phase, general guidelines have been written with thorough documentation on how preprocessing of the data is performed and what are the built tests and analyses carried out in this phase. In this way, both the dissertation and the dissertation webpage can be used as support material for the source code of this project.

4.2 Datasets

4.2.1 Collection of Data in Hong Kong for this Project

With many of machine learning problems what matters the most is the amount of data one has for building their models. This is due to the fact that machine learning is overall prone to overfitting even after very careful development of the model [32]. Larger datasets allow this overfitting to gradually decrease [23]. This is due to the fact that more data makes it virtually impossible to design a function which responds to all the data, especially random noise based outliers. This principle can be shown on a learning curve where the training set error gradually grows whereas the cross validation set error gradually declines as the dataset size gets bigger.

[23]

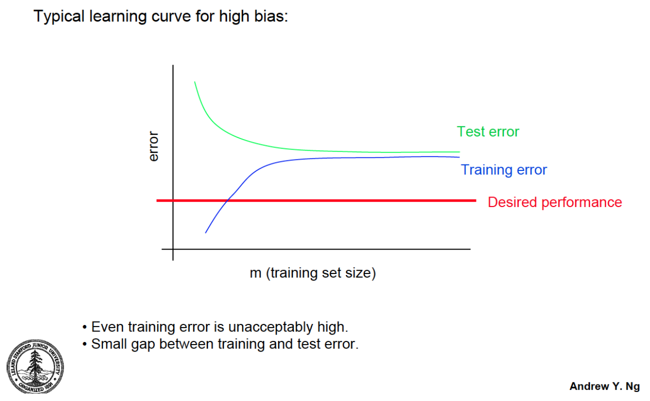


Figure 1: Learning curve example obtained from Coursera ML Course

Incident Detection algorithms suffer from general machine learning issues but the dataset size also influences whether at accident time there exists a sufficient amount of records in the specific segment. This means that the data acquisition process is a very important step in the development of an incident detection model, because more data gives ground to better testing possibilities for the model.

In the beginning of the project two datasets were available: the taxi dataset, which consisted of GPS data for 445 local taxis collected about every 35 seconds in 2010, and the accident dataset, which stored 14,943 traffic incidents that happened in Hong Kong in 2010 with location information. To obtain a better picture of the available data in Hong Kong, focus was put on three sources.

Textual information obtained from social media has been used successfully in combination with GPS records in recent research to detect traffic accidents [38] [40]. Based on the the article by Wang et al. [38] Twitter was considered as a potential data source for accident information but due to lack of

Hong Kong based traffic information agents and overall smaller Twitter usage in Hong Kong there did not seem to be any realistic use for it. In Facebook, a couple of groups that shared traffic information were found, however, the information there was sparse and very unstructured. From other sources mainly two textual datasets were found:

- Hong Kong Commercial Radio provided traffic news webpage (traditional Chinese) [29]
- Transportation Department provided special traffic news (English,traditional Chinese, Mandarin) [10]

In the end, the author of this dissertation decided not to prioritise the integration of social media data into this project because of the relatively limited size of the social media datasets. The author does not, however, exclude the possibility and usefulness of using social media in further experiments.

Government provided datasets proved the most useful for this project as Hong Kong is quite keen on taking advantage of digital solutions to advance its functionality. There are multiple applications in Hong Kong accessible on the government website [26]. Hong Kong has also created a webpage where it keeps most of its available public data [24]. The webpage includes a traffic speed map dataset [25] which collects all the observations from speed sensors in Hong Kong. As it is available only from 2013, Hong Kong Transport Department was contacted to request data from 2010. Although most of the public data is easily accessible online, it was not possible to get access to the Hong Kong roadworks database because of security reasons.

Private companies seem to be a little less willing to share their data. Hong Kong Taxi has provided the taxi dataset consisting of GPS observations collected in the year 2010 but enquiries to companies like Google and Hong Kong Citybus for traffic information did not carry any results. In addition,in 2017 Hong Kong Taxi company chose to make the taxi GPS dataset available for purchase. It is likely the negative responses from private entities are rather a rule than an exception. This is because data buying and selling has grown significanly with the raise of the Big Data Paradigm in the recent decade [4].

4.2.2 Description of Datasets Used in this Project

GPS dataset

- Dataset Name : Hong Kong taxi GPS data in 2010
- Data Source : Hong Kong Taxi Company
- Size - 30GB

Description GPS data for 445 local taxis about every 35 seconds in 2010 Altogether there are 314,190,268 rows of information in this dataset.Each row stands for an observation collected from a specific taxi at a specific time. All of the data is divided into 52 .mdb files each of them containing all the data for one week. Example extract of this can be shown in Table 1

Headers from <week_nr>.mdb

- HkDt: Date
- HkTm: Time
- ACC: Unspecified
- FlagDown: Unspecified
- PosID: Unspecified
- DevID: GPS recorder ID
- SpeedKmHr: Speed the taxi was travelling in km/h
- Lat: Latitude coordinates in WGS84 geometrical system
- Lon: Longitude coordinates in WGS84 geometrical system
- Direction: Direction in radian

HkDt	HkTm	ACC	FlagDown	PosID	DevID	Lat	Lon	SpeedKmHr	Direction
01.01.2010 00:00:00	30.12.1899 00:26:16	1	0	218231348	1036	2,23253403e+01	1,14169060e+02	5,00000000e-01	1,15370003e+02
01.01.2010 00:00:00	30.12.1899 00:26:16	1	0	218231378	1036	2,23253403e+01	1,14169060e+02	5,00000000e-01	1,15370003e+02
01.01.2010 00:00:00	30.12.1899 00:26:55	1	0	218231834	1036	2,23253479e+01	1,14169014e+02	4,00000006e-01	6,07500000e+01
01.01.2010 00:00:00	30.12.1899 00:27:28	1	0	218232265	1036	2,23253384e+01	1,14168983e+02	4,00000006e-01	2,25869995e+02
01.01.2010 00:00:00	30.12.1899 00:28:03	1	0	218232687	1036	2,23253956e+01	1,14169006e+02	4,00000006e-01	3,20699997e+01
01.01.2010 00:00:00	30.12.1899 00:28:36	1	0	218233112	1036	2,23254223e+01	1,14169067e+02	8,39999962e+00	3,48130005e+02
01.01.2010 00:00:00	30.12.1899 00:29:10	1	0	218233530	1036	2,23256226e+01	1,14169083e+02	8,00000012e-01	1,67700005e+01
01.01.2010 00:00:00	30.12.1899 00:29:43	1	0	218233961	1036	2,23263226e+01	1,14169037e+02	2,00000003e-01	8,37699966e+01
01.01.2010 00:00:00	30.12.1899 00:30:18	1	0	218234390	1036	2,23263454e+01	1,14169052e+02	2,00000003e-01	8,05400009e+01
01.01.2010 00:00:00	30.12.1899 00:30:53	1	0	218234840	1036	2,23261776e+01	1,14169769e+02	1,00000001e-01	1,50259995e+02
01.01.2010 00:00:00	30.12.1899 00:31:27	1	0	218235263	1036	2,23254414e+01	1,14169853e+02	3,90000010e+00	1,82779999e+02
01.01.2010 00:00:00	30.12.1899 00:32:00	1	0	218235667	1036	2,23247452e+01	1,14170013e+02	1,60000002e+00	1,86470001e+02
01.01.2010 00:00:00	30.12.1899 00:32:34	1	0	218236148	1036	2,23243523e+01	1,14169952e+02	9,00000000e+00	2,33729996e+02
01.01.2010 00:00:00	30.12.1899 00:33:09	1	0	218236548	1036	2,23247204e+01	1,14169357e+02	2,00000003e-01	1,66550003e+02
01.01.2010 00:00:00	30.12.1899 00:33:42	1	0	218236958	1036	2,23246384e+01	1,14169189e+02	1,00000001e-01	2,62800007e+01
01.01.2010 00:00:00	30.12.1899 00:34:16	1	0	218237386	1036	2,23246136e+01	1,14169121e+02	1,00000001e-01	1,76089996e+02
01.01.2010 00:00:00	30.12.1899 00:34:50	1	0	218237807	1036	2,23245773e+01	1,14169022e+02	1,00000001e-01	2,29080002e+02
01.01.2010 00:00:00	30.12.1899 00:35:24	1	0	218238248	1036	2,23245296e+01	1,14168877e+02	3,00000012e-01	7,08000031e+01
01.01.2010 00:00:00	30.12.1899 00:35:59	1	0	218238681	1036	2,23254318e+01	1,14169052e+02	1,10000002e+00	3,01999998e+00
01.01.2010 00:00:00	30.12.1899 00:36:33	1	0	218239101	1036	2,23254871e+01	1,14169060e+02	6,99999988e-01	4,87299995e+01
01.01.2010 00:00:00	30.12.1899 00:37:06	1	0	218239531	1036	2,23255444e+01	1,14169106e+02	6,00000024e-01	5,66899986e+01
01.01.2010 00:00:00	30.12.1899 00:37:40	1	0	218239944	1036	2,23256283e+01	1,14169182e+02	6,00000024e-01	5,96500015e+01
01.01.2010 00:00:00	30.12.1899 00:38:14	1	0	218240378	1036	2,23256664e+01	1,14169205e+02	1,00000001e-01	1,00779999e+02
01.01.2010 00:00:00	30.12.1899 00:38:47	1	0	218240799	1036	2,23256721e+01	1,14169189e+02	2,00000003e-01	7,16100006e+01
01.01.2010 00:00:00	30.12.1899 00:39:21	1	0	218241227	1036	2,23256779e+01	1,14169189e+02	2,00000003e-01	9,91299973e+01
01.01.2010 00:00:00	30.12.1899 00:39:54	1	0	218241638	1036	2,23259087e+01	1,14169121e+02	1,21999998e+01	3,55089996e+02
01.01.2010 00:00:00	30.12.1899 00:40:28	1	0	218242102	1036	2,23263321e+01	1,14169151e+02	1,00000001e-01	1,07519997e+02
01.01.2010 00:00:00	30.12.1899 00:41:04	1	0	218242513	1036	2,23265324e+01	1,14169495e+02	1,51000004e+01	8,33600006e+01
01.01.2010 00:00:00	30.12.1899 00:41:38	1	0	218242924	1036	2,23261108e+01	1,14169800e+02	9,00000000e+00	1,70559998e+02

Table 1: weekNr.mdb

Accidents dataset

- Dataset Name - Hong Kong accident data in 2010
- Data Source - Transportation Department
- Size - 15.6 MB

Description: This dataset describes accidents which happened in Hong Kong in the year 2010. It assigns a severity level to each accident and specifies the location of each accident according to HK 1980 coordinate system [1]. Altogether there happened around 14,000 accidents in 2010 in Hong Kong.

Variables from acc_info.xls Example extract of this can be seen in Table 2

- ACC_NO: ID of the accident
- SEVERITY: Three levels of severity where zero is the mildest and three is most severe
- ACC_DATE
- ACC_TIME: Precise time of the accident
- LONG: Longitude coordinates (WGS 84) - initially the coordinates were in HK 1980 system
- LAT: Latitude coordinates (WGS 84) - initially the coordinates were in HK 1980 system

[1] More information on the coordinate system can be found in Subsection 2

ACC_NO	SEVERITY	ACC_DATE	ACC_TIME	Long	Lat	PREC_LOCTN
1	3	1/1/2010	0153	114.174409779	22.276712347	Johnston Road at the junction of Triangle Street Hong Kong
2	3	1/1/2010	0415	114.240695518	22.265059305	Yue Wan Market No. 33 - 33 Yee Fung Street Hong Kong Loading area
3	3	1/1/2010	0019	114.165681784	22.339465670	Tai Po Road at the junction of Cornwall Street SSPO Kowloon
4	2	1/1/2010	0129	114.167323129	22.327644747	Outside No. 50 Tai Po Road South SSPO Kowloon (The 1st lane was closed for about 45 minutes)
5	3	1/1/2010	0148	114.171041423	22.310911231	Nathan Road near junction of Wing Sing Lane YT Kowloon
6	2	1/1/2010	0214	114.169701901	22.314442143	Hamilton Street near junction of Portland Street YT Kowloon
7	3	1/1/2010	0615	114.169070676	22.320944143	MK Kowloon Somewhere in Mongkok area
8	3	1/1/2010	0645	114.173798049	22.297112531	Mody Road near junction of Minden Row YT Kowloon
9	3	1/1/2010	0245	114.173652441	22.298981867	Carnarvon Road near Cameron Road YT Kowloon
10	3	1/1/2010	0122	113.976319937	22.400993195	Lamppost H1686 San Wo Lane East ??????H1686?????????
11	3	1/1/2010	0010	113.946227459	22.239243188	Near Lamppost BC0548 Tung Chung Road LT New Territories
12	3	1/1/2010	0735	114.127933314	22.357438519	Near Lamppost DC 0118 Kwai Chung Plaza Kwai Foo Road KWC New Territories
13	3	1/1/2010	0845	114.185975862	22.277976397	Leighton Road Yun Ping Road Hong Kong
14	3	1/1/2010	1100	114.167491460	22.277705468	Near Lamppost 43189 Queen's Road East Wan Chai Hong Kong
15	3	1/1/2010	1550	114.199203247	22.290852796	King's Road West near junction of Tong Shui Road North Point Hong Kong
16	3	1/1/2010	1805	114.158767978	22.280856425	Des Voeux Road Central near junction of Ice House Street
17	3	1/1/2010	2152	114.173342424	22.277633453	Luard Road
18	3	1/2/2010	1000	114.179717292	22.274797843	Near Lamppost 41574 Morrison Hill Road South Wan Chai Hong Kong
19	3	1/2/2010	1125	114.203085019	22.291782369	King's Road near junction of Tin Chiu Street North Point Hong Kong
20	3	1/2/2010	1145	114.221956714	22.283370704	Tai Hong Street near junction of Hong Cheung Street
21	3	1/2/2010	1330	114.192662002	22.284179931	King's Road East near junction of Lau Li Street Hong Kong
22	3	1/2/2010	1420	114.136669927	22.259558197	Near Lamppost 34290 Chi Fu Road Hong Kong
23	3	1/2/2010	1538	114.132334307	22.286757366	Kennedy Town Praya East near junction of Queen's Road West Hong Kong
24	3	1/2/2010	1840	114.195272627	22.287033335	No. 32 - 32 Fortress Hill Road South North Point Hong Kong
25	3	1/2/2010	1922	114.183530266	22.269307097	Outside Hong Kong Sanatorium And Hospital No. 2 - 4 Village Road West Happy Valley Hong Kong
26	3	1/2/2010	2137	114.170945686	22.279457574	No. 3 - 3 Gloucester Road East WAN CHAI Hong Kong
27	3	1/3/2010	0120	114.173342424	22.277633453	Near Lamppost 37976 Repulse Bay Road
28	3	1/3/2010	1000	114.164910402	22.278129739	88 - 88 Queensway
29	3	1/2/2010	2200	114.188692908	22.280053269	Outside Hong Kong Central Library Causeway Road Wan Chai Hong Kong
30	3	1/3/2010	1615	114.132295555	22.286558681	Outside No. 11 - 15 Praya Kennedy Town Hong Kong
31	3	1/3/2010	1833	114.155358151	22.243243321	Near Lamppost 35283 Ap Lei Chau Bridge Road West Aberdeen Hong Kong
32	3	1/3/2010	2052	114.151373883	22.281830743	Opposite to No. 89 - 89 Caine Road East Hong Kong

Table 2: acc_info.xls

Speedpanel dataset

- Dataset Name - Hong Kong Speed Panels
- Data Source - Hong Kong Transport Department
- Size - 4.6 GB

Description: Speed data collected on Hong Kong main roads. Compared to speed panels data available in 2017, speed data in New Territories/Shatin did not exist in 2010 as Speed Map Panels there were only commissioned in 2013. For every day, one observation is collected in every two minutes from all speed panels.

This dataset is based on two types of files:

- tsm_dataspec.pdf - This file maps speedpanel nodes to coordinates in HK 1980 format
- `<DATETIME>.csv` - Multiple files for each day of the year with observations for each speedpanel pair collected every two minutes.

Variables from tsm_dataspec.pdf Example extract of this can be seen in Table 3.

- Link ID: stands for a specific road segment between start node and end node. Each node has a mapping to a specific location
- Start Node: ID of the starting node of the speedpanel
- Start Node Eastings
- Start Node Northings
- End Node: ID of the ending node of the speedpanel
- End Node Eastings
- End Node Northings
- Region: HK or K or ST or M
- Road Type: MAJOR ROUTE or URBAN ROAD

Link ID	Start Node	Start Node Eastings	Start Node Northings	End Node	End Node Eastings	End Node Northings	Region	Road Type
722-50059	722	834038.674	816345.067	50059	833862.7	816441.553	HK	MAJOR ROUTE
724-722	724	834148.783	816250.647	722	834038.674	816345.067	HK	URBAN ROAD
752-875	752	835099.22	815634.373	875	834918.334	815759.379	HK	URBAN ROAD
756-752	756	835352.749	815640.774	752	835099.22	815634.373	HK	URBAN ROAD
760-756	760	835602.186	815600.695	756	835352.749	815640.774	HK	URBAN ROAD
762-50078	762	836029.641	815629.493	50078	836298.985	815671.02	HK	MAJOR ROUTE
781-50098	781	836585.353	815155.819	50098	836705.363	815357.127	HK	MAJOR ROUTE
786-7861	786	836793.573	815852.754	7861	836714.038	815819.064	HK	URBAN ROAD
787-788	787	836656.904	815755.248	788	836606.459	815875.71	HK	MAJOR ROUTE
787-7882	787	836656.904	815755.248	7882	836603.538	815878.324	HK	MAJOR ROUTE
787-50179	787	836656.904	815755.248	50179	836362.957	815673.882	HK	MAJOR ROUTE
788-868	788	836606.459	815875.71	868	836682.388	815883.302	HK	MAJOR ROUTE
789-7908	789	836748.301	815908.341	7908	836950.758	815938.449	HK	URBAN ROAD
793-877	793	837428.196	816174.589	877	837616.85	816570.66	HK	MAJOR ROUTE
793-7918	793	837434.612	816161.378	7918	837006.391	815940.388	HK	URBAN ROAD
811-818	811	838249.981	817134.408	818	838881.248	817243.884	HK	MAJOR ROUTE
818-821	818	838881.248	817243.884	821	839406.398	817161.385	HK	MAJOR ROUTE
821-50107	821	839406.398	817161.385	50107	839823.434	816928.645	HK	MAJOR ROUTE
868-870	868	836682.388	815883.302	870	836803.159	815997.652	HK	MAJOR ROUTE
870-4651	870	836803.159	815997.652	4651	836884.007	816087.665	HK	MAJOR ROUTE
875-3402	875	834918.334	815759.379	3402	834754.135	815848.947	HK	URBAN ROAD
877-46498	877	837616.85	816570.66	46498	837982.489	816997.558	HK	MAJOR ROUTE
877-793	877	837626.399	816561.52	793	837434.612	816161.378	HK	MAJOR ROUTE
893-4652	893	832738.968	816762.428	4652	833101.677	816778.101	HK	MAJOR ROUTE
896-724	896	834264.546	816083.739	724	834148.783	816250.647	HK	URBAN ROAD
910-931	910	840060.004	816792.715	931	840101.178	816617.612	HK	MAJOR ROUTE
930-931	930	840495.49	816655.752	931	839976.076	816575.425	HK	MAJOR ROUTE
930-9101	930	840495.49	816655.752	9101	840052.115	816777.46	HK	MAJOR ROUTE

Table 3: tsm_dataspec.pdf

Variables from <DATETIME >.csv For each two minute timeinterval three types of information are collected. All the speedpanels pairs are stored in columns for each pair. The observations collected in specific two-minute time intervals are stored in rows. Example of this is presented in Table 4.

- Node XXX-YYY Speed (km/h): stands for the average speed of all the collected observations in this road segment between location XXX and YYY in a time period of two minutes.
- Node XXX-YYY Color: can be G (green), Y-(yellow) or R (red) where green stands for good and red stands for bad traffic condition. Enquiries about the methods used in specifying these classes have not been responded at the time of writing this dissertation.
- Node XXX-YYY Travel Time (mins) - Average travel time for cars travelling between two speedpanel nodes.

Date	Time	Node 3402-8979 Speed (km/h)	Node 3402-8979 Color	Node 3402-8979 Travel Time (mins)	Node 3651-4632 Speed (km/h)	Node 3651-4632 Color
1/1/2010	0:00:35	55.55472633	G	0.2	77.77887123	G
1/1/2010	0:02:35	55.55472633	G	0.2	69.5602234	G
1/1/2010	0:04:35	54.42624134	G	0.21	72.99038487	G
1/1/2010	0:06:35	55.55472633	G	0.2	77.77887123	G
1/1/2010	0:08:35	48.56595273	G	0.23	77.77887123	G
1/1/2010	0:10:35	48.34575136	G	0.23	72.051464	G
1/1/2010	0:12:35	55.55472633	G	0.2	77.7166583	G
1/1/2010	0:14:35	55.55472633	G	0.2	73.63084721	G
1/1/2010	0:16:35	55.55472633	G	0.2	74.32273534	G
1/1/2010	0:18:35	55.55472633	G	0.2	72.42247927	G
1/1/2010	0:20:35	54.6013361	G	0.2	77.77887123	G
1/1/2010	0:22:35	55.55472633	G	0.2	77.77887123	G
1/1/2010	0:24:35	55.55472633	G	0.2	77.77887123	G
1/1/2010	0:26:35	55.55472633	G	0.2	77.38159219	G
1/1/2010	0:28:35	55.55472633	G	0.2	73.87847406	G
1/1/2010	0:30:35	55.55472633	G	0.2	77.77887123	G
1/1/2010	0:32:35	52.62866681	G	0.21	72.49348572	G
1/1/2010	0:34:35	55.55472633	G	0.2	77.77887123	G
1/1/2010	0:36:35	55.55472633	G	0.2	75.65706027	G
1/1/2010	0:38:35	55.55472633	G	0.2	74.67300765	G
1/1/2010	0:40:35	55.55472633	G	0.2	77.77887123	G
1/1/2010	0:42:35	55.55472633	G	0.2	77.28339215	G
1/1/2010	0:44:35	55.55472633	G	0.2	77.77887123	G
1/1/2010	0:46:35	55.55472633	G	0.2	77.77887123	G
1/1/2010	0:48:35	55.55472633	G	0.2	73.5806153	G
1/1/2010	0:50:35	55.55472633	G	0.2	75.71351312	G
1/1/2010	0:52:35	55.55472633	G	0.2	77.77887123	G
1/1/2010	0:54:35	55.55472633	G	0.2	77.77887123	G
1/1/2010	0:56:35	55.55472633	G	0.2	77.67755985	G
1/1/2010	0:58:35	55.55472633	G	0.2	74.03786775	G

Table 4: <DATETIME.csv>

4.2.3 Overall Execution Sequence

The general order of execution steps are similar to the ones discussed in Kinoshita et al. [18]. However, with Hong Kong data the step of combining speedpanel data with GPS records is incorporated. In addition, some of the preprocessing steps like choosing a basemap format and integration of speedpanel data is described in a more technical and concise manner. This as mentioned previously is done because this dissertation pursues to be a bridge into understanding the phases of this project without a significant time cost.

This is the general order of preprocessing the data before it could be inserted into the algorithm.

1. Getting the data
2. Choosing a good basemap format
3. Route segmentation
4. Map matching for accidents and GPS records
5. Linear interpolation, trajectory generation
6. Speedpanel data integration
7. Data labelling

One could see that before starting the model training, there are a lot of phases that need to be gone through. Most of the steps described here are quite complex and strongly dependent on the previous phases. Starting from the route segmentation part, software developed by participants of this project is used for data cleaning and transformation, and its execution is described in a data-centered manner.

4.2.4 Technical description of the project source code

The source code for performing, analysing and testing steps 3-7 in the data preprocessing pipeline and incident detection is written in Java 8. It was redesigned to use the Gradle build system for library management.

The source code for model training using the EM algorithm is written by the authors of the paper: "Real-time traffic incident detection using a probabilistic topic model". Its written using C++11.

The code for visualising the results and comparing the results is written using Python 3.4 [2]

[2]Route Segmentation, Mapmatching for accidents and GPS records, Linear interpolation, Speedpanel Data Integration and Data Labeling

4.3 Basemap Data

4.3.1 Choice of tools and Map Format

The first step in any geospatial project is to choose a correct map format. OSM format was chosen for the specified background map format and the geospatial structures are built around it. OSM allows the use of additional open source tools for geographical querying and modification (Qgis [34],JOSM [30],Mapzen [33] and multiple other tools). There are various types of formats for an OSM map [3]. For the dissertation, the .osm format was chosen for formatting and as a data source. With mapping, it is important to have a tool to visualise the documents to cross check that everything is correct. To the author's knowledge, Qgis [34] is the most used tool for that, thus this project is also based on Qgis; however, as OSM is a widespread format, if necessary, it is also possible to use other tools to process and analyse the maps.

4.3.2 Coordinate Systems

Coordinate systems for mapping spatial data can be either

- projected - coordinates in northing, easting. For example HK1980 Grid - The accidents file is in this format. These are coordinates converted to a flat paper kind of surface; or
- spherical - coordinates in longitude, latitude. For example wgs84. Coordinates on the sphere.

The datasets of this project use both of these coordinate systems so conversion needs to be performed. As the largest dataset by far is the taxi GPS record dataset, the conversion is done from HK 1980 to WGS84 format. Hong Kong Government has provided a web application to perform such conversion [12]. The datasets that needed conversion were speedpanel data and accident data. For accident data, an additional procedure of adding 800000 to the coordinates was needed before being able to convert the coordinates with the government provided tool [12].

4.3.3 Details about OSM format

The cornerstones of OSM are

- Nodes and
- Ways

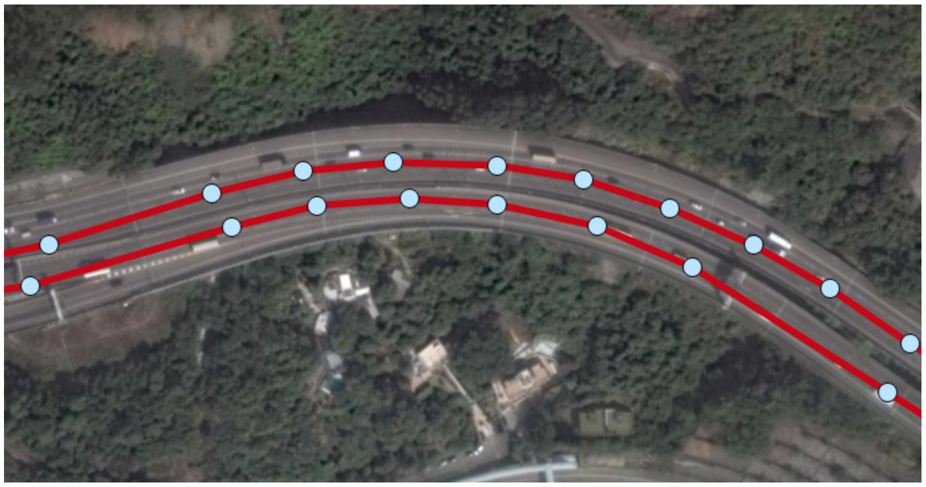
Nodes carry the geographic coordinates in the OSM data model. A way only gets geometry via the nodes that are members of it. Most of the time such nodes will be untagged and only serve to determine the way geometry. There are also nodes with tags, these show points of interests. An example of nodes united under a way can be seen on Figure 2a.

4.3.4 Choosing Roads

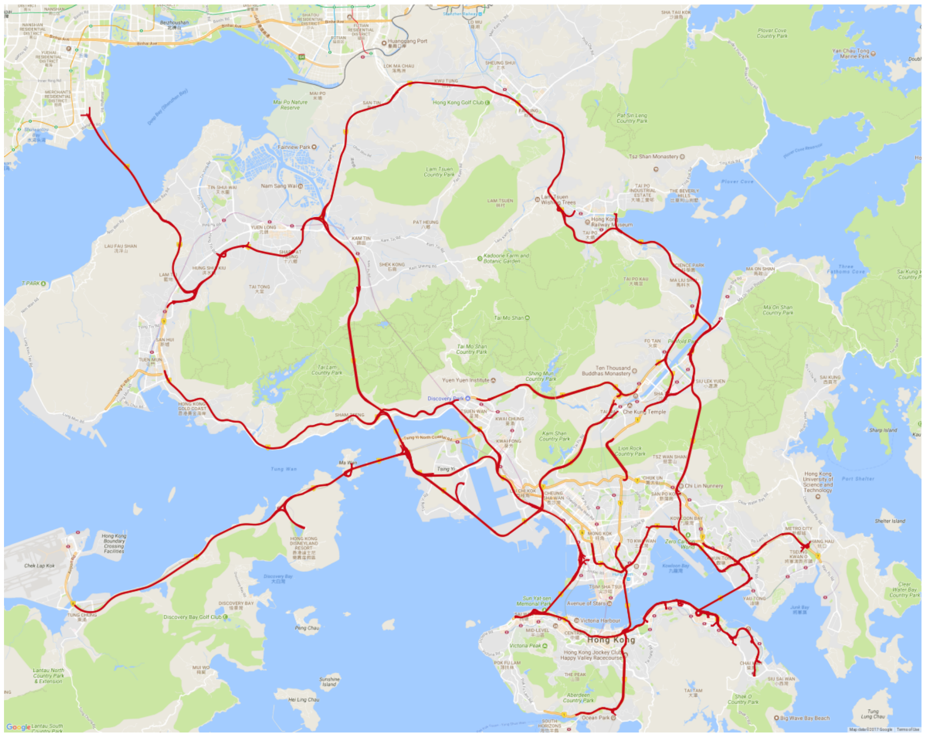
As mentioned before, nodes and ways can both contain specific tags with information. For this project only certain ways and nodes were needed. More specifically, nodes and ways which stood as roads were chosen. These roads were also filtered so only roads with the OSM tag trunk and motorway were kept. Such a choice was made because these tags specify most important roads in a country's system and are likely to produce the most data with the highest quality. For a developmental project this is a very important aspect because it allows to focus on other aspects like model development instead of trying to satisfy all conditions. This also allowed to include roads which are under the supervision of speedpanels. The roads that were finally chosen to be analysed can be seen on Figure 2. The ways (roads) have only one direction. To perform the filtering of the roads a Java-based geospatial tool JOSM [30] was used.

4.3.5 OSM parsing

To make the visualisation of the geospatial data easier, a static class called `bgt.parsing.OSMparser` was created. This parser is capable of transforming most of the intermediate data that is generated in the preprocessing phase into visualisable OSM format. For example GPS records from taxis and accidents, trajectories, speedpanel location data and detection rate can be and are visualised with this class in the later phases of this project. This parser serves as a project on its own and could be used separately to visualise geospatial data.



(a) Here nodes are marked with light-blue and ways based on nodes are marked with red



(b) Roads included in the analysis are marked with red

Figure 2: Basemap

4.4 Step I: Segmentation of Routes

4.4.1 General

- Running time - less than 10 seconds.

This step takes ways and nodes and modifies them so that in each way the distance between sequential nodes is within a predefined interval (50 m-100 m). As was mentioned earlier, the ways both in the input and the output have only one direction. In principle, this step takes the map with ways consisting of ordered nodes (shown as points) like shown in Figure 3a and transforms these nodes so the nodes will be distributed equally like shown in Figure 3b. In later phases, the road straights between the nodes are going to be the structural background for map matching, model building and incident detection. Compared to the old version here the code was redesigned to make visualisation with Qgis possible and the overall structure more clear. In addition, functions to summarise the information on the routes were built in a class called `main.java.bgt.RouteSegmentation.Utilityfunctions.java`.

4.4.2 Code

Files

Data Models

- bgt/Model/Node.java
- bgt/Model/Way.java
- bgt/Model/Routes.java
- bgt/Model/Boundaries.java

Controllers and Interfaces

- RouteSegmentor.java

Execution Before continuing this part it is important to note that newly created data objects that have not appeared before in this text are marked with bold.

1. First the `/data/route\_segmentation/Hong_Kong_Highways-Merged-Remove_Deleted.osm` file is read by OSMparser.java
2. Based on the parsed result a **Routes** object named "routes" is created. It consists of 3 other objects: **Boundaries**, **Ways** (List), **Nodes** (List).
3. Routes object transformation function: routes.resegmentWays(...) is then initiated by RouteSegmentor.java object, which is the controller of this segmenting step.

4. After routes.resegmentWays(...) has finished, it returns the transformed routes object. After that object to file parsing functions are initiated:
   - OSMparser.routesToOSM(routes, ""); Writes the routes object back to osm format
   - OSMparser.wayToOSM(routes.getWayList().get(0),routes) Writes a sample way from the routes object to osm format.

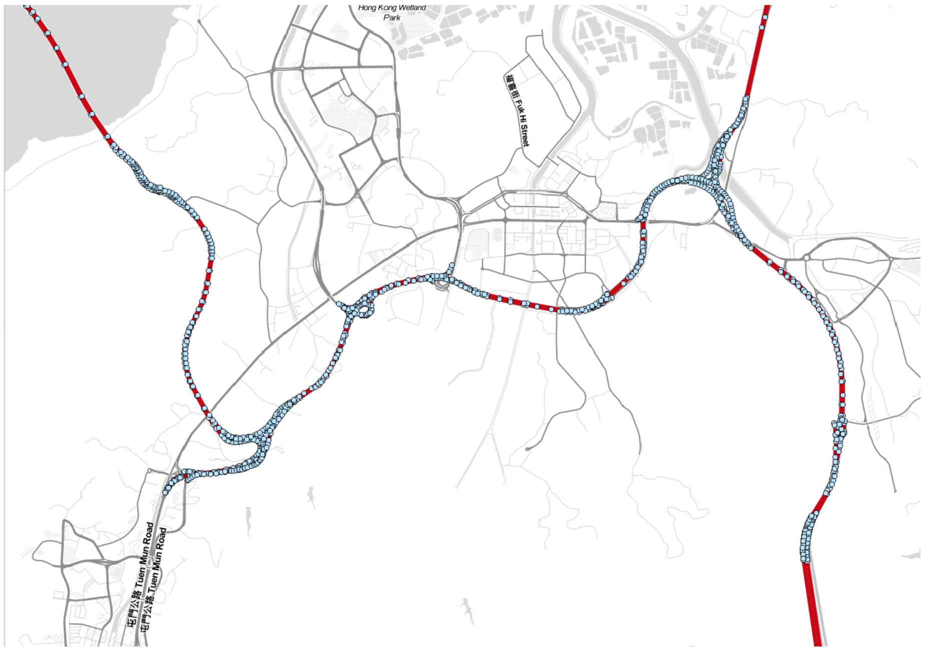
4.4.3 Input and Output files

Input `/data/route_segmentation/Hong_Kong_Highways-Merged-Remove_Deleted.osm`

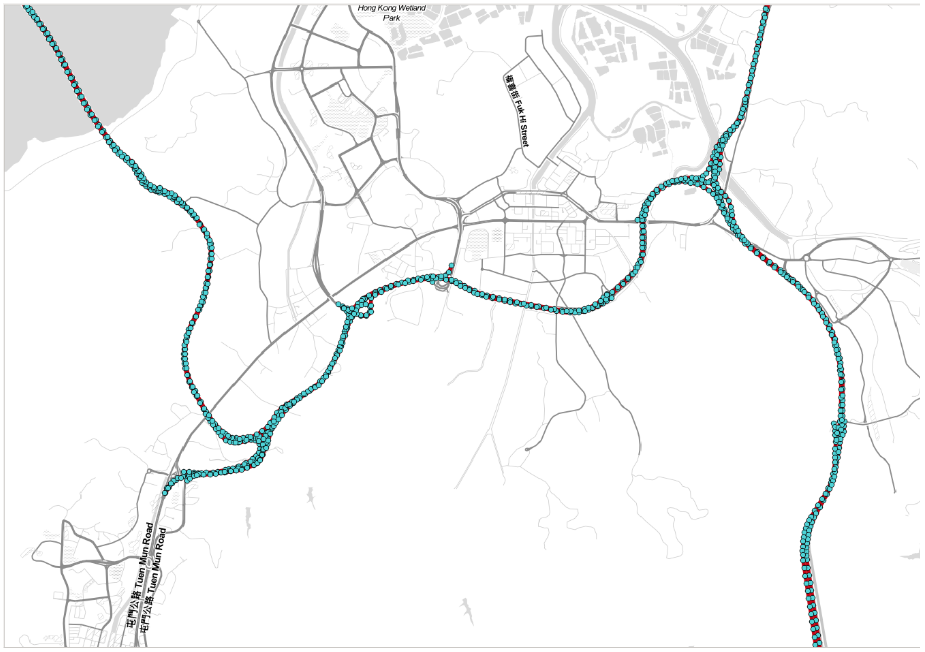
Output `/data/route_segmentation/Hong_Kong-result.osm` `/data/route_segmentation/<String.valueOf(wayForPrint.getId())>+".osm"`

4.4.4 Test and Analysis

- After the segmentation Utilityfunctions.findeExtremumInWay(Way way, BiPredicate <Double, Double>p, double extremum) calculates the length of ways to makes sure they are between the allowed limits. It outputs the tests to the terminal.
- Visualisation like shown Figure 3, 3a and 3b. With geospatial data processing, visualisation is one of the best ways to verify that you have received a correct result.



(a) input



(b) output

Figure 3: Here you can see how distances between nodes vary in lengths. This is due to the osm map format which uses nodes as geography markers. In some places more nodes are needed than in others to show the geometry of the road. After segmentation, however, for this project the distances are more or less even and in the bounds of 50-100 m

4.5 Step II: MapMatching

4.5.1 General

- Initial running time - Around five hours
- After redesigning the code to use temporary files - less than two hours.

This part reads in the GPS records collected from Hong Kong taxis and accidents records received from the Hong Kong Transport department. Both of the records are then matched to specific road segments which are described in the previous section. This is a very important and very complicated part in the whole incident detection project because slight inaccuracies in matching the GPS data and accident data can have negative results in the overall performance of the incident detection model. To perform map matching the segmented map is read in and Segment objects with lists for Accidents and Records are created. Each generated segment will be assigned a direction. Next, a grid with 100 m X 100 m squares will be generated over the whole Hong Kong area. Every segment is assigned into 1 or multiple grid squares. After creation of the grid, the map is ready for matching GPS records and accidents to it. For each record that is matched, first its location inside the grid is calculated, then the record is compared with all the segments inside the grid elements based on direction and distance. The matching of accidents is similar however without direction comparison.

4.5.2 Code

Files

Data Models

- bgt/Model/Grid.java
- bgt/Model/GridElement.java
- bgt/Model/Segment.java
- bgt/MapMatching/Accident.java
- bgt/Model/Routes.java
- bgt/MapMatching/Record.java

Controllers and Interfaces

- MapMatcher.java

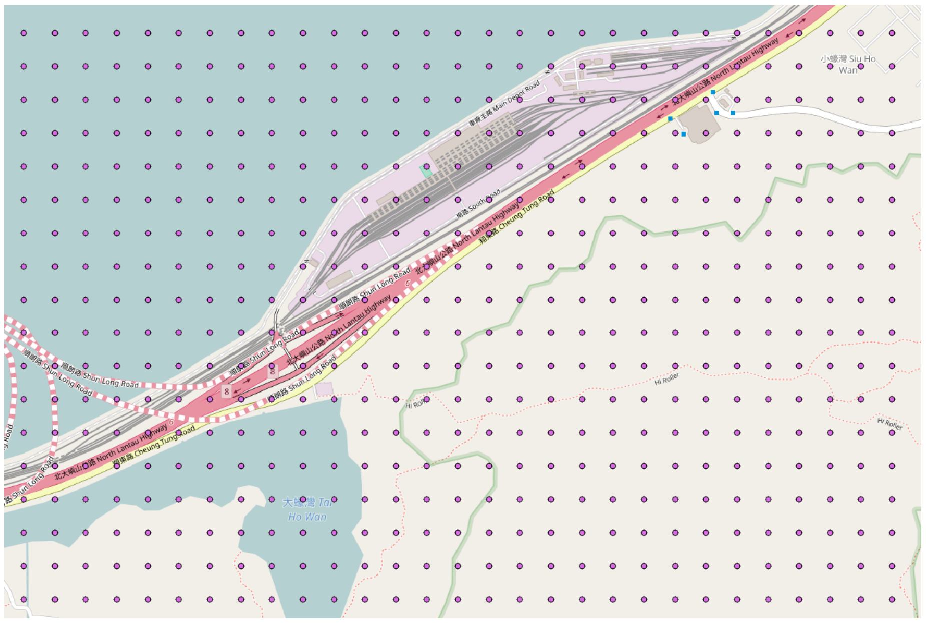
Execution

Record matching

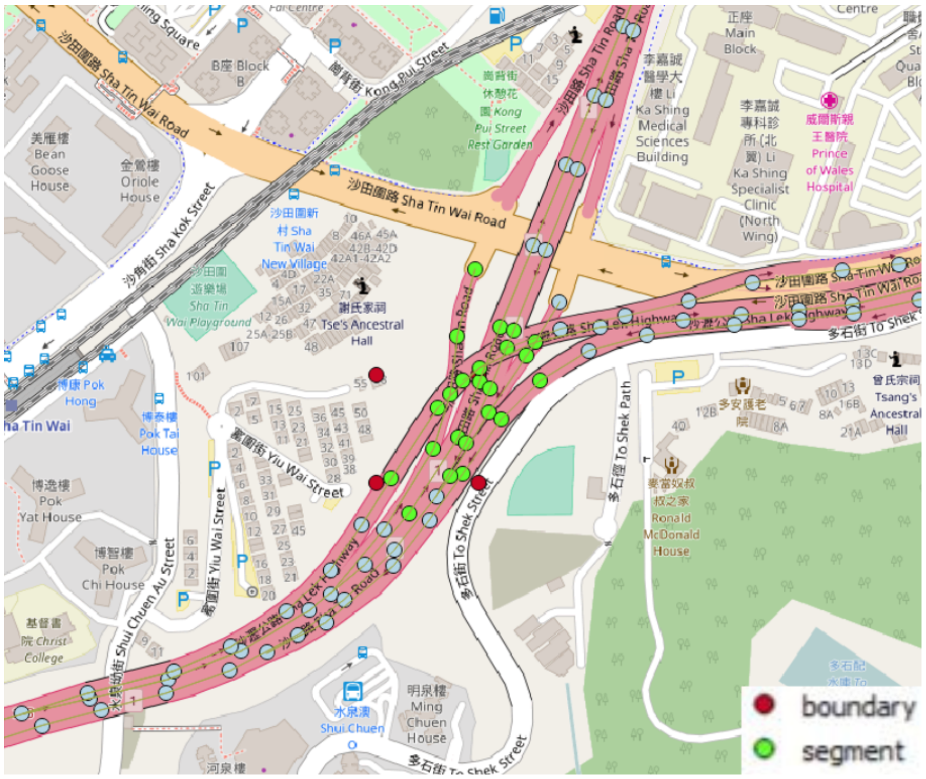
1. **MapMatcher mm = new MapMatcher()** First the constructor of the controller is called. In the constructor, The `Hong_Kong-result.osm` file generated in the previous step is read in and based on that, a new Routes class object is created which represents a map with segmented ways. Based on the Routes object generated in the previous step, new **Segment** objects are generated for every way and for every sequential node pair. For each segment its constructor also triggers the direction calculation for that segment. The direction calculation is based on the starting node and the end node coordinates which are nodes sequentially taken from one directional way according to the way order. Each segment object that is generated is added to two datastructures. Here advantage is going to be taken of the mutability of Java objects. When in map matching phase later on accidents and GPS_record objects are added to Segments, it has to be done only once.

   - `ArrayList<Segment> segList`
   - `HashMap <String, Segment> segMap`

2. After the constructor has generated a Routes object and the Segment objects, function **mm.indexSegs()** is called to generate a spatial grid over Hong Kong map. This grid divides the Hong Kong area into 100 m X 100 m squares. Each Segment object generated in the previous phase is assigned to one or multiple grid squares based on its start and end location. The grid in Java is represented as a multidimensional list which stores segment IDs in the form of a string. `List<String>[][] segIDGrid`. A specific example of how grids are stored over the map area is shown on Figure 4a. How segments are placed to one grid can be seen on Figure 4b.



(a) Grid example



(b) Segment assigned to a grid which is a square bordered with red corners.

Figure 4: Grid element containing segments. Grid is a construct built for optimizing the map matching algorithm. A segment is a one-directional road part between two nodes.

3. `mm.parseRecords()` reads in every record from the taxi GPS record .mdb database files for every week and creates a **Record** object based on that. The code was redesigned so Record objects are read in one week at a time. Each record is matched to a specific segment and written into a temporary segment file. After going through each file it prints out how much time it took to perform matching for all the files and what is the percentage of the records matched to roads. The general execution in Java looks like this:

```
public long parseRecords(String flag, int start_fileNum, int
    end_fileNum, int foldNr,boolean justTest) throws IOException {
       // map the record into the segments

       long start = System.currentTimeMillis();
       createEmptySegmentFiles(foldNr);
       int lastFileNr = justTest ? 2 : 52;
       for (int i = 0; i <= lastFileNr; i++) {//(for_loop)
           if (flag.equals("train") && (i >= start_fileNum) && (i
               <= end_fileNum)) {
               continue;
           }
           if (flag.equals("test") && ((i < start_fileNum) || (i >
               end_fileNum))) {
               continue;
           }
           // NAMING PART
           String records_filename = "";
           if (i < 10) {
               records_filename = "data/GPS_data/TaxiData20100" + i
                   + ".mdb";
           }else {
               records_filename = "data/GPS_data/TaxiData2010" + i
                   + ".mdb";
           }

           parseRecordsOfFile(records_filename);//(parsing)
           appendIntermediateResultsFlush(foldNr);//(append)
           printElapsedTime(start);
       }
       System.out.println("Fold " + foldNr + " took:");
       printElapsedTime(start);
       return printElapsedTime(start);

   }
```

The comments with syntax "(`<text>`)" stand for IDs to reference code from outer text.

(a) At code line (for _loop) this function goes through all the database files. There are 52 files. Each file stands for one week.

(b) At code line (parsing) this function starts parsing a specific database file and reading in all the records. Each record that is read in is matched to a specific segment with `mm.mapRecord(Record record)` function.

(c) At code line (append) this function writes to a temporary segment file the Records that were assigned to each Segment object. The records are written to the the file in the same order they were added to the segment recordList.

More specifically `mm.mapRecord(Record record)`

```java
public String mapRecord(Record record) {
    Node recNode = record.getLocation();
    // (GRIDMAPPER)
    int latIndex = (int) (Node.calcDist(recNode.getLat(),
        MIN_LON, MIN_LAT, MIN_LON) / GRID_SIZE);
    int lonIndex = (int) (Node.calcDist(MIN_LAT,
        recNode.getLon(), MIN_LAT, MIN_LON) / GRID_SIZE);

    int minLatIndex = latIndex - 1 >= 0 ? latIndex - 1 : 0;
    int maxLatIndex = latIndex + 1 <= MAX_LAT_INDEX ? latIndex
        + 1 : MAX_LAT_INDEX;
    int minLonIndex = lonIndex - 1 >= 0 ? lonIndex - 1 : 0;
    int maxLonIndex = lonIndex + 1 <= MAX_LON_INDEX ? lonIndex
        + 1 : MAX_LON_INDEX;
    // (GRIDMAPPER)

    List<String> checkedSegIDList = new ArrayList<>();
    double minDist = minAcceptableDist+1;
    String minSegID = null;
    for (int i = minLatIndex; i <= maxLatIndex; i++) {
        for (int j = minLonIndex; j <= maxLonIndex; j++) {//
             (GRID SEGMENTS)
            // map the segments in each grid
            List<String> segIDList = this.segIDGrid[i][j];//
                here are the segments that belong to the
                specific gridelement . these segments are mapped
                in indexSegs()
            for (String segID : segIDList) {
                if (checkedSegIDList.indexOf(segID) == -1) {//
                     OII
                    // this segment has not been checked
                    Segment seg = this.segMap.get(segID);
                    if (seg.calcRecordDirectDist(record) >
                        minAcceptableDeg) {
                        continue;
                    } else {
                        double dist = seg.calcNodeDist(recNode);
                        if (dist < minDist) {
                            minDist = dist;
                            minSegID = segID;
                        }
                    }
                }
            }
        }
    }

    if (minDist < minAcceptableDist) { (GOOD ENOUGH)
        // there exists a segment that has a distance less than
            100 from the record,
        // and their direction difference is less than 45
            degree.
        this.segMap.get(minSegID).addRecord(record); // adds
            TRAFFIC RECORD TO CLOSEST SEGMENT
        return minSegID;
    }
    return "";
}
```

- At code line (GRIDMAPPER) this part maps the Record into the right place in the grid - The mapping chooses three potential squares in the grid totaling 300 m X 300 m. All the segments that were assigned to these grids are going to be looped through in the next code segment.
- At code line (GRID SEGMENTS) all the segments inside this grid area looped through. For each segment a direction comparison is done with the record 1. If the direction between the record and the segment is less than 45 degrees, the distance is compared between the record and the specific segment.
- At code line (GOOD ENOUGH) if a segment has been found that has a distance less than variable minAcceptableDist from the record, and their direction difference is less than 45 degree (checked earlier) this record is added to the closest segment recordList with `this.segMap.get(minSegID).addRecord(record);`

4. Finally `generateFullOutput(int foldNr, String record_match_filename )` function is called to put all segments separate files together into one large mapmatch_result.txt. The function goes through the segment list generated in the constructor which has all the segments listed in the order the segments were ordered in ways. It then reads in the records of the segment to segment's recordlist and sorts the recordList. After the recordList has been sorted it outputs the result to the file `test|train_mapmatch_result.txt` `this.segList`

Accident matching public void parseAccidents() throws IOException, BiffException

- For every accident it calls the mm.mapAccident(acc) function.
  1. This finds nine closest grid squares in the segIDGrid list/matrix
  2. It goes through all the segments in the chosen grid squares to find the closest Segment for the Accident.
  3. It assigns the Accident to only one segment.
- outputAccResults() writes all the accidents to the outputfile named `data/map_matching/acc_match_result.txt`

```
public void outputAccResults(String acc_match_filename) throws
     IOException{
    FileWriter fw_acc_result = new FileWriter(acc_match_filename);
    Iterator<Segment> segItr = segList.iterator();
    while (segItr.hasNext()){
        Segment seg = segItr.next();
        if (seg.accNum() > 0){
            Iterator<Accident> accItr = seg.accList.iterator();
            while (accItr.hasNext()){
                Accident acc = accItr.next();
                fw_acc_result.write(
                        seg.seg_id + "\t" + acc.severity + "\t" +
                            acc.month + "\t" +
                               acc.day + "\t" + acc.acc_time + "\t" +
                               acc.getAcc_location().getLat()+","+acc.getAcc_location().getLon()
                                   + "\n"
                );
            }
        }
    }
    fw_acc_result.close();
```

4.5.3 Input and Output files

Input

- `Hong-Kong_result.osm` map file
- `<WEEK>.mdb` taxi database files - 52 files
- `axx_info.csv` accidents file with location information, the accident level and the time.

Output

- `train|test_match-result.txt` Example of this is shown in Table 5. In this file records are grouped into segments after which they come in this file. Inside segments the records are ordered based on time.

Tag: S/R	ID	Device ID	Month	Day	Time	Latitude Longitude	Direction	Speed
S	4338869_0_0							
R	218493312	1544	1	1	7.0622222222	22.248531341552734,114.16349029541016	55.7999992371	99.3199996948
R	218808560	1544	1	1	14.1375	22.2485408782959,114.1636962890625	57.5	96.4499969482
R	218379787	1499	1	1	3.8405555556	22.24854278564453,114.16352081298828	67.3000030518	99.2099990845
R	218922186	1446	1	1	16.4347222222	22.248504638671875,114.1637954711914	55	98.4000015259
R	218779995	1139	1	1	13.5397222222	22.248502731323242,114.16346740722656	67.9000015259	98.5
R	218228934	1549	1	1	0.3844444444	22.24834632873535,114.1637954711914	44.4000015259	117.4100036621
R	218683794	1445	1	1	11.5269444444	22.248563766479492,114.16340637207031	55.2000007629	95.9899978638
R	219211282	1524	1	1	22.5180555556	22.24782943725586,114.16362762451172	60.2999992371	100.2200012207
R	218678128	1449	1	1	11.4091666667	22.248577117919922,114.16326904296875	78.5999984741	96.9300003052
R	219233958	1449	1	1	23.0208333333	22.248592376708984,114.16325378417969	63.7999992371	99.7799987793
R	218529128	1450	1	1	8.0938888889	22.248519897460938,114.16346740722656	59.5999984741	98.3700027466
R	218820746	1450	1	1	14.3975	22.248430252075195,114.16376495361328	60.7999992371	98.5100021362
R	218382914	1444	1	1	3.9188888889	22.248592376708984,114.16314697265625	63.2000007629	99.2900009155
R	218406758	1444	1	1	4.5441666667	22.248502731323242,114.16365814208984	60	96.4400024414
R	218893073	1444	1	1	15.8525	22.248580932617188,114.16316986083984	75.9000015259	98.1299972534
R	219157832	1245	1	1	21.3308333333	22.248607635498047,114.1630630493164	64.8000030518	98.0199966431
R	218954451	1376	1	1	17.1044444444	22.248567581176758,114.16351318359375	68.9000015259	97.3099975586
R	219014725	1290	1	1	18.3252777778	22.24856185913086,114.16360473632812	46.5	97.8199996948
R	218479874	1187	1	1	6.6622222222	22.24858283996582,114.16336059570312	61.4000015259	94.2600021362
R	218873558	1495	1	1	15.4691666667	22.24867820739746,114.16354370117188	36.0999984741	72.4499969482
R	219276049	1532	1	1	23.9680555556	22.248716354370117,114.16357421875	63.9000015259	86.6800003052
R	218374693	1545	1	1	3.7113888889	22.24848747253418,114.16350555419922	79.9000015259	98.5699996948
R	218683740	1545	1	1	11.5258333333	22.248573303222656,114.16314697265625	65.3000030518	96.9400024414

Table 5: Part of test_match_result.txt

- `acc_match_result.txt` . Example of this can be seen in Table 6 information, the accident level and time.

Segment id	Accident severity	Month	Day	Time	Latitude/Longitude
4338869_0_0	3	9	7	21.25	22.248617402,114.162992085
16281867_0_0	3	11	19	18.4166666667	22.28187712,114.161009327
16281867_2_0	3	5	9	20	22.281615368,114.162474564
22371280_4_0	3	8	26	17.3333333333	22.368510334,114.071751076
22371280_15_0	3	1	25	9.7666666667	22.36940833,114.078061525
22732384_0_0	3	1	13	16.8333333333	22.270598579,114.180415793
22732384_0_0	3	11	5	11.9666666667	22.270779192,114.180406095
23143626_0_0	3	6	11	22.8333333333	22.304805131,114.235317619
23143626_0_0	3	7	5	22.3666666667	22.304750954,114.235298187
23143626_0_0	3	10	6	12.4166666667	22.304850168,114.235637901
23143626_1_0	3	1	20	9.9166666667	22.304498009,114.235540705
23143626_1_0	3	2	7	13.5833333333	22.304624455,114.235492233
23143626_1_0	3	3	12	15.1833333333	22.304651551,114.235482539
23143626_1_0	3	3	16	10	22.304678639,114.235492255
23143626_1_0	3	6	1	17.3333333333	22.304606394,114.235492225
23143626_1_0	3	6	27	23.3333333333	22.304488979,114.235540701
23143626_1_0	3	8	15	0.6666666667	22.304579264,114.235598968
23143626_1_0	3	10	17	3	22.304552204,114.235511613
23143626_1_0	3	12	11	17.9333333333	22.304443829,114.235530977
23143626_2_0	2	4	15	6.5	22.304218054,114.235559998
23143626_2_0	3	5	13	6.4166666667	22.304145802,114.235579377
23143626_2_0	3	8	23	21.2333333333	22.304362448,114.235822091
23143626_2_0	3	11	9	9	22.304208974,114.235695863
23143626_2_0	3	11	22	8.1666666667	22.303856664,114.236015976
23143626_2_0	3	11	29	20.6666666667	22.304163873,114.23555027
23298496_11_0	3	1	29	21.8166666667	22.261874976,114.180037193

Table 6: Part of acc_match_result.txt

4.5.4 Test and Analysis

1. `test.java.bgt.MapMatching.MapMatcherTest public void parseRecords()`
   The first test is to check that the intermediate results in temp folder to which results are appended does not have any newlines. It also checks that the endresult file does not have any newlines in between. It verifies that the total amount of rows in the endresult file has the same amount of rows as in all the segment files. The test also checks that all the matched records in the endresult have exactly the same records that exist in the temporary files.

2. `test.java.bgt.MapMatching.MapMatcherTest public void checkDBorder()`
   This test was used to make sure that the database rows for GPS records are in a timely order. The GPS records are not in an exact order in regard to time. For each file it calls the function checkDbFile().

3. `test.java.bgt.MapMatching.MapMatcherTest public void parseRecordsTrajSegOrder()`
   This test checks that the segments are printed out in the exact order they were listed in ways.

4. `test.java.bgt.MapMatching.MapMatcherTest parseRecordsTimeSegOrder()` This test checks that records assigned to each segment in the `mapmatch_result.txt` file are in order.

5. `main.java.bgt.MapMatching.Analysis buildHistogram()` Writes down each segment' s density.

6. `main.java.bgt.MapMatching.Analysis segAccDensityOSM()` visualises each segment's density of GPS records in the OSM map

7. `main.java.bgt.MapMatching.Analysis segRecDensityOSM()` visualises each segment's density of accidents in the OSM map

8. `main.java.bgt.MapMatching.Analysis printGridWithSegments()` Builds the grid with most segments assigned to the grid.

4.6 Step III: Linear Interpolation

4.6.1 General

- Initial running time - Without extra memory specifications for the JVM, could not complete running. Had to modify the JVM Heap settings. Time taken: 8 hours.

- After redesigning the code to use temporary files: 1 hour.

On faster roads, the taxis' travelling speed is so high that GPS readings which take place in 30 second time intervals cannot create an observation in each sequential segment. On the other hand, on slower routes with heavy congestion, taxis can generate multiple observations inside one segment. In this phase, trajectory generation is done for each device. If inside a trajectory, there are some segments skipped as shown in the upper part of Figure 4, interpolated records are going to be created between the sequential segments. At the same time, one trajectory can only contribute an observation to one segment.

4.6.2 Code

Files

Data Models

- bgt/Model/Segment.java

- bgt/Model/Routes.java

- bgt/MapMatching/Record.java

Controllers and Interfaces

- bgt/LinearInterpolation/Analysis.java
- bgt.LinearInterpolation.InpolatParser
- bgt.LinearInterpolation.LinearInterpolation
- bgt.LinearInterpolation.TrajectoryGenerator

Execution

Overall: First a constructor is called to generate a Routes and Mapmatching object. These objects provide the underlying data structures for performing linear interpolation. After that function `li.parseAndInterpolatePieceWise(fileName,i)` is called. This function takes the result from the map matching step: `train_match_result.txt` and

1. Obtains all the Device ID-s in the train_match_result.txt file
2. Divides the Device ID-s in each file into buckets consisting of specified number of device ID-s
3. Creates a new `TrajectoryGenerator` object named `tG`
4. For each bucket it calls:
    - `tG.parseRecords(filename, fracDevID, trajsMap, lines)` function which reads into Hashmap `devRecMap` all the GPS records that are matched to segments for the devices in the bucket.
    - `tG.genTrajForDevices(devRecMap)` which generates trajectories based on the GPS Record objects read in by `tG.parseRecords()`
    - `interpolate(trajsMap)` function which removes and generates GPS Record objects based on some information obtained from the trajectories

`TrajectoryGenerator.parseRecords(filename, fracDevID, trajsMap, lines)` This function reads in all the GPS observations from all the segments for GPS devices specified in the HashMap `fracDevID`. These Records are added to a list in a HashMap `HashMap<Integer (device ID), List<List<Record>>> partjalTrajsMap`

`TrajectoryGenerator.genTrajForDevices(HashMap<Integer, List<Record>> devRecMap)` The trajectory generation was completely revisited for this project as the previous trajectory generation logic missed an enourmous number of trajectories as can be seen on Figure 16 . This function receives as an input an HashMap with GPS Record lists for all the GPS devices specified in the bucket.

For each device it

1. Sorts the devices GPS Record list based on Record observing time
2. Goes in a timely order through the GPS Record list and adds a new Record to trajectory list created earlier if the new Record is spatially and temporarily close to the previous Record. Othewise it adds the earlier created trajectory to a finished trajectory list and generates a new trajectory with the current Record as a start of the trajectory.

The restrictions that sequential GPS observations must follow to be considered as belonging to the same trajectory are:

```
if (thisSeg.way_id == lastSeg.way_id &&
                      thisSeg.inner_id >= lastSeg.inner_id &&
                      rec.month == lastRec.month &&
                      rec.day == lastRec.day) {
                  if (rec.time > lastRec.time && rec.time - lastRec.time
                       <= (1.0/60.0)){
                      //lastTraj.add(rec);//
                      trajLength++;
                      trajectory.add(rec);
                  }else if(rec.time == lastRec.time){ // GPS device is
                       broken in this case
                      continue;// just eliminates failed records.
                  }else{
                      addTrajForDevice(partjalTrajsMap,devId,trajectory);
                      trajLenghts.add(trajLength);
                      trajLength=1;
                      trajectory= new ArrayList<>();
                      trajectory.add(rec); // new trajectory is started
                  }
              }
...
}
```

In the previous code segment

- `thisSeg` is the segment that the current Record (`rec`) was matched to whereas `lastSeg` is the segment that the last record (`lastRec`)in the sorted list was matched to.
- The limitation for the trajectory generation is that the records to be listed inside the same trajectory must belong to the same way. Only then can the records be considered as sequential by being matched to segments that have sequential inner ID-s in the context of that way.

Interpolation: After a Hashmap of trajectories for all the specified devices has been created, interpolation function is called. This function goes through the data and modifies it, so that all segments have exactly one Record **from each trajectory**. The general steps for intepolation are

1. Going through all the devicesHashMap<Integer, List<List<Record>>> trajsMap generated by tG.genTrajs(...) function.

```
Iterator trajsItr = trajsMap.entrySet().iterator();
while (trajsItr.hasNext()){ // ITERATER OVER ALL DEVICES
   Map.Entry dev_trajs_pair = (Map.Entry)trajsItr.next();
   int dev_id = (int)dev_trajs_pair.getKey(); //
   List<List<Record>> trajs = (List<List<Record>>); // (1)
       dev_trajs_pair.getValue(); // GET NEXT LIST OF
       TRAJECTORIES FOR A PARTICULAR DEVICE
}
```

2. Going through each devices trajectories

```
Iterator<List<Record>> trajItr = trajs.iterator();
while (trajItr.hasNext()){
   List<Record> traj = trajItr.next();
   ...
}
```

   In each trajectory, its first and last is extracted

```
Record firstRec = traj.get(0);
int firstInner_id = segsMap.get(firstRec.seg_id).inner_id;
Record lastRec = traj.get(traj.size()-1);
int lastInner_id =
    segsMap.get(traj.get(traj.size()-1).seg_id).inner_id;
```

3. If trajectory takes place only in one segment then multiple records are merged into one.

```
  if (lastInner_id == firstInner_id){
     double distance = Node.calcDist(firstRec.getLocation(),
         lastRec.getLocation());
     double speed =
         distance/1000.0/(lastRec.time-firstRec.time); // GET
         SPEED
     double time = (firstRec.time + lastRec.time) / 2;
     Record newRec = new Record(dev_id, firstRec.month,
         firstRec.day, time, speed, firstRec.seg_id);
     newTraj.add(newRec);
  }
```

4. If there are more segments than one in a trajectory it will be looped through. If the difference of segment ID-s between sequential records is larger than one, the GPS has skipped segments and new Records are created through interpolation into the skipped segments. The segments

themselves were generated sequentially from ways so there is an ordering between segments from ways.

```
int inner_id_diff = lastInner_id-firstInner_id;
      double[] times = new double[inner_id_diff+2];
      times[0] = firstRec.time; times[inner_id_diff+1] =
           lastRec.time;
      int i = 0, j = 1, index = 1;
      while (j < traj.size()){
          Record sRec = traj.get(i);
          Record eRec = traj.get(j);
          int sInner_id = segsMap.get(sRec.seg_id).inner_id;
          int eInner_id = segsMap.get(eRec.seg_id).inner_id;
          if (eInner_id - sInner_id >= 1){
              double dist_s = Node.calcDist(sRec.getLocation(),
                   segsMap.get(sRec.seg_id).endNode);// DISTANCE
                   BEWTWEEN THE CURRENT START RECORD IN THE SEGMENT
                   OF THIS TRAJECTORY
              // AND THE SEGMENT END NODE
              double dist_e =
                   Node.calcDist(segsMap.get(eRec.seg_id).startNode,
                   eRec.getLocation());
              //// DISTANCE BEWTWEEN THE END RECORD IN THE END
                   SEGMENT OF THIS TRAJECTORY PART
              // AND THE SEGMENT START NODE
              double dist = dist_s + dist_e;// DISTANCE BETWEEN 2
                   RECORDS IN TRAJECTORY
              String[] seg_id_frags = sRec.seg_id.split("_");
              double[] inter_dists = new
                   double[eInner_id-sInner_id-1];
              for (int k=sInner_id+1; k<eInner_id; k++){ //
                  String seg_id =
                       seg_id_frags[0]+"_"+k+"_"+seg_id_frags[2]; //
                       GETS THE INTERMEDIATE SEGMENT ID
                  Segment inter_seg = segsMap.get(seg_id);
                  double inter_dist =
                       Node.calcDist(inter_seg.getStartNode(),
                       inter_seg.getEndNode());// GETS THE
                       INTERMEDIATE SEGMENT LENGTH
                  inter_dists[k-sInner_id-1] = inter_dist;
                  dist += inter_dist;
              }
              double timeDiff = eRec.time - sRec.time;
              times[index] = sRec.time + dist_s/dist*timeDiff;
              index++;
              for (int k=sInner_id+1; k<eInner_id;
                   k++){//INTERPOLATES TIME FOR THE INTERMEDIATES
                   SEGMENT
                  double time_add =
                       inter_dists[k-sInner_id-1]/dist*timeDiff;
```

```
                times[index] = times[index-1] + time_add;
                index++;
            }
        }
        i++; j++;
    }
```

5. Based on the info above, new GPS records are generated .

```
    for (i = firstInner_id+1; i<lastInner_id; i++){
      int pure_index = i - firstInner_id+1;
      String innerSeg_id = first_seg_id_frags[0] + "_" +i + "_" +
          first_seg_id_frags[2];
      double innerSpeed = 0.0;
      innerSpeed =
          Node.calcDist(segsMap.get(innerSeg_id).startNode,
          segsMap.get(innerSeg_id).endNode)
             / 1000.0 / (times[pure_index] - times[pure_index - 1]);
      double innerTime = (times[pure_index-1] + times[pure_index])
          / 2.0;
      Record newInnerRec = new Record(dev_id, firstRec.month,
          firstRec.day, innerTime, innerSpeed, innerSeg_id);
      newTraj.add(newInnerRec);
    }
```

Output Interpolation Result: The results of interpolation phase unlike in the previous mapmatching phase are categorized by time into 24 hours and written into file `train|test_interpolation_result.txt`. This function

1. Adds temporal info to Records matched to segments - Divides segments into 24 hours.
2. Goes through all the interpolated trajectories
3. Goes through all the records in a interpolated trajectory and performs this operation:

```
    fw_result.write(seg_id_split[0]+"_"+seg_id_split[1]+"_"+(int)rec.time
        + "," + (int)Math.round(rec.speed) + "\n");
```

This operation writes to a file: way_id + inner_id + recorded time/HOUR. For example speed observation at segment 0 in way 22371280 that was recorded on midnight (24.00-01.00) with 75 km/h speed is written into the output file like this: 22371280_0_1, 75. The previous output is generated for the training of the accident detection model. If, however, Test is specified for the interpolation, the output is going to look different. More specifically the output will have additional information like the exact time and the date of the recorded observation.

4.6.3 Input and Output files

Input

- `data/map_matching/fold_1..4/train_match_result`

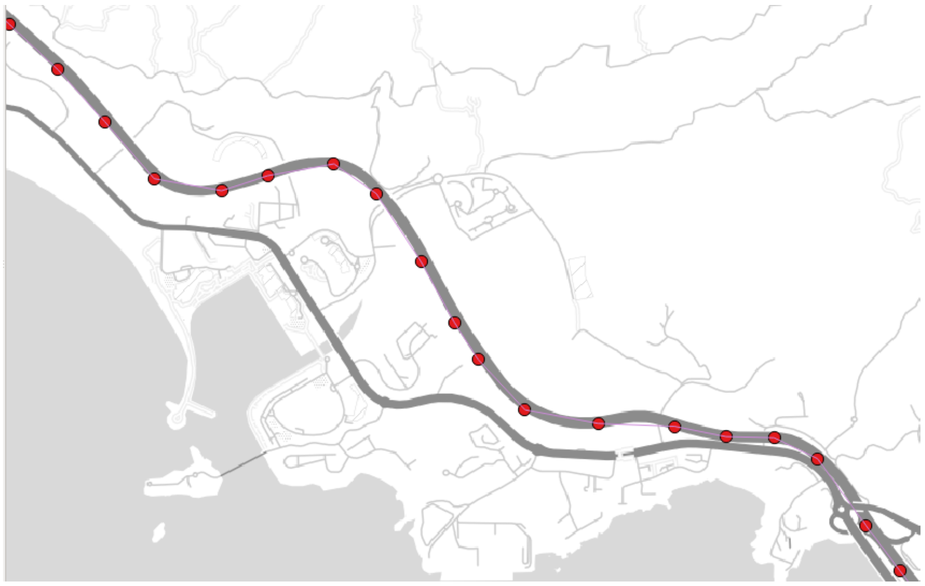
Output

- `data/linear_interpolation/traininterpolation_result.csv`

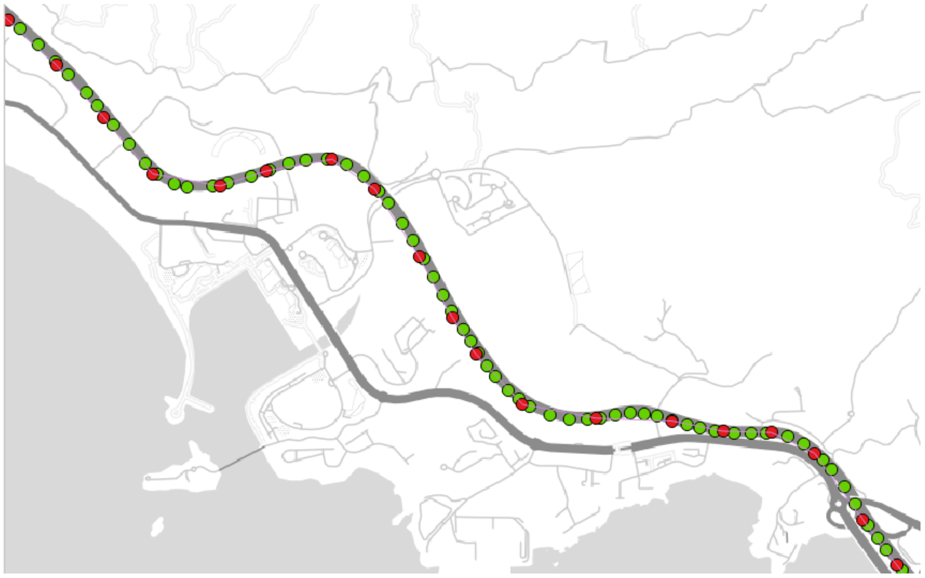
4.6.4 Test and Analysis

- `main.java.bgt.Analysis public void getHistorgramOfTrajectoriesAndLongest(int sampleSize)` As trajectory generation for linear interpolation is one of the most complicated parts in this project both technically and conceptually, thorough testing functions needed to be generated. The following function mainly does two things

  1. Generation of two histograms containing trajectory lengths. The trajectories are generated for a number of (sampleSize) devices.
     - Outputs `plots/trajHistogram.txt` - Goes through each trajectory for each device and generates a file which in rows has lengths for each trajectory.
     - Outputs `plots/size2biggerTrajSegmentHistogram.txt` - Also goes through each trajectory, but does not take into account trajectories which have their ending and starting positions with the same location coordinates. While going through the trajectories this function also chooses (default) from the devices trajectory list N the longest trajectories using the function `getXlongestTrajs(HashMap<Integer, List<List<Record>>> trajsMap`.
  2. Longest trajectory print to osm format. It writes the N longest GPS coordinates into a file `Trajectories.osm`. It also generates a file called `Ways.osm` which stores the ways the trajectories were assigned to. In addition, this function acts as a test to verify that one trajectory is assigned to only one way. To generate the .osm files this function uses the function `generateOSMforSegmentAndTraj(List<List<Record>> trajList)`. An example output of this visualisation can be seen in Table 16.



(a) A trajectory generated by a taxi travelling at around 90 km/h. Due to high speed the GPS has not recorded this observation in every segment



(b) With interpolation this issue can be solved. In this example, red dots stand as GPS observations and green dots in between are observations that are added through interpolation.

4.7 Step IV: Speedpanel Data Integration

4.7.1 General

- Running time - 20 minutes

One of the key contributions of this dissertation was to integrate speedpanel data with GPS data. The speedpanel dataset consists of average speed observations in 2 minute time intervals. As the taxi dataset was available for year 2010, only speedpanels that were installed in year 2010 have been integrated as a data source to this project. The distribution of speedpanels in Hong Kong can be seen in Figure 6.

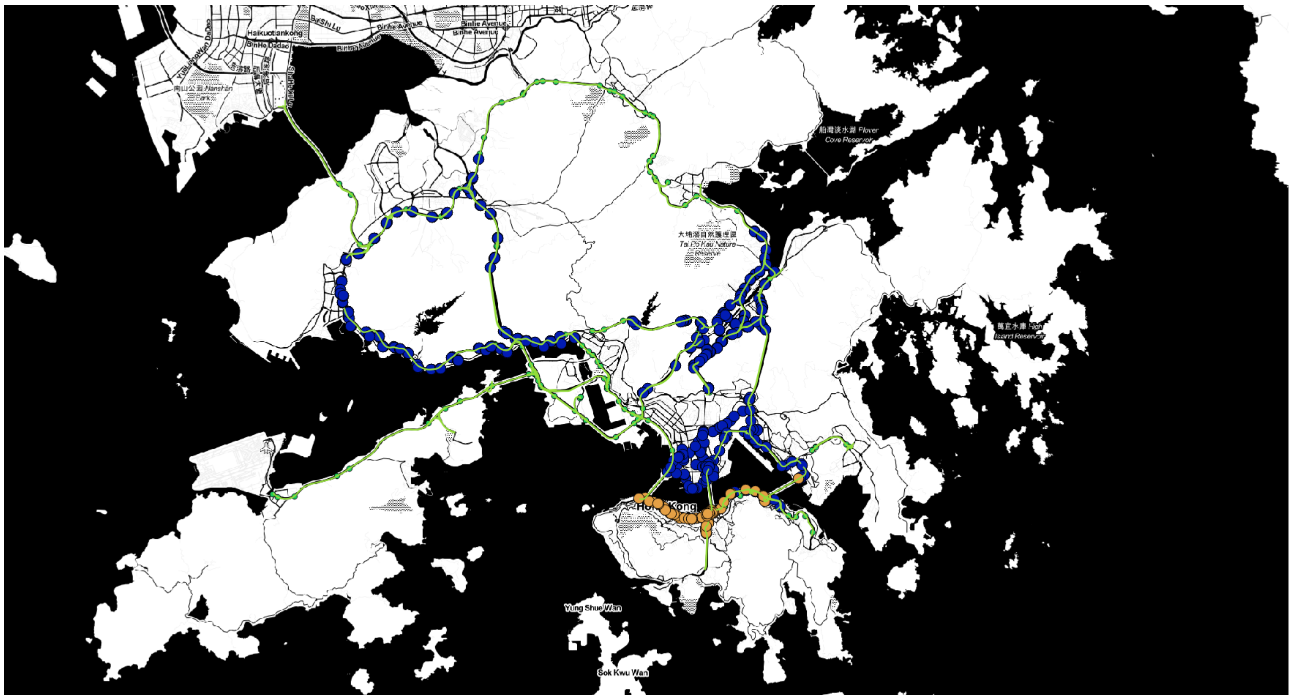


Figure 6: The speedpanels that are available before year 2013 are shown in color beige. The speedpanels that are available after year 2013 are shown in the colors blue and beige.

4.7.2 Code

Files

Data Models

- bgt.Model.Grid

Link ID	Start Node	Start Node Eastings	Start Node Northings	End Node	End Node Eastings
722-50059	722	834038.674	816345.067	50059	833862.7
724-722	724	834148.783	816250.647	722	834038.674
752-875	752	835099.22	815634.373	875	834918.334
756-752	756	835352.749	815640.774	752	835099.22
760-756	760	835602.186	815600.695	756	835352.749

Table 7: Speedpanels specifications

- bgt.Model.GridElement
- bgt/Model/Node.java
- bgt.Model.Record
- bgt.Model.Routes
- bgt.Model.Segment
- bgt.Model.Way

Controllers and Interfaces

- bgt.SpeedPanel.AppendSpeedPanel
- bgt.SpeedPanel.MatchSpeedPanelSegments
- bgt.SpeedPanel.SpeedPanelToProjectFormater

Execution: Initially, speedpanel locations were stored in the form shown in Table 7.

1. From this format the locations of speedpanel start and end points in HK_1980 coordinate system were extracted. As this project is based on WGS84 coordinate system, conversion needed to be performed using Hong Kong Government webpage. Next, the speedpanels were transformed into OSM format where start and endpoints of the speedpanels were stored as Nodes and speedpanels were represented as Ways. For this processing `bgt.SpeedPanel.SpeedPanelToProjectFormater` was used.

2. After transforming speedpanel data into OSM format, the speedpanel based Ways were matched to segments generated in the Route Segmentation phase. For this, exactly the same function was used as in map matching and when assessing closeness it took into account the absolute distance between segments start and end points and the segments directions. After automatic map matching, manual filtering was done to eliminate false matches. This process was done using `bgt.SpeedPanel.MatchSpeedPanelSegments extends MapMatcher`. The general steps done by this source code are as follows:

(a) MatchSpeedPanelSegments matchSpeedPanelSegments = new MatchSpeedPanelSegments("speedPanelWays.osm"); - reads in speedPanels file as MapMatcher base map.

(b) matchSpeedPanelSegments.indexSegs(); - indexes the base map just like in GPS observations matching

(c) matchSpeedPanelSegments.doMatching(); - matches Hong_Kong-result.osm segments to speed_panel based ways map.

3. The previous phase resulted in a file which had for every speedpanel ID listed all the segments that were matched to it. This file was read in by `gt.SpeedPanel.AppendSpeedPanel ; bgt.SpeedPanel.SpeedPanelObsParser` which does two things

   - speedPanelParser.appendAllObservations(GenerationType.TRAIN); - builds input for model training phase and appends this to linear interpolation result.
   - speedPanelParser.appendAllObservations(GenerationType.TEST); - builds input for labelling phase, which the labelling phase must read in separately.

4.7.3 Input and Output files

Input

- `/data/speed_panels/tsm_dataspec.pdf`
- `/data/route_segmentation/Hong_Kong-result.osm`

Output

- `oFileName+"speedPanelWays.osm"` : Generated by `bgt.SpeedPanel.SpeedPanelToProjectFormater`. This is a file which contains speepanel pairs as ways, each way is built from two nodes
- `data/speed_panels/SpeedPanelSegments.osm` : Generated by `bgt.SpeedPanel.MatchSpeedPanelSegm` : osm result file to visualise the segments and the corresponding speedpanels.
- `data/speed_panels/spMatched.txt` : Generated by `bgt.SpeedPanel.MatchSpeedPanelSegments` : File which lists the speedpanels with the matched segments.
- `data/speed_panels/append_train/train_interpolation_result1|2|3|4.txt` : This file consists of the interpolated GPS records and the observations from the speedpanels. This file is ready to be inserted into the training of the model.
- `data/speed_panels/labeling/fold_1|2|3|4/<seg\usid>.txt` : File with speedpanel observations for input to the labeling phase.

4.7.4 Test and Analysis

No automatic tests were designed for this phase. However `bgt.SpeedPanel.MatchSpeedPanelSegments` outputs a .osm file which visualises how well the speedpanels were matched to the segments.

4.8 Step V: Labelling

4.8.1 General

- Initial running time for labelling was less than ten minutes. This was due to a mistake in the code and the fact that interpolation phase generated a small amount of data. After fixing the mistake in the labelling phase it also needed to be redesigned because the the old code was not able to run with the amount of data linear interpolation generated. The total running time with the largest dataset - 5 GB - is around one hour.

To test the incident detection method, observations that were recorded in the close vicinity of an accident both time and distance-wise needed to be labelled with accident flags up. This was done in the labelling phase.

4.8.2 Code

Files

Data Models

- bgt/Model/Node.java
- bgt.Model.Record
- bgt.Model.Accident
- bgt.Model.Routes
- bgt.Model.Segment
- bgt.Model.Way

Controllers and Interfaces

- bgt.LabelData.LabelData

Execution: First a MapMatcher object named mm is generated. This mm object is going to be used for its data structures, namely HashMap with segment information. After this a new Labeldata object `ld` is generated and ld.performlabelling(int foldNr, Boolean addSpeedPanels, Boolean addGps) is called. The function ld.performlabelling() is responsible for everything in this phase. This function, one way at the time

1. reads in all the Accidents to the Segments `accList`
2. reads in all the observations from the GPS dataset to the Segments recordList
3. reads in all the observations from the speedpanel dataset to the Segments recordList
4. sorts both the `recordList` and `accList`
5. goes through each Segments recordList and raises the Records accident flag if it has an accident in the close vicinity.
6. after each Record has been checked for Accident Occurence in the close vicinity the result is going to be written out for each way in a separate file as a stream of speed observations with the accident flag raised or not raised.

4.8.3 Input and Output files

Input

- `data/linear_interpolation/fold_1|2|3|4/test_interpolation_result.csv` - GPS records for labelling
- `data/speed_panels/labelling/<seg-ID>.txt` - speed panel observations for labelling

Output

- `data/label_data/fold1|2|3|3/label_result_<way-ID>.txt` Result file of labelling. This file tries to simulate a real traffic stream for each segment. The speed observations in the labelling phase are grouped by segments. In each segment the observations are grouped by 24 hours and there is a stream of observations for each date in the specified hour. Each observation is classified as having an accident or not having an accident. The general order looks like this
  - Segment 1
    - Hour 0
      - Observations stream for 1 January 2010
      - Observations stream for 2 January 2010
      - ...
    - Hour 1
    - ...
  - Segment 2
  - ...

5 Topic Modeling and the Experiment

5.1 Topic Modeling

5.1.1 General

The derivation of the model and its approximation algorithm, and the discussion about the feature is based on multitude of different books, , tutorials and articles [1] [5] [2]. Each idea and description proposed is mostly a mixture of ideas derived from different sources.

The specific formulas for training the incident detection model and performing incident detection are given in Kinoshita et al. [18]. A Latent Dirichlet Allocation (LDA) equivalent method is used for modeling the traffic. First of all, the implemented method divides all the roads in Hong Kong into `S` segments. A segment is specified as a certain section of a route during a certain period of time. Next, speed observations obtained from GPS and road sensor data are used to train a traffic model. The result of the training will be `K` global traffic states distributed in each segment according to the segment's local parameters. After the training of the model, each new observation is compared to the prior historical distribution in the segment using divergence functions.

5.1.2 Definitions

- S - number of segments
- K - number of traffic states
- N - number of speed observations
- Z - latent random variable - binary vector with ones and zeros and K elements (only one element = 1)
- X - observed random variable - scalar value, a speed observation in this case. (could be extended to be a vector)

5.1.3 Topic Modeling Theory

The topic model fitted on the traffic data in the article Kinoshita et al. [18] is derived from a probabilistic topic modeling technique. Topic modeling with latent variable mixture models is a broadly covered field in literature and has numerous applications in textual analysis, traffic modeling and genetic engineering. [35] [8] [39]

The mixture of different models with latent variable parameters was first proposed by Blei et al. [6] in a paper describing Latent Dirichlet Allocation.

The aim of this methodology is to classify observations into topics. To do this, a generative probabilistic process is assumed to exist behind every observation. That means every observable observation is drawn from a certain latent random variable distribution or distributions. The goal is to reverse this process by inferring these latent random variables from the observed data.

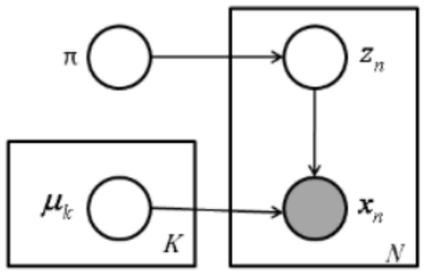


Figure 7: The graphical model for speed observation generation. The observed elements are shown in gray.

Model Behind each observation: In the generative model, every random variable except for the observation is considered to be latent. First, for each segment,

1. a random variable Z (binary vector with only 1 element = 1) is drawn from a multinomial distribution which is dependent on parameter π. The multinomial distribution represents the probability over K different global traffic states in segment `S`
2. based on the drawn Z, an observation is generated from the Poisson distribution specified by Z. Each Poisson distribution over speeds is dependent on a global parameter μ

This process can be visualised with a simple graphical model shown on Figure 7. Equivalently this model also means that every segment is considered as a mixture of traffic states which by themselves are distributions, thus every segment is a mixture model.

The likelihood function for observations of speed-values for the entire set of observed data is given by the following equation:

$$L(\mu, \pi; X) = \prod_{1}^{S} \prod_{1}^{N_s} \sum_{1}^{K} p(x_{sn}|\mu_k)p(\pi_{sk})$$

It can be interpreted as: For each road segment, for each observation, what is the combined likelihood of drawing observations X with parameters μ and π.

There are two significant parameters in this model. The local segment parameter π, which affects how traffic states are distributed in each segment, and the traffic state parameter μ for each of the K traffic states. Having established this generative model to the data, there exist multiple techniques to optimise the parameters of this model to the data. It is important to make a difference between the random variables (X -observed , Z-latent) and the point estimated parameters (μ and π).

5.1.4 Bayes background

Theoretically the most straightforward approach to infer the parameters from the observed data is to use Bayesian inference. This can be thought of as reversing the generative process.

General Example of Bayes formula in probability Theory

$$P(A|B) = \frac{P(B|A)P(A)}{\sum P(B|A)P(A)}$$

In Bayesian inference, It is supposed that there is a distribution over the latent random variables and data is fixed. The main aim of Bayesian rule is thus to calculate the probability of parameter/s based on the given data. It is often done incrementally, i.e. through improving/deepening the belief in certain parameters/ infered latent variable as the amount of data "going through" the calculations grows.

$$\overset{\text{Posterior}}{P(\theta|data)} = \frac{\overset{\text{Likelihood}}{P(data|\theta)}\overset{\text{Prior}}{P(\theta)}}{\overset{\text{Marginal}}{P(data)}}$$

All the elements in the above formula are dependent on a chosen model. The model becomes important for the margin to explain what the probability of data is. It is the probability of the data based on the model and chosen parameters. Here, the Bayesian approach would also theoretically work out. To do that, it would be necessary to consider the parameters μ and π in addition to the segment distribution variable Z as random variables. The general application of the Bayes formula in this case would look like this

$$p(\theta, Z|x) = \frac{p(\theta, z, x)}{p(x)}$$

In this formula θ is considered as the set of all parameters an Z is considered as latent random variables which show the cluster of or class of the observation (e.g. which Poisson distribution generated the observation). The likelihood of the data in the above formula's denominator which is also considered the normalising factor or the marginal becomes intractable to compute in real life with more complex models. Thus approximation methods like the Variational inference or Monte Carlo sampling should be used [6]. In this case a more simple approximation was chosen in the face of the Expectization Maximization (EM) algorithm. This method resembles more the variational inference based methods but only gives point estimates to the parameters μ and π.

5.1.5 Approximation Methodology

The goal for the used EM algorithm is simply to give a point estimate for the parameters μ and π from the observations so $p(x)$. The straight log-likelihood maximisation for the parameters $p(X|\theta) = \sum_{i=1}^{N} logp(x, \theta)$ is difficult because of all the possible classes each observation could come from. However the nature of the above-presented model allows to divide the parameter estimation into two steps. More explicitly we can repeatedly construct a lower-bound on the log-likelihood function (E-step), and then optimise that lower-bound (M-step). This approach follows the logic of the EM algorithm. The exact formulas for approximation in the E-step and M-step are brought out in the method-proposing paper and are left out from this dissertation for the sake of clarity. The general idea for the approximation goes like this: First ther is the E-step. In this step, the Expectation for the complete dataset (X,Z) is calculated. To do that, the posterior probability of Z given X is calculated. This can be seen as the responsibility that component k takes for explaining the observation x. The calculated posterior probability for Z used in a specific combination with the parameters likelihood allows through the usage of Jensen's equality to derive a lower bound for the likelihood function. This lower bound can then maximized in the M-step. Going iteratively through E-step and M-step monotonically increases the likelihood function.

5.1.6 Detecting Anomalies

After the parameters have been inferred globally and for each of the segments, new observations will be compared with the historical data based model. To do that, Bayesian inference is used again inside a specific time window to build a new distribution for the specific local segments using again the global traffic state parameters. Then the new posterior distribution is compared with the old distribution using weighted KL divergence.

5.2 Initial Experiment

5.2.1 Experiment flow

This method was initially only tested on probe car data. The Experiment setup was very similar to experiment done in Kinoshita et al. [18]. And overall consisted of five steps. (Figure 1)

1. Segment Generation: All the segments are same in regards to the size and time window.

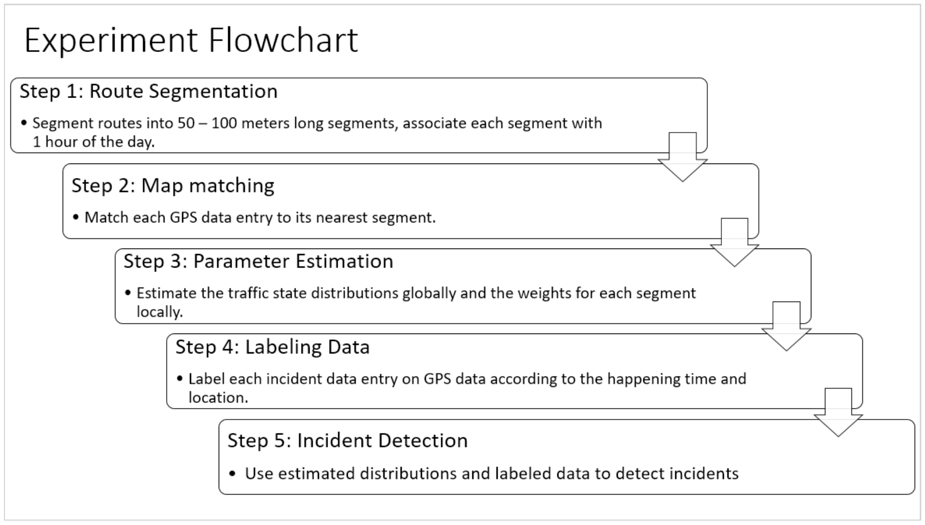
2. Map matching

3. Parameter Estimation: The EM algorithm was used for parameter estimation.

4. Accident information labelling: Data used from the Hong Kong Police Department to label segments anomalous or nonanomalous.

5. Incident Detection: Calculate the anomaly value for a segment as the average of the degree of an anomaly of all cars passing over the segment in the time window.

Figure 8: Flowchart of experiment



Experiment Flowchart

Step 1: Route Segmentation

- Segment routes into 50 – 100 meters long segments, associate each segment with 1 hour of the day.

Step 2: Map matching

- Match each GPS data entry to its nearest segment.

Step 3: Parameter Estimation

- Estimate the traffic state distributions globally and the weights for each segment locally.

Step 4: Labeling Data

- Label each incident data entry on GPS data according to the happening time and location.

Step 5: Incident Detection

- Use estimated distributions and labeled data to detect incidents

5.2.2 Evaluation

To evaluate the performance of the algorithm, a 4-fold cross-validation experiment was designed. The dataset was partitioned into 53 weeks, 13 of which were considered as testing data. (Figure 2)

Detection Rate (DR) and False Alarm Rate(FAR) were measured and plotted the receiver-operating curve (ROC) to review the result for each road. We also calculated the area under the ROC curve (AUC) for each road. In et al. Kinoshita [18] the experiments conducted resulted in very high accuracy model with the highest AUC of 0.957. (Figure 10)

In Hong Kong our best result was close to Japan's results, however overall the DR and AUC was very low. (Figures 11 and 12) and the results were also extremely sparse as can be seen in Figure 13. This was caused from multiple bugs in the initial project's source code.

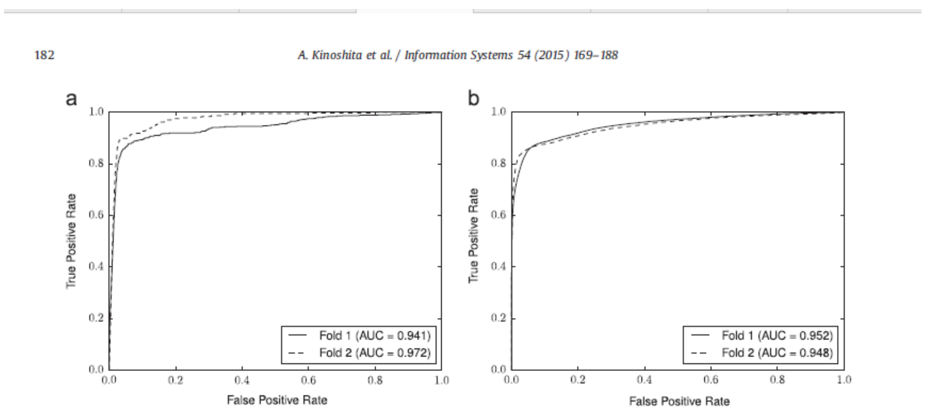


Figure 10: Results for Tokyo

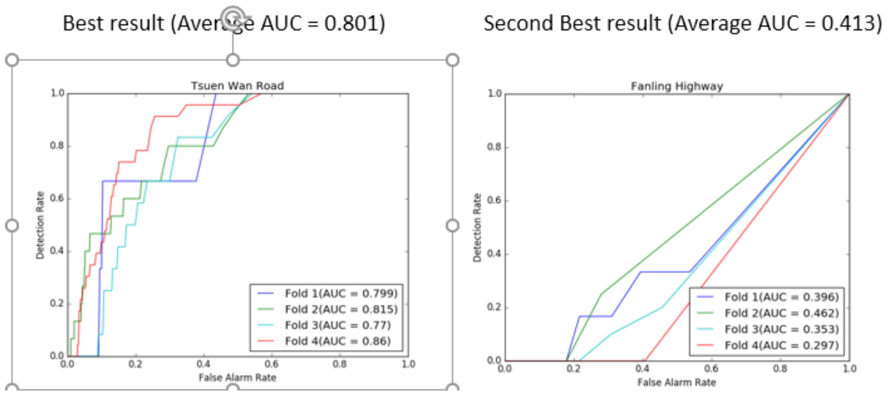
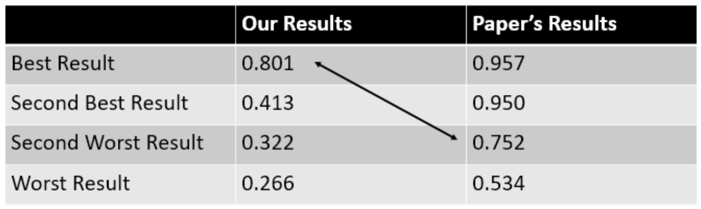


Figure 11: Results for Hong Kong



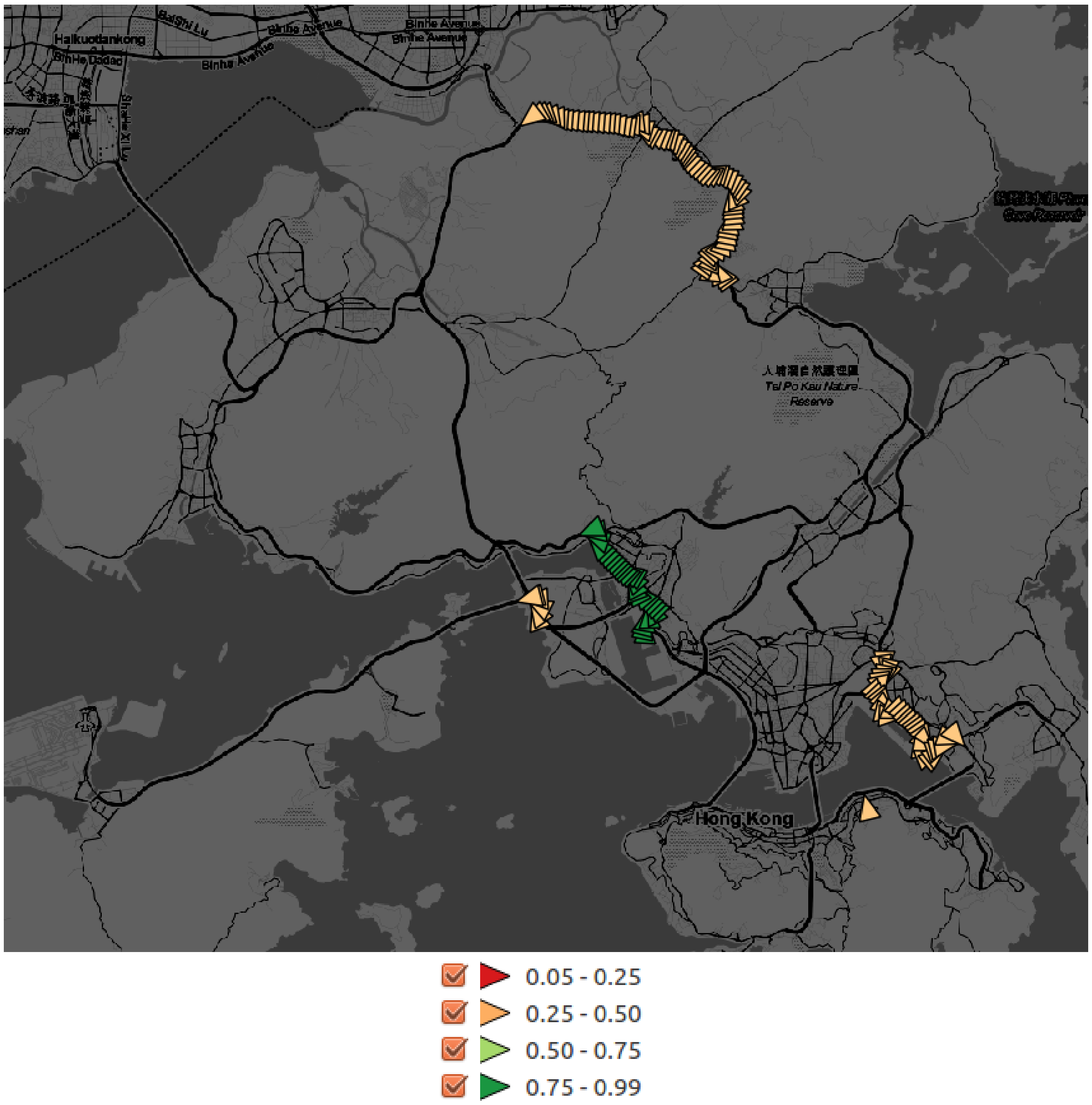
	Our Results	Paper’s Results
Best Result	0.801	0.957
Second Best Result	0.413	0.950
Second Worst Result	0.322	0.752
Worst Result	0.266	0.534

Figure 12: Comparison

5.3 Experiment with the Redesigned Project

5.3.1 Route segmentation

50-100 m length was chosen for the specified segment length. This was done based on the underlying article. The total number of ways consisting of segments under analysis is 820. The longest way is 16 km and has 294 segments. The shortest way consists of 2 segments and is in total 5 m in length. A sample of shortest and the longest ways can be seen in Tables 8a and 8b



(a) AUC values visualized

Figure 13: The accident detection rate visualized on top of Hong Kong Map for accident detection model built with the old project source code based on GPS data.

WAY NAME	# SEGS	# 50 M SEGS	LENGTH (M)
Tuen Mun Road	294	234	16385.61029
Tuen Mun Road	292	232	16249.97835
North Lantau Highway	171	184	13107.83674
North Lantau Highway	168	175	13089.73244
Tolo Highway	192	148	11048.8303
Tolo Highway	199	156	11027.57572
Fanling Highway	147	148	10867.83975
Fanling Highway	151	147	10848.08142
Yuen Long Highway	173	146	10179.26136
Yuen Long Highway	184	143	9888.009811
San Tin Highway	68	83	6202.393162
San Tin Highway	66	79	6165.426254
Kong Sham Western Highway	98	82	5544.233737
Hong KongShenzhen Western Corridor	33	72	5526.422509
Tai Lam Tunnel	94	70	5438.935537
Tai Lam Tunnel	87	72	5378.011619
Hong KongShenzhen Western Corridor	33	71	5272.977185
Kong Sham Western Highway	88	77	5254.712841
Tate's Cairn Tunnel	72	65	5212.883567

(a) Longest Ways

WAY NAME	# SEGS	# 50 M SEGS	LENGTH (M)
Aberdeen Praya Road	2	2	5.372700376
Chatham Road North	2	2	6.332794367
null	2	2	6.345545839
Island Eastern Corridor	2	2	6.856716348
Chatham Road North	2	2	9.990102505
Connaught Road West Flyover	2	2	10.84602047
Eastern Harbour Crossing Approach	2	2	11.11539192
null	2	2	12.32904788
null	2	2	13.18528737
Canal Road Flyover	3	2	13.32475661
Gloucester Road	2	2	13.51179995
null	3	2	13.71445326
null	2	2	13.90960906
null	2	2	13.95621544
Canal Road Flyover	2	2	14.42187093
Tong Shui Road	4	2	14.48736221
Tong Shui Road	4	2	14.52567316
Canal Road Flyover	2	2	14.61963583

(b) Shortest Ways

5.3.2 Map Matching

In the experiment a record needed to be closer than 25.1 m from the segment and had to have less than 45 degree direction difference to be assigned to a segment. This rather conservative bound was chosen based on visual observations. This difference can be seen at Figure 15. Although the visualised example shows matching speedpanel based ways with osm based segments the logic can be extended to GPS data. The amount of GPS records assigned to segments with bound 25 is around 12-13 percent of all the data. With a bound of 50 m this was 14-15 and with a bound of 100 m the percentage was around 18-19. The maximum amount of records assigned to a way was 752,321 and the smallest amount of records was 0. The overall distribution of data assignement to segments (50-100 m) can be seen in Histogram 14. If comparing the record assignements with the data shown in Kinoshita et al. data based on Japan highways, difference per 1 km varies notably, however some ways in that paper which received good results have fewer observations per 1 km than in the segments in this project. Table 9 shows a sample of the map matching phase results for ways longer than 1 km and a comparison with the data.

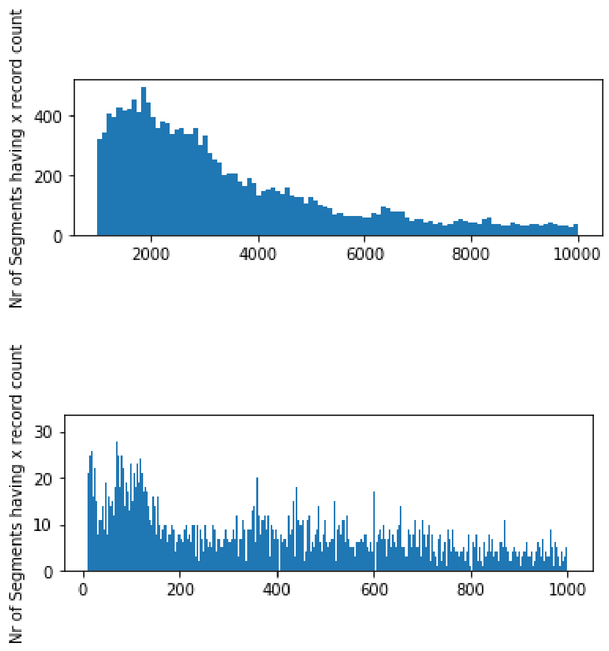
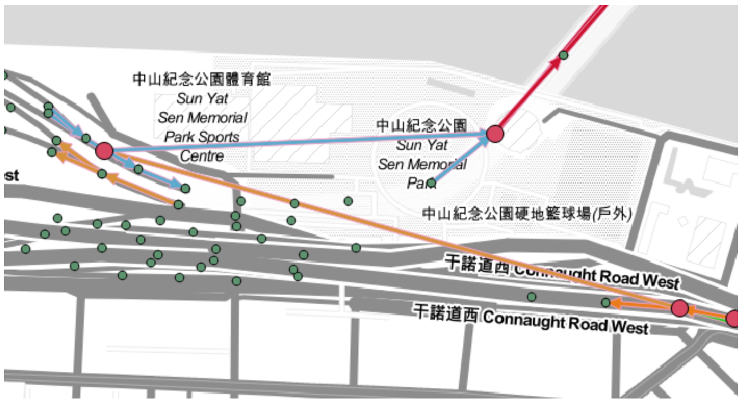
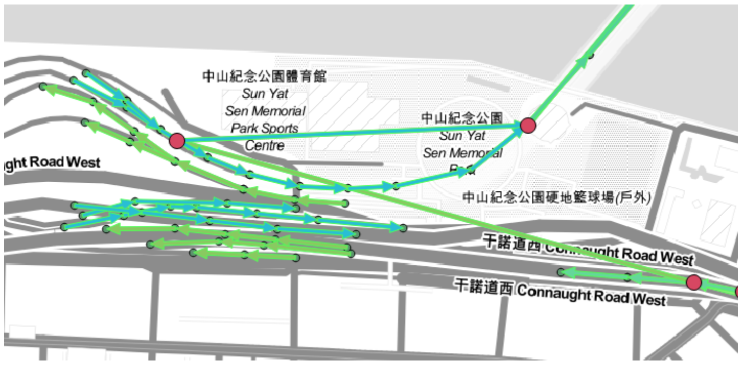


Figure 14: histogram of number of observations assigned to 50-100 m segments



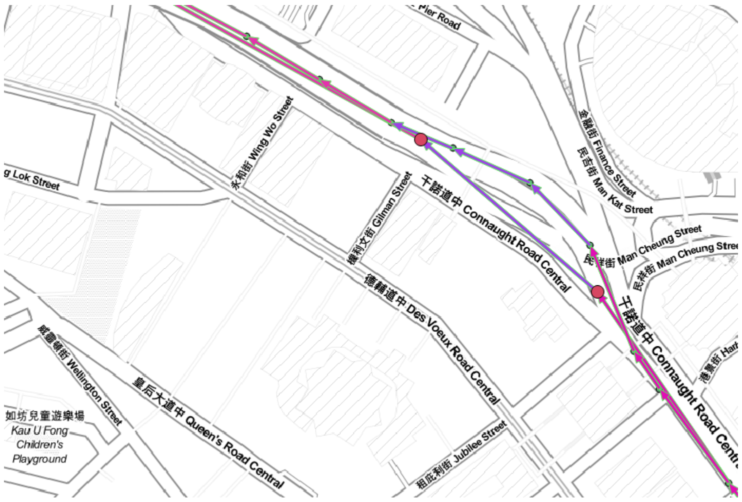
中山紀念公園體育館
Sun Yat Sen Memorial Park Sports Centre
中山紀念公園
Sun Yat Sen Memorial Park
中山紀念公園硬地籃球場(戶外)
干諾道西 Connaught Road West
干諾道西 Connaught Road West

(a) 25 m



中山紀念公園體育館
Sun Yat Sen Memorial Park Sports Centre
中山紀念公園
Sun Yat Sen Memorial Park
中山紀念公園硬地籃球場(戶外)
ght Road West
干諾道西 Connaught Road West
干諾道西 Connaught Road West

(b) 100 m



(c) less complex segments produce similiar results for 25 m and 100 m bounds

Figure 15: The lines between red dots are speedpanels and all others stand as regular segments. The segments with the same color as speedpanels are assigned to the speedpanel. The 100 m bound generates a lot of errors with more complex segments. The 25 m approach misses some matches in more complex segments but that approach is preferred to not erroneously assign a very large number of observations to wrong segments.

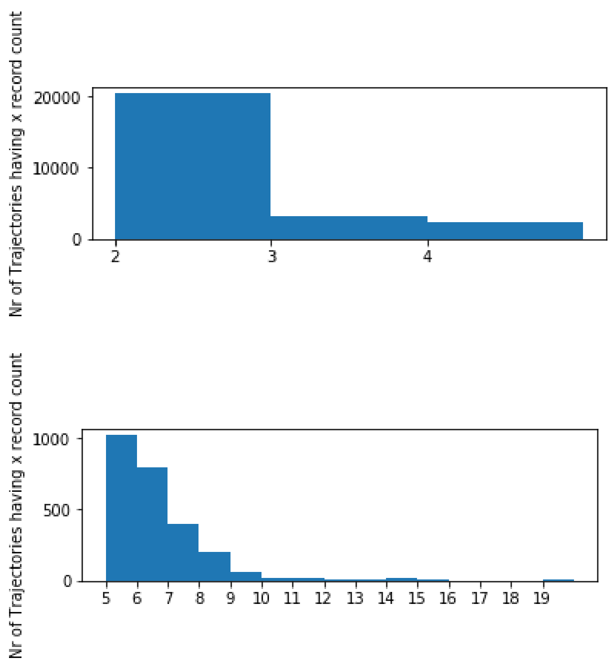
WAY NAME	LENGTH (M)	GPS OBS	OBS PER KM	JAPAN OBS PER KM
East Kowloon Corridor	1301.254729	463605.75	356275.938652131	1496684.375
Princess Margaret Road	1029.217407	300149.25	291628.617975755	1414756.25
East Kowloon Corridor	1298.306558	287871	221728.064320538	1287201.282
Kai Tak Tunnel	1280.977129	270565.5	211218.056805759	1284097.583
Kwun Tong Bypass	4915.807183	695085.75	141398.090715146	1276124.265
Tsing Kwai Highway	4276.513076	464729.25	108670.134228768	1267284.071
Tsuen Wan Road	4192.699929	369478.5	88124.2412423548	1247595.929
Tsuen Wan Road	3993.547231	345702.75	86565.3340259694	1226469.405
Tsing Kwai Highway	4265.844522	362262	84921.5197909175	1220469.431
West Kowloon Highway	4842.6111	372656.25	76953.5777919478	1186850.171
Tate's Cairn Highway	4910.23785	316707	64499.3195187072	. Bottom half
North Lantau Highway	13089.73244	752321.25	57474.1503272469	495843.6667
North Lantau Highway	13107.83674	705863.25	53850.4761694186	409887.8947
Tolo Highway	11027.57572	411754.5	37338.6236879995	289826.2791
Tolo Highway	11048.8303	403686.75	36536.6051463384	159418
Tuen Mun Road	16385.61029	508936.5	31059.9660917482	150724.5
Tuen Mun Road	16249.97835	428970	26398.1890166641	145031.5

Table 9: Map matching results for ways

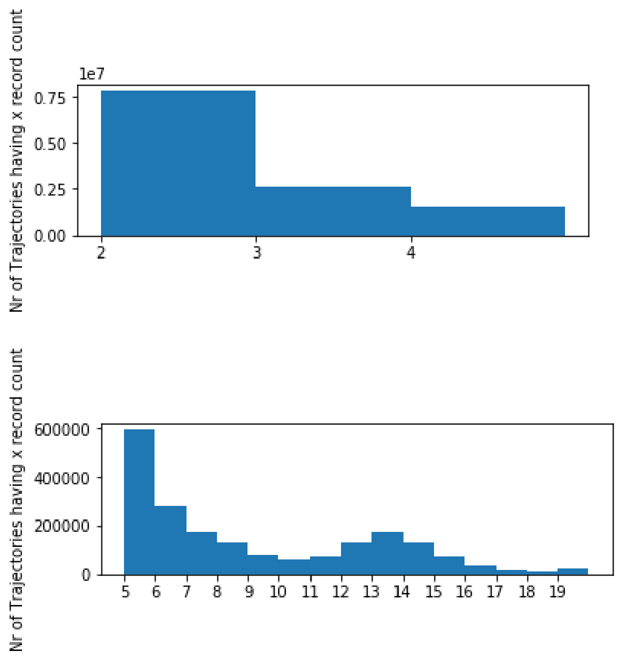
5.3.3 Linear interpolation

The linear interpolation phase matched only one observation in each trajectory with a specific segment. This in some cases led interpolation resulting in fewer observations in a segment than without it. This sort of elimination was necessary because there existed a lot of long trajectories with the speed 0 and with the same start and end segments.

Because many ways under the current dataset are extremely short (less than 100 meters), and the trajectory generation bases itself on ways, there exists a large number of length one trajectories. Longer ways, however, were able to create a large amount of long trajectrories. The distribution over trajectory lengths can be seen in Figure 16b. Updating the source code for new trajectory building methodology introduced significantly better results as can be seen in the comparison with Figure 16a. The longest trajectories were generated, as one would assume, on the longest and straightest ways. An example of this can be seen in Figure 17. For some longer ways intepolation added a very large amount of observations. This can be seen in Table 10 After interpolation for all the roads on average 93,764,615.25 observations were matched. This is three times more than without interpolation and also three times more than with interpolation with the older project.



(a) Histogram of trajectory lengths for trajectories generated by the old source code



(b) Histogram of trajectory lengths for trajectories generated by the updated source code (1e7 = 10000000)

Figure 16: As can be seen in the comparison of these two histograms, the new method of trajectory generation produces significantly better results.

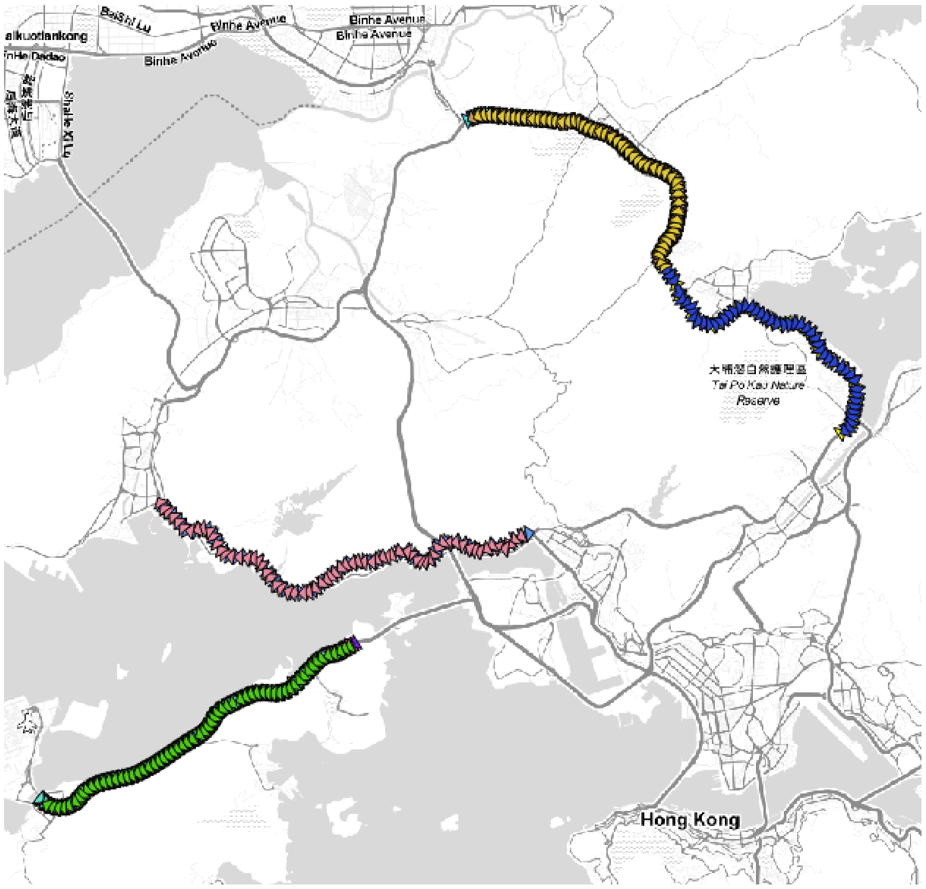


Figure 17: Longest trajectories generated in the dataset. Routes: Fanling Highway, Tolo Highway, Tuen Mun Road and Norh Lantau Highway.

WAY NAME	LENGTH (M)	GPS OBS	INTERPOLATED OBS	INTERPOLATED OBS / GPS OBS
North Lantau Highway	13107.83674	705863.25	9093941.25	12.8834321
North Lantau Highway	13089.73244	752321.25	8700963	11.56548881
Hong KongShenzhen Western Corridor	5272.977185	6821.25	65409.75	9.589114898
Tuen Mun Road	16249.97835	428970	4070546.25	9.489116372
Tuen Mun Road	16385.61029	508936.5	4809003.75	9.449123319
Tolo Highway	11027.57572	411754.5	3883506.75	9.431607305
Kong Sham Western Highway	5254.712841	9479.25	89049	9.394097634
Fanling Highway	10848.08142	151314.75	1412535	9.335078041
Tolo Highway	11048.8303	403686.75	3717345.75	9.208490866
San Tin Highway	6202.393162	41289	378768.75	9.173599506
San Tin Highway	6165.426254	36022.5	324594	9.010868207
Tsing Long Highway	4708.441113	44851.5	396926.25	8.849787633

Table 10: Interpolation introduced a lot of new observations for some ways

5.3.4 Addition of Speedpanels Data to the Dataset

Alltogether there were 62 speedpanels installed in the year 2010. 60 speedpanels were matched with a total of 250 segments (with each segment belonging to one speedpanel). In addition, some manual filtering was done to ensure the best results. The addition of observations to some of the ways can be seen in Table 11 with some ways having 4,000 percent more observations compared to only having interpolated GPS data. Altogether addition of speedpanels doubled the dataset from 30 million to 60 million observations.

WAY NAME	LENGTH (M)	O+I	O+I+SP/O+I	OBS IN KM	O+I+SP/JAPAN
Eastern Harbour Crossing	2279.4351	4719277.5	448.110668	2070371.514	1.383305357
Western Harbour Crossing	1994.91239	3152508.75	236.2358793	1580274.285	1.055850058
Cross-Harbour Tunnel	1846.835837	1410518.25	37.9845492	763748.581	0.5102936823
Island Eastern Corridor	861.7992294	410932.5	22.16823111	476830.8975	0.3185914849
Island Eastern Corridor	745.1039265	637176	13.1152724	855150.5063	0.5713632885
Island Eastern Corridor	621.1596679	1296850.5	10.83743231	2087789.287	1.39494293
Rumsey Street Flyover	734.6134674	2216259	8.715769728	3016904.942	2.015725554
Gloucester Road	532.6508503	1629403.5	6.36437885	3059046.088	2.04388189
Island Eastern Corridor	870.1400171	563577.75	3.292267452	647686.2792	0.4327474048
Island Eastern Corridor	585.5226051	571883.25	3.186196551	976705.6728	0.652579588
West Kowloon Highway	4842.6111	3709018.5	1.464954428	765912.939	0.5117397842

Table 11: O - GPS Records matched with segments; I - Interpolated Records; SP- Speedpanel based records. The comparison with Japan compares best results of Japan's matched segments per 1 km.

5.3.5 Labelling

Redesigning the code for labelling and linear interpolation also allowed to match a significantly higher amount of accidents to the records and use a larger set of roads for testing the incident detection method. In fact, there were more than 500 times more results when including the data from the speedpanels and 300 times more when only the GPS data was used . The comparison for accident matching can be seen in Table 12

	OLD PROJECT	INTERPOLATED GPS	INTERPOLATED GPS + SPEEDPANELS
Records marked with accidents	13.25	6431.75	11278
Nr of accidents on segments	8.75	2760.5	2875.25

Table 12: The accident counts for all routes (Averaged over folds)

5.3.6 Training if the Model

The training of the model took around 5 minutes to 2 hours based on the input dataset. It took 15 minutes to train a dataset with 30 million observations, around 45 minutes for a dataset with 90 million observations and 2 hours for a dataset with 140 million observations. The amount of iterations the EM algorithm took varied from 10 to 26.

5.3.7 Incident Detection and Results

In summary, the experiments were conducted based on

1. Only GPS data - 25 m GPS record matching boundary
2. GPS data (same as previous) + speedpanel data
3. Only GPS data - 100 m GPS record matching boundary, boundary for used way lengths: 1,500 m
4. Only speedpanel data

The input data observation amount in each dataset can be seen in Figures 18

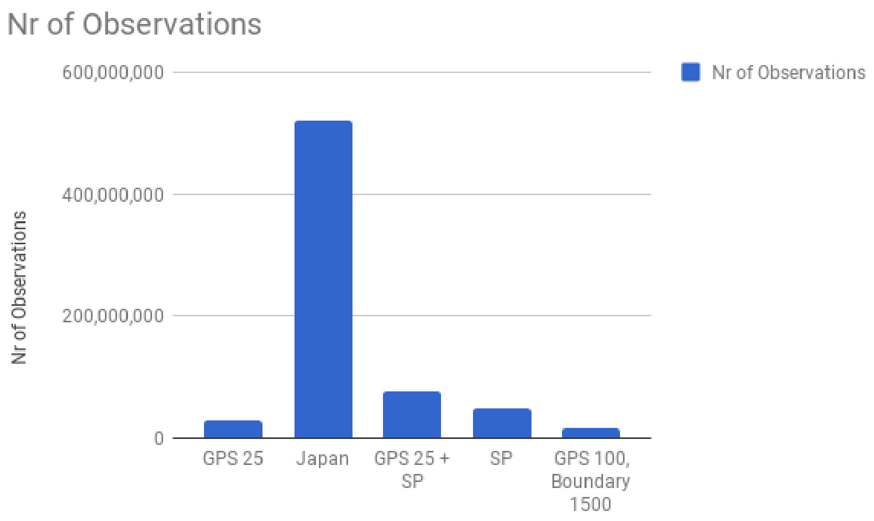


Figure 18: Histogram for dataset sizes, the tests with Japan dataset were conducted in Kinoshita et al.

All experiments were done using 60, 300, 600, 1,800, 3,600 second-length time windows for incident detection anomaly calculation. As the accident labelling results for the old source code are too sparse to be considered as ground truth, no analysis will be conducted with those results.

As experiments (3) and (4) resulted in AUC results less than 0.5, to save time and space, from this part on result analysis focus is directed only to experiments (1) and (2). In general, the input data for the model has a lot of variance and outliers so feature analysis is a very important part for this experiment. The fisker plots shown in Figure 20, 21 together with Table 13

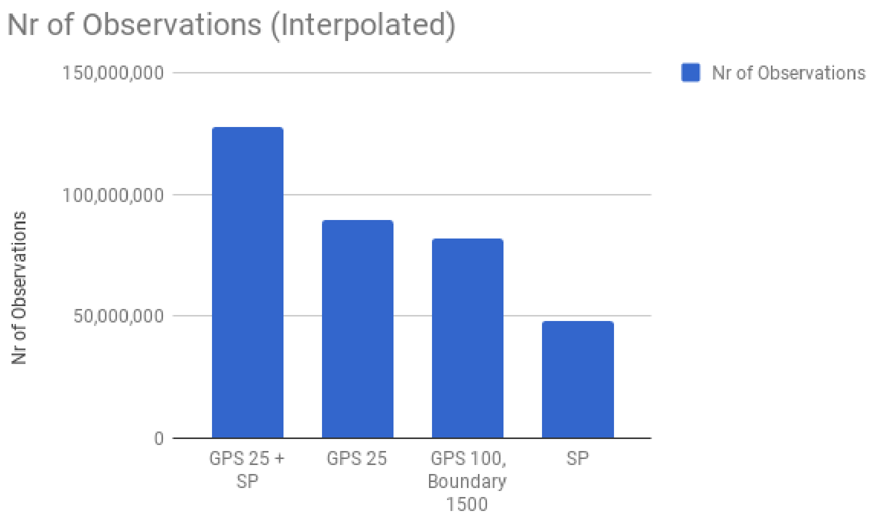


Figure 19: Histogram for datasets including interpolation. The tests with the Japan dataset are not included because the exact number of observations created with interpolation is not mentioned in their paper.

for all the different features show the data distributions well. Each feature was chosen based on observations of experiment results. In addition, new features were created based on the underlying data:

- (GPS + Interpolated Records) / GPS records - this feature was chosen based on the hypothesis that well-working interpolation should indicate long trajectories which in turn indicates a higher probability of correct observations.
- Accidents/ Accident records - this feature was chosen based on the fact that more observations during an accident should give a more accurate result.
- MULTIPLE - Accident Records / Max Accident Records * accident records / accidents * (GPS+interpolation)/GPS

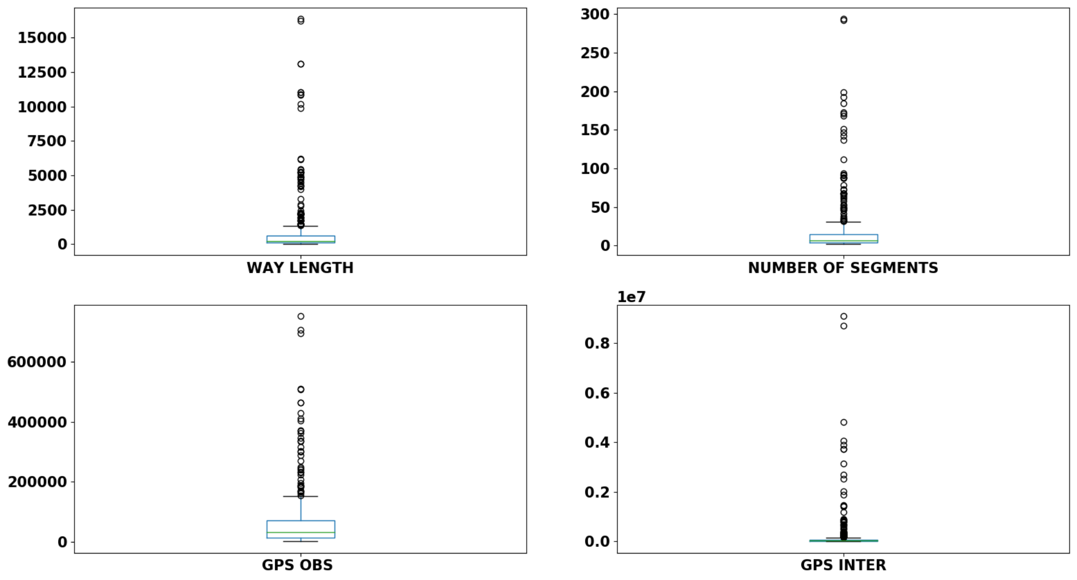


Figure 20: Fisker plots for features of data; GPS INTER - GPS records after interpolation

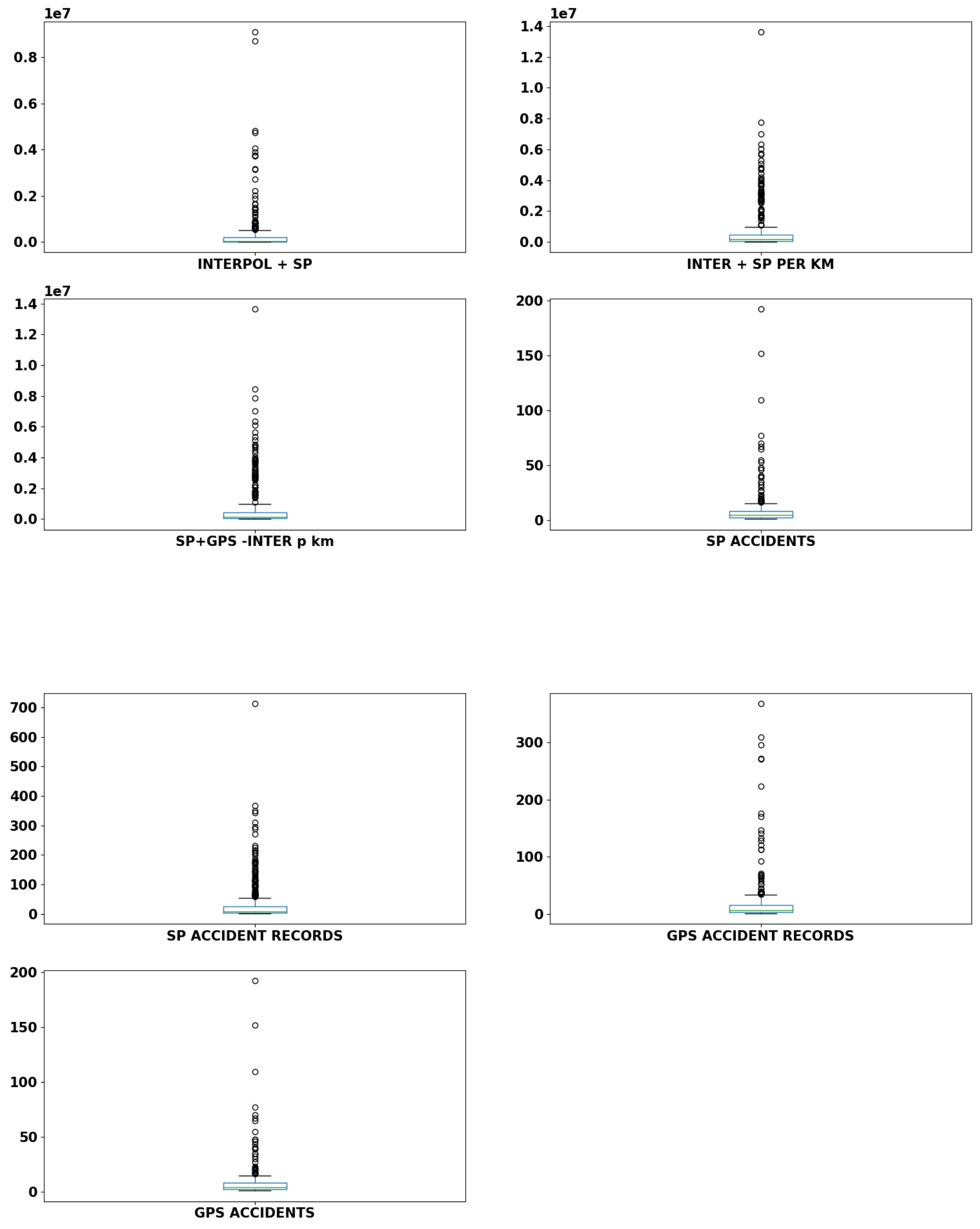


Figure 21: Fisker plots for features of data; SP - speedpanels

NAME	# SEGS	# 50 M SEGS	ID	LENGTH (M)	GPS OBS	GPS P KM
Std	26.21739625	21.90945442	140548037.8	1592.945011	77101.35136	281461.6684
Max	294	234	524508455	16385.61029	752321.25	3885370.813
Min	2	2	4338869	5.372700376	0	0
Average	12.96214896	9.814407814	177370296.3	613.0813136	39498.98168	167922.0129
NAME	GPS INTER	GPS + SP	INTER/OBS	SP/INTER	GPS IN KM	OBS IN KM
Std	586382.7439	643288.0609	1.496475217	93.16944221	281461.6684	1542432.685
Max	9093941.25	9093941.25	12.8834321	2565.676471	3885370.813	18235456.87
Min	0	0	0	0	0	0
Average	114486.7097	172938.7866	1.346403972	7.583817799	167922.0129	576795.1857

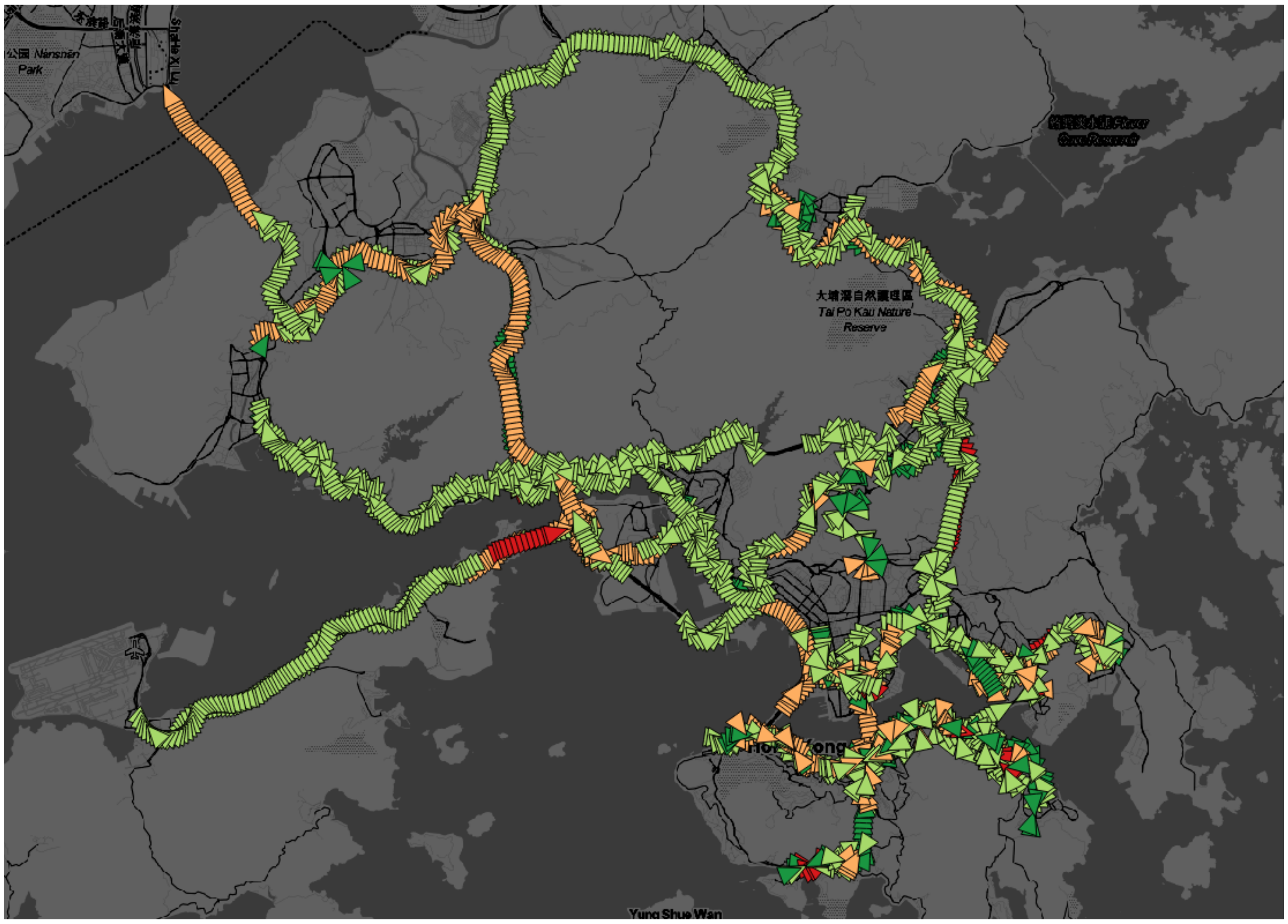
Table 13: Summary of the input dataset; INTER - interpolated; SP - speedpanel observations; OBS in km - SP + GPS

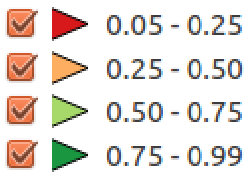
Only GPS data - 25 m GPS record matching boundary

The results for all experiments for different folds varied a lot within a segment. This was due to different distributions of traffic accidents in each fold. To solve this issue, the AUC average was calculated using weights which were determined by the number of accidents in a specific fold.

The average AUC received using only GPS data and not dismissing outliers for 441 ways is 0.5714. The disribution of AUC for each fold can be seen in Figure 23a. In this experiment only window size TW 3600 is visualized because all the window sizes produce same results. The distribution of AUC values on roads over Hong Kong can be seen in Figure 22.

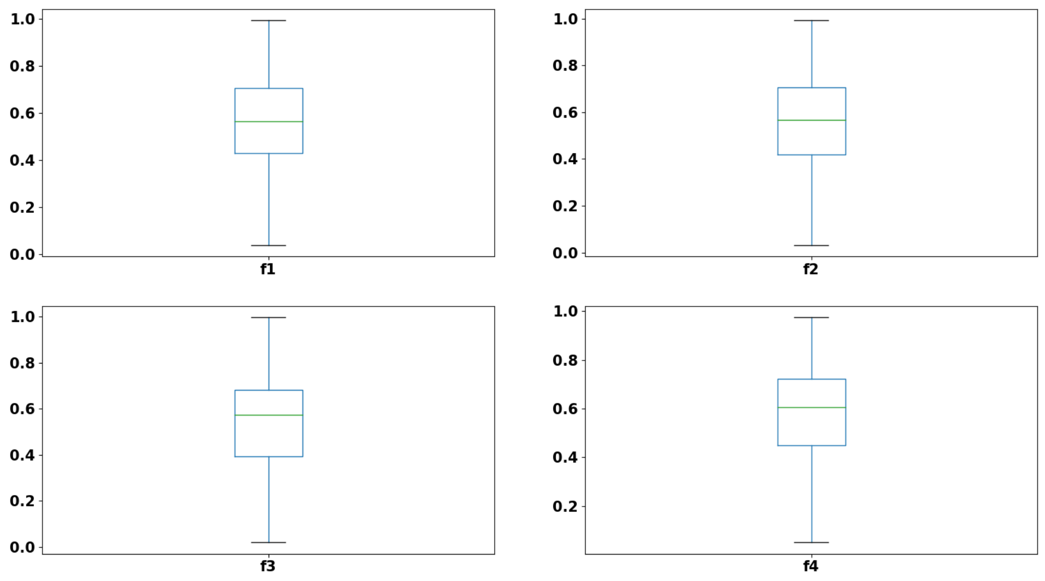
A correlation matrix together with a scatter matrix for the average AUC can be seen in Figure 23b. The correlation is not strong between AUC and any of the variables. However looking at the scatter plots in Figure 24 and 25 for multiple features like ACC/ACC_COUNT, GPS OBS PER KM, there starts to exist a certain correlation with the AUC value after elimination of some bottom end observations. However this kind of relationship is complex and seems to depend on multiple parameters. Thus further research is needed.



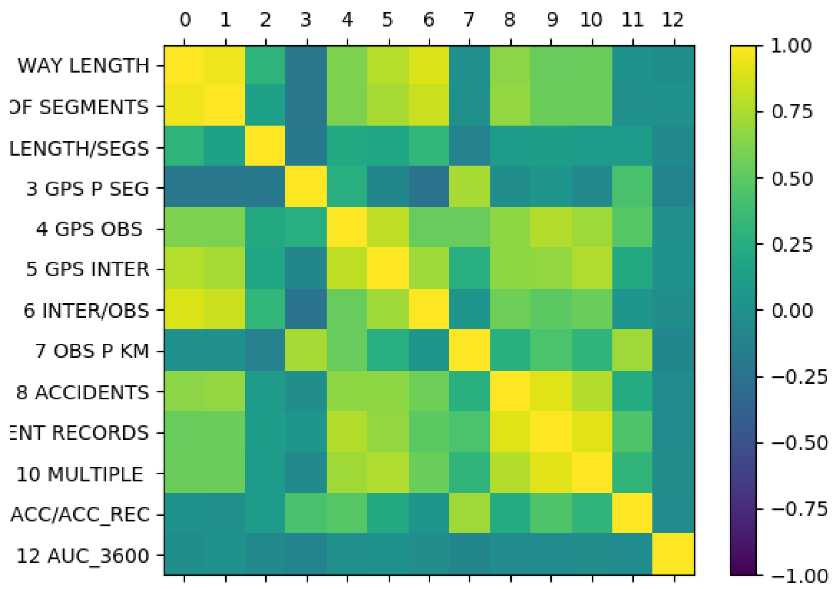


(a) AUC values visualized

Figure 22: The accident detection rate visualised on top of Hong Kong Map for accident detection model based on GPS data only.



(a) Fisker plots for all folds



(b) Correlation between elements of the dataset

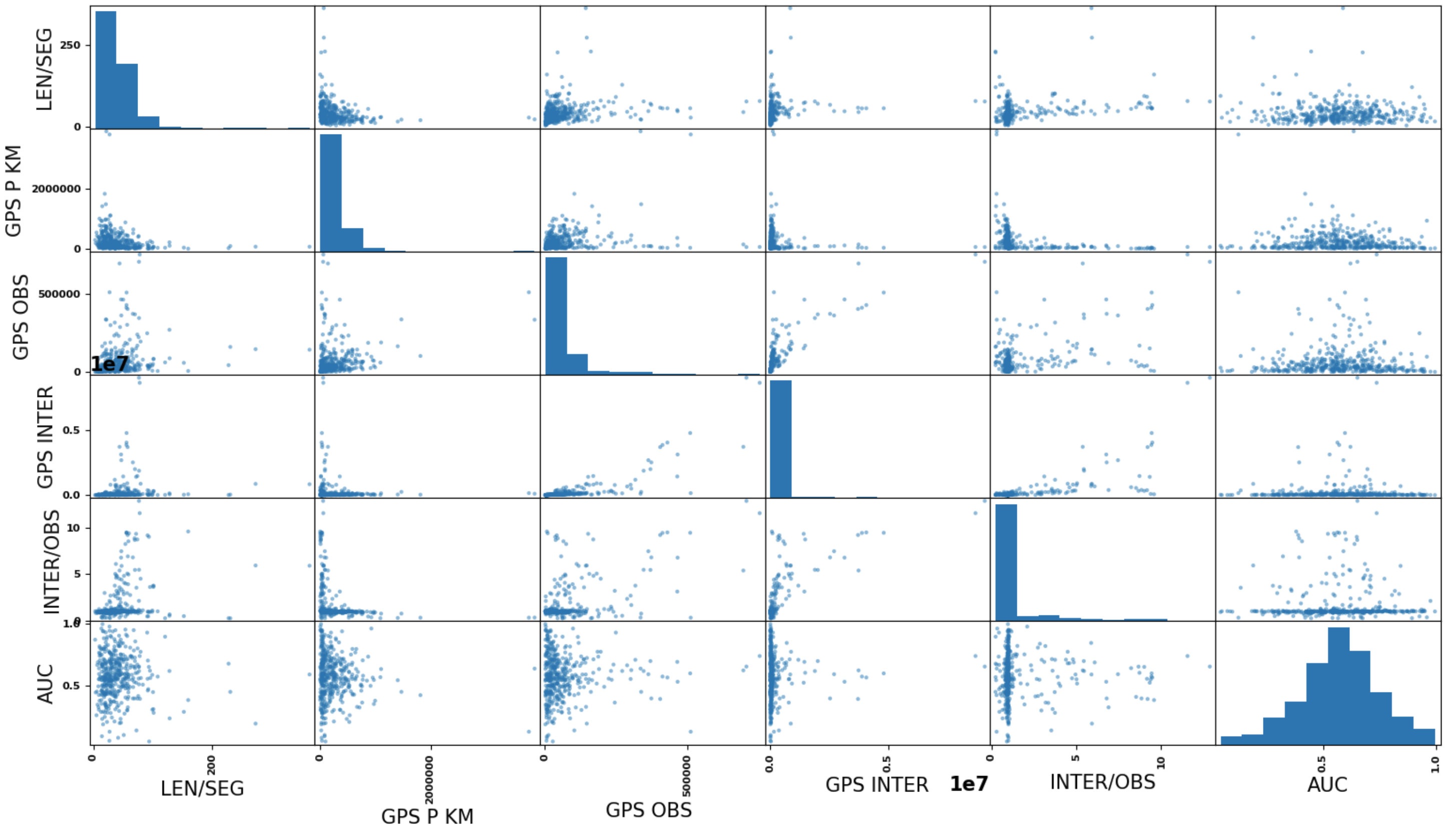


Figure 24: 1e7=10000000

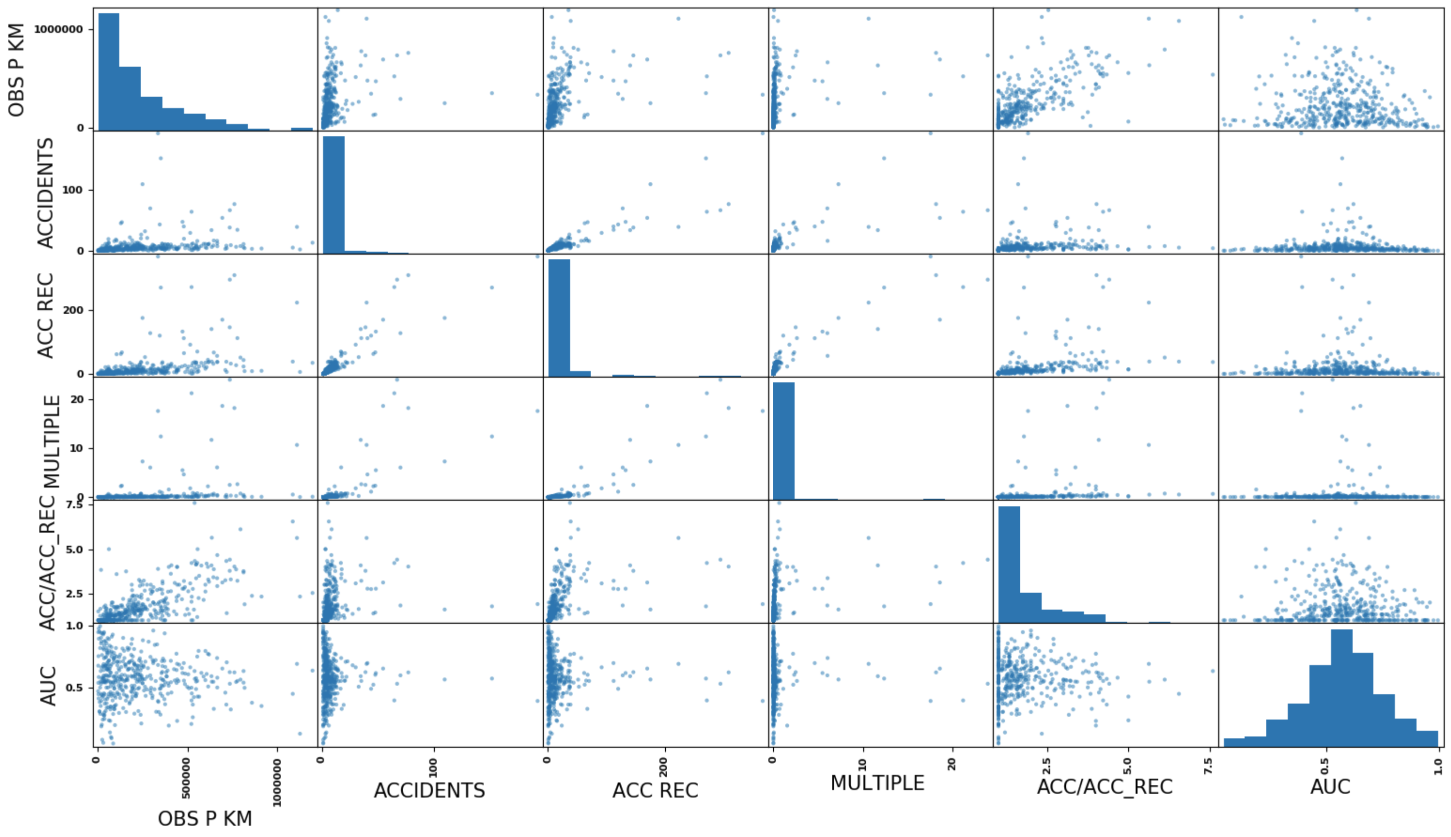
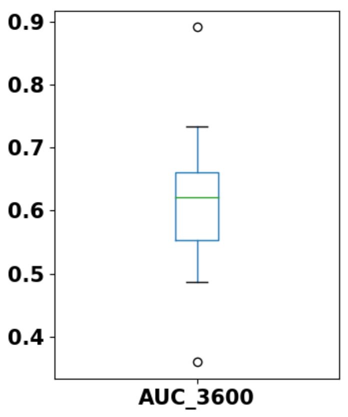


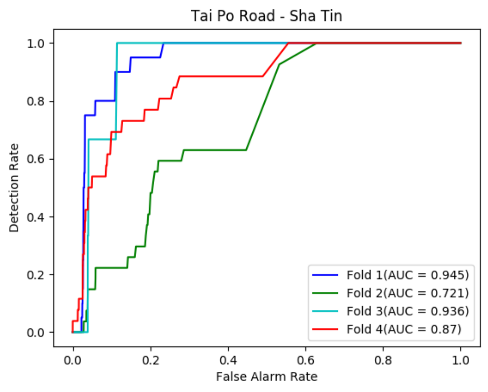
Figure 25: 1e7=10000000

Taking the Japanese dataset as an example, elimination of Ways which contained fewer than 140,000 actual GPS observations (minimum amount of records per 1 km in Japan's dataset) per km and were shorter than 200 m produced an AUC average of 0.62 which can be seen in Figure 26a. This, however, reduced the number of ways to 20. An example of good results in the experiment can be seen in Figure 26

Addition of speedpanel data: Adding speedpanel data resulted in a slightly higher overall result of AUC 0.5743. The fisker plots of each fold can be seen onf Figure 27. The AUC results on roads can be seen in Figure 29. In total 4 ways were added under the analysis compared to the result using only GPS data.



(a) AUC with more than 140 000 GPS observations per km, more than 500 m length ways



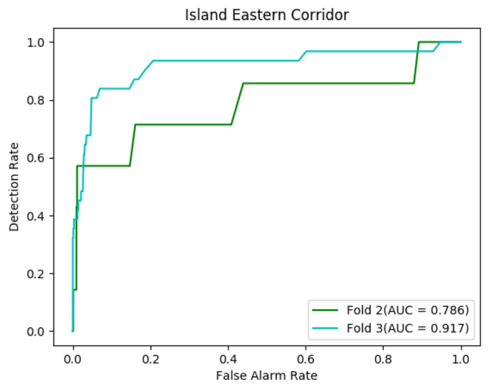


Figure 26: Examples with more than 6 accidents and more than 17 records classified as accidental

In this part there existed no correlation between the data. The correlation matrix can be seen in Figure 28. Scatter matrixes, as they offer a relatively similar result to using only GPS data, are skipped to save room. The AUC initially again does not have any correlation but when ways which have fewer than 140,000 observations or fewer than five accidents with less than two records per accident are filtered a correlation of 0.3 between accident/accident records and the AUC value appears.

The ways which had both speedpanels and GPS data average was 0.6.

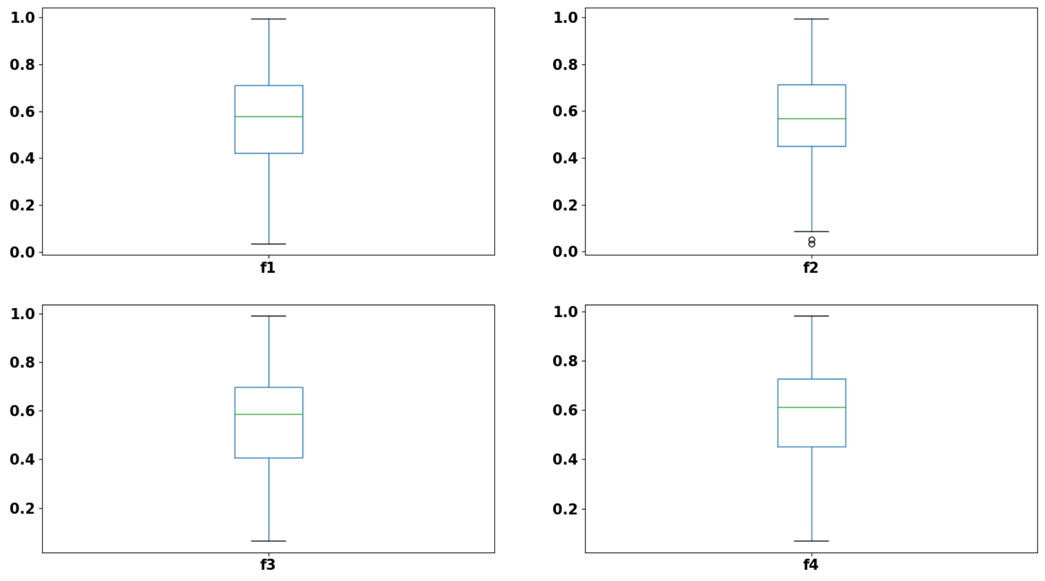


Figure 27: Fisker plots for all folds

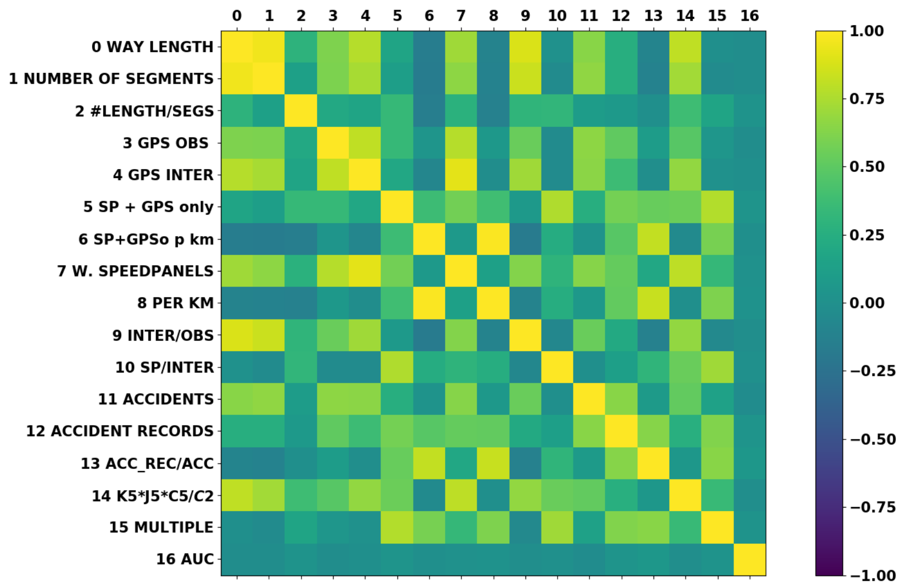
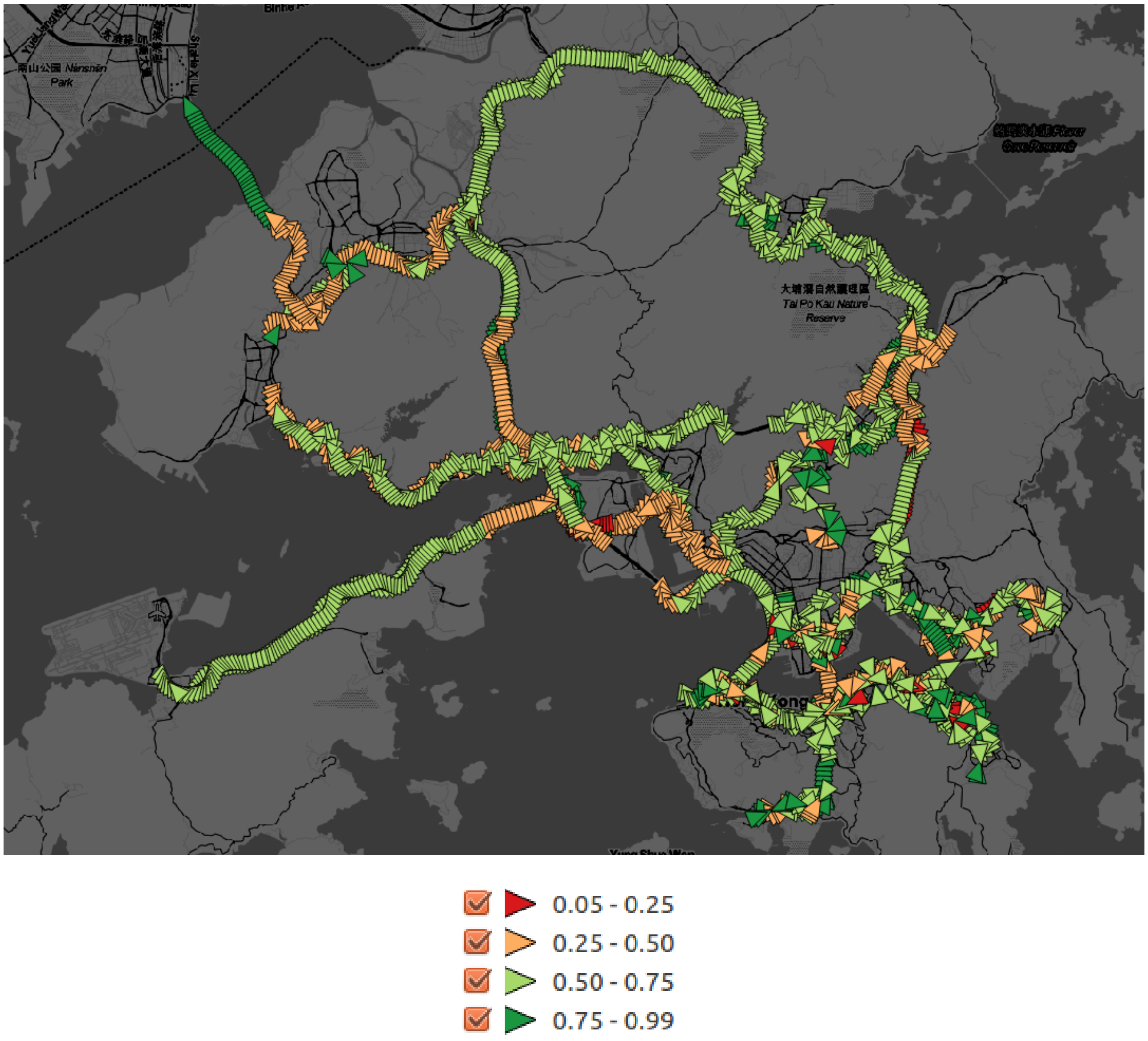


Figure 28: Corelation between elements of the dataset

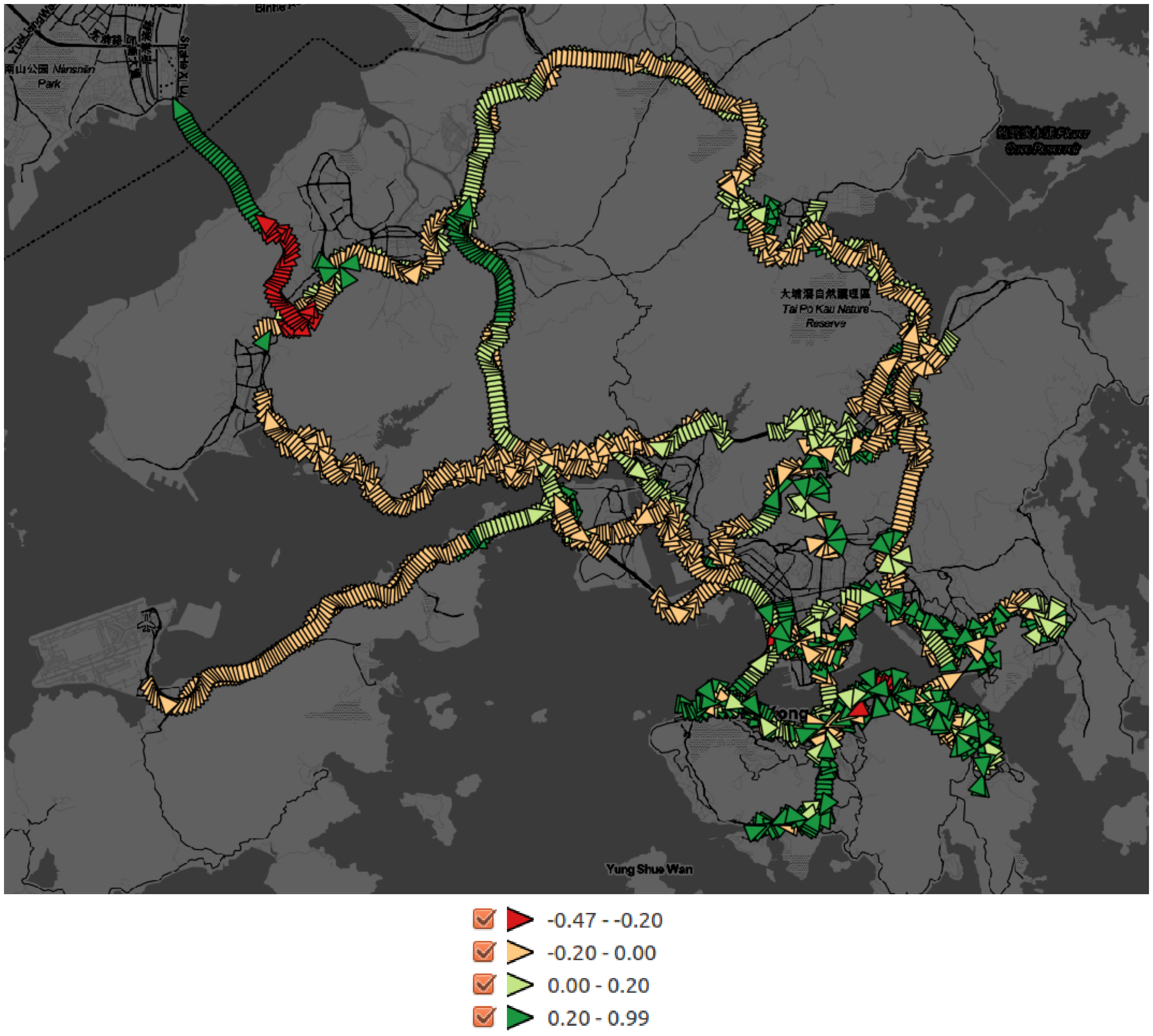
The comparison for experiments with the Speedpanel data and without it is visualised in Figure 30. The speedpanels' locations are visualised in Figure 31. It is clear the results for the dataset using speedpanels are better (in many cases more than 20 percent) in Hong Kong and Kowloon area, i.e. in city-areas, and on the opposite, away from the city areas, the results are better for the model which is based on only GPS data.

The overall distribution of AUC values for all the models can be seen in Histogram 32



(a) AUC values visualised

Figure 29: The accident detection rate visualised on top of Hong Kong map for accident detection model based on GPS and speedpanel data



(a) AUC scales

Figure 30: Comparison of AUC between the experiments with and without the speedpanel dataset. For each way the result was calculated according to this formula: (GPS + SP) AUC - (GPS) AUC.

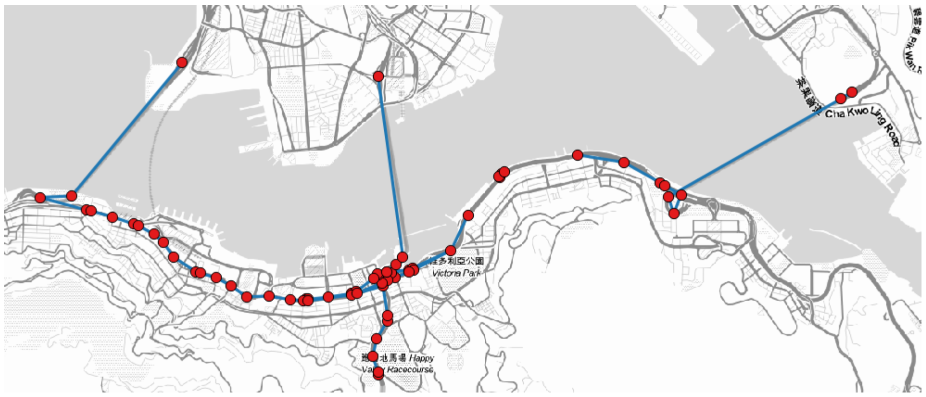


Figure 31: Speedpanel locations in year 2010

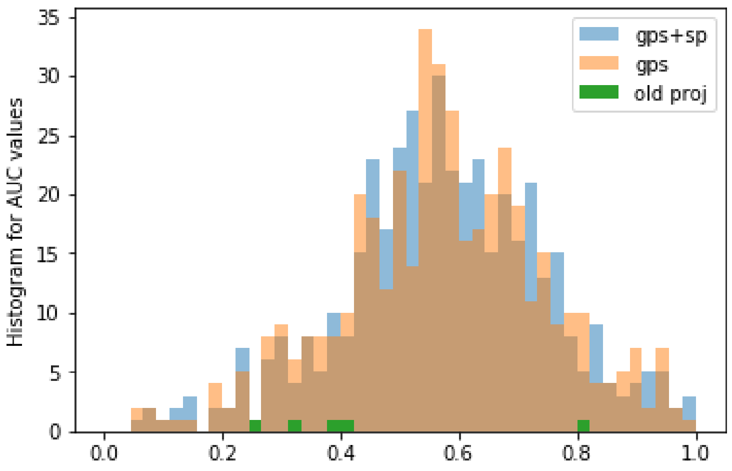


Figure 32: Histogram for results 1. Using the old source code and GPS data 2. Using rewritten source code and the same GPS data 3. using the rewritten source code with GPS and speedpanel data

6 Discussion

Development of a well-documented pipeline

The focus on the development of a data preprocessing pipeline paid off significantly. Altogether four mistakes were detected in the project under development and fixing them improved the accuracy and the results drastically.

In addition, thank to visualization and documentation, it was possible to achieve a much better overview of Incident Detection in Hong Kong.

Dataset effect on the result

Based on the observations of the all of experiments (1-4) it seems the overall amount of data as an input to the model plays a strong role in producing the results. The worst result came from using only the speedpanel dataset which due to lack of interpolation could produce the least amount of data to segments. The second worst results were received with the dataset using only GPS observations assigned to segments longer than 1.5 km and belonging in trajectories longer than 500 m. This filtering done in experiment 3 due to filtering produced less data than the methodology in experiment 1 . Using GPS data with interpolation without the filtering produced the second best result and adding speedpanel data to it produced the best result. 19

The averages of the best result in Hong Kong are still 0.2 less compared to the AUC results received in et. al Kinoshita. This based on the experiments (1-4) conducted on Hong Kong dataset likely results from a smaller amount of data (Figure 18). The results are however not that much worse which the first experiments lead to believe. Comparing these results should also not be taken too seriously as Et. al Kinoshita article considered roadworks as traffic incidents as well. Due to inability to get access the roadworks database in Hong Kong, this paper used traffic accidents recorded by Hong Kong Police for testing the model. This likely introduced additional false results.

Based on the observations on experiments (1-2) the dataset which included speedpanels seems to be performing better in the city area (Figure 30) and the dataset not including speedpanels has a slightly higher result on the longer roads. Considering that there existed no correlation whatsoever between the observation amounts in a specific way and the AUC results in that way, this leads to believe that the global parameters determined the total amount of data in the dataset have very important role to play comparing to the local variables assigned to each segment. This again speaks for a larger amount of data as a possible jumping board forreceiving better results for incident detection.

Comparing the results with the results received in Kinoshita et al. should also not be taken too seriously as that article considered roadworks as traffic incidents as well. Due to inability to get access to the roadworks database in Hong Kong, this paper used only traffic accidents recorded by Hong Kong Police for testing the model. This potentially introduced additional false results.

The results in experiments (1-2) also indicate that the balance of dataset regarding from what kind of roads most of the data is collected from, determines the results on different types of roads. If most of the observations are collected from the city area, the model is going to be more biased and better- performing there. This improvement or imbalance could have a negative effect on other types of roads like highways. This claim, however, needs further research with information which would allow the classification of ways to this project's needs. For example, a future direction of research could be the categorising of the data into specific groups either by clustering similar roads or supervised grouping. E.g. making a difference between a road in Central and a highway in Lantau. This could lead to training more informative traffic states and can also give some information on where topic modeling does well and where it does not. Categorisation of the data based on time as is another consideration. Data, when it's some holiday, is likely different from regular weekday data. Taking this into account could reduce false negatives.

Both the issue with the size of the dataset and the imbalance of observations between the city area and highways should be possible to improve by taking advantage of the fact that in the year 2013,as many as 600 new speedpanels were added all over the area of Hong Kong compared to the 62 speedpanels in the year 2010. Using the data generated from multiple years in a row could produce enough data to train this model to successfully predict traffic accidents.

As the speedpanels are not exactly placed the same as the segments to which GPS data is matched to, specific methodology in regard to the spatial effect of speedpanels needs to be developed. As the available road sensor data is based on averages, a decision methodology on how many GPS observations one speedpanel observation equals to at a specific time and location needs to be developed.

Trained model

Currently the proposed model estimates the parameters in all the locations as point estimates. Although the exact estimation of parameters gives the same results as when considering the parameters as random variables coming from distributions, it does not give the ability to assess the strengths of predictions in each of the segments. This is without a reason because a lot of authors have proposed methods how to integrate parameter inference to the EM algorithm [1] [5]. In addition, the predicition capabilities could also be assessed using regular statistical reliability methods like p-value and trust intervals to give information on how trustworthy each assessment is based on the underlying data.

Addition of parameter assessment can be used to further develop the Hong Kong traffic monitoring system to take advantage of traffic monitoring using probabalistic graphs and source to target queries on top of such graphs [15].

Essentially when speed observations are the only input to the model, the training deals with finding an average λ for each of the K traffic states and then mixing these averages together on each road. Based on that, the choice of Poisson distribution is suprising because it is distribution which gives a probability to a number of occurences of a specific situation, not the probability of the occurence itself. Here a Gaussian distribution with two parameters instead of one for the Poisson distribution would be naturally more suitable because for such data the standard deviation is also an important consideration.This is what the Poisson Mixture model completely misses. Using Gaussian Mixtures together with the EM algorithm is a standard pratice in machine learning and both point estimate and inference methods have been developed for it [1] [2].

The methods in traffic accident detection can broadly be divided into rule-based methods: methods which have defined the characteristics of traffic accidents manually [36] [40] [18] an methods which try to use supervised learning to differ traffic accidents based on some chosen features [36] [31] [16]. In addition , methods have been developed which statistically model a normal condition for traffic and perform anomaly detection [38]. The method used in this dissertation falls into the third category method of modeling a normal traffic state from which anomalies are detected. Although this method is aimed at modeling a normal traffic state for each of the road segments, it fails to model traffic accidents information as its purpose is only to detect divergence from that modeled normal state. Considering for example that in Hong Kong there happened 14,000 accidents in a year, this is done without justification as traffic accident patterns could provide the model with valuable information. Considering that using speedpanels it was possible to obtain around 14 thousand accident records for 3 thousand accidents, addition of an accident model to this algorithm can prove to be useful. The current model only takes into account the traffic state in a certain time and does not consider the traffic state change into its model. The sudden change of traffic definitely is a likely indicator of a traffic accident which this model completely misses. Wang et al. has proposed a Neural Network and Time Series Analysis based solution which models traffic accidents and also takes into account the time factor. That solution has only been tested using loop detectors and has not shown AUC values in their results. Overall, based on the quantity and divergence of methods performing incident detection in the literature, and a lack of papers comparing the results on one specific dataset, the author of this dissertation strongly feels that an experimental study comparing the latest state-of-the-art methods for incident detection on one dataset could proce to be very useful.

Topic modeling is based on co-ccurence of observations in certain clusters. In Natural Language Processing (NLP), LDA model assumes a co-occurrence of similar words on one specific topic and assigns for each word a topic based on systematic co-occurences. It is possible to easily analyse whether the classified topics actually are good or not based on the words inside the topics. If the words ”pollution” and ”playground” have been classified to act together we can clearly say that it is not performing well. An example of good co-occurence is the words ”marine” and ”pollution” co-occuring in a topic. For NLP this kind of extra grouping of words to topics is more important because the words do not have a natural linear numerical basis to be grouped together. Speed observations however are already grouped together on a numerical scale. It is more likely that observations 71 and 72 occur together than 71 and 10. Considering the capapility of the LDA, however, extending this model to using multiple features could improve the model with less simplistic connections by creating traffic states based on non-linear features. Such extra information could be useful for cases like in Hong Kong, where the dataset is relatively small. The addition of extra traffic features has been proposed by Kinoshita et al. in their paper but they have not implemented the solution for this.

Another approach to solving the small datasets issue is the use of social media data . Hidden Markov Models have been under extensive research to model PCD data, specially taxi data separately [28] [19] [14]. Wang et al. [38] proposed an extended Coupled Hidden Markov Model to integrate GPS probe data and traffic related tweets to estimate traffic conditions of the arterial network, and used a sequential importance sampling based EM algorithm to learn the parameters. This paper showed a high-level correlation of predicted traffic anomalies and social media findings and was able to get high prediction rates on Chicago data. Currently some data has been collected from Hong Kong radios, Government news and Facebook to get additional textual information from the social media. This could increase the incident detection capability in Hong Kong when integrated to the incident detection model.

7 Summary

This dissertation implemented a real time automatic incident detection system in Hong Kong based on road sensor and GPS data. It took over a source code for the incident detection and fixed four larger mistakes in it which affected the incident detection result to a very large extent. The source code was redesigned in all the phases of data preprocessing for optimisation and a more clear architecture. In addition, automatical and visual-based testing and analysis was added to get a better overview of each phase results. The source code was redesigned so it would be compatible with the OSM format. In addition, documentation was written for the source code. To take advantage of the specific data in Hong Kong, research was done on the available datasets and based on that, road sensor data was integrated into the incident detection project. Fixing the mistakes in the source code and addition of road sensor data improved the incident detection performance drastically. In addition, parameter tuning was performed to find the best solution for Hong Kong. Experiments were performed with multiple different parameter and data sets. The results received were analysed with numerous different methods. Based on the experiments, multiple ideas for improving the incident detection capability in Hong Kong were proposed.

References

[1] *Theory and Use of the EM Algorithm*, chapter 1. Foundations and Trends in Signal Processing, 2010.

[2] *CS229 Lecture notes*, chapter 9. Stanford University Press, 2011.

[3] Osm map formats, 2017.

[4] M. Balazinska, B. Howe, and D. Suciu. Data markets in the cloud: An opportunity for the database community. *Proc. of the VLDB Endowment*, 4(12):1482–1485, 2011.

[5] C. M. Bishop. *Pattern recognition and machine learning*, chapter 9.1-9.4. springer, 2006.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[7] P.-T. Chen, F. Chen, and Z. Qian. Road traffic congestion monitoring in social media with hinge-loss markov random fields. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 80–89. IEEE, 2014.

[8] X. Chen, X. Hu, X. Shen, and G. Rosen. Probabilistic topic modeling for genomic data interpretation. In *Bioinformatics and Biomedicine (BIBM), 2010 IEEE International Conference on*, pages 149–152. IEEE, 2010.

[9] Y. Cheng, M. Zhang, and D. Yang. Automatic incident detection for urban expressways based on segment traffic flow density. *Journal of Intelligent Transportation Systems*, 19(2):205–213, 2015.

[10] H. K. T. Department. Special traffic news, 2010-2017.

[11] H. K. T. Department. Traffic speed panes, 2017.

[12] H. K. Government. Conversion tool for coordinates, 2016.

[13] B. Guangtong. Automatic incident detection, 2016.

[14] R. Herring, A. Hofleitner, P. Abbeel, and A. Bayen. Estimating arterial traffic conditions using sparse probe data. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 929–936. IEEE, 2010.

[15] M. Hua and J. Pei. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 347–358. ACM, 2010.

[16] X. Jin, D. Srinivasan, and R. L. Cheu. Classification of freeway traffic patterns for incident detection using constructive probabilistic neural networks. *IEEE Transactions on Neural networks*, 12(5):1173–1187, 2001.

[17] Y. Jin, J. Dai, and C.-T. Lu. Spatial-temporal data mining in traffic incident detection. In *Proc. SIAM DM 2006 Workshop on Spatial Data Mining*, volume 5, 2006.

[18] A. Kinoshita, A. Takasu, and J. Adachi. Real-time traffic incident detection using a probabilistic topic model. *Information Systems*, 54:169–188, 2015.

[19] J. Kwon and K. Murphy. Modeling freeway traffic with coupled hmms. Technical report, Technical report, Univ. California, Berkeley, 2000.

[20] Z. Liao, Y. Yu, and B. Chen. Anomaly detection in gps data based on visual analytics. In *Visual Analytics Science and Technology (VAST), 2010 IEEE Symposium on*, pages 51–58. IEEE, 2010.

[21] J. Lu, S. Chen, W. Wang, and H. van Zuylen. A hybrid model of partial least squares and neural network for traffic incident detection. *Expert Systems with Applications*, 39(5):4775–4784, 2012.

[22] J. P. Mueller and L. Massaron. *Machine Learning for Dummies.* John Wiley & Sons, 2016.

[23] A. Ng. Learning curves, 2011.

[24] G. of the Hong Kong Special Administrative Region. Information portal of government of the hong kong special administrative region.

[25] G. of the Hong Kong Special Administrative Region. Speedmap data.

[26] G. of the Hong Kong Special Administrative Region. Government mobile applications, 2017.

[27] G. Press. Cleaning big data: Most time-consuming, least enjoyable data science task, survey says, forbes, 2016.

[28] Y. Qi and S. Ishak. A hidden markov model for short term prediction of traffic conditions on freeways. *Transportation Research Part C: Emerging Technologies*, 43:95–111, 2014.

[29] H. K. C. Radio. Traffic incident news, 2010-2017.

[30] I. Scholz. Java openstreetmap editor, 2017.

[31] D. Srinivasan, X. Jin, and R. L. Cheu. Adaptive neural network models for automatic incident detection on freeways. *Neurocomputing*, 64:473–496, 2005.

[32] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[33] M. D. Team. Mapzen, 2017.

[34] Q. D. Team. Qgis project, 2017.

[35] C. Wang and D. M. Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 448–456. ACM, 2011.

[36] J. Wang, X. Li, S. S. Liao, and Z. Hua. A hybrid approach for automatic incident detection. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1176–1185, 2013.

[37] S. Wang, L. He, L. Stenneth, S. Y. Philip, Z. Li, and Z. Huang. Estimating urban traffic congestions with multi-sourced data. In *Mobile Data Management (MDM), 2016 17th IEEE International Conference on*, volume 1, pages 82–91. IEEE, 2016.

[38] S. Wang, F. Li, L. Stenneth, and S. Y. Philip. Enhancing traffic congestion estimation with social media by coupled hidden markov model. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 247–264. Springer, 2016.

[39] J. Yuan, Y. Zheng, and X. Xie. Discovering regions of different functions in a city using human mobility and pois. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 186–194. ACM, 2012.

[40] M. K. c. N. O. d. Yusuke HARA a, Koji MATSUDA b and K. INUI. Estimating traffic states and identifying their causes and effects through probe and social media data analysis. In *HKTS*, 2016.

[41] T. Zhu, J. Wang, and W. Lv. Outlier mining based automatic incident detection on urban arterial road. In *Proceedings of the 6th International Conference on Mobile Technology, Application & Systems*, page 29. ACM, 2009.